UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM DESENVOLVIMENTO
DE APLICATIVOS MÓVEIS E INTERNET DAS COISAS

SÉRGIO HIROSHI NONOMURA

FISCALIZAÇÃO DE OBRAS ATRAVÉS DE APLICATIVO MÓVEL

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2020

SÉRGIO HIROSHI NONOMURA

FISCALIZAÇÃO DE OBRAS ATRAVÉS DE APLICATIVO MÓVEL

Monografia apresentada para obtenção do título de Especialista no Curso de Pós-Graduação em Desenvolvimento de Aplicativos Móveis e Internet das Coisas, Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná - UTFPR.

Orientador: Professor Leandro Batista de Almeida.

CURITIBA
2020

TERMO DE APROVAÇÃO

FISCALIZAÇÃO DE OBRAS ATRAVÉS DE APLICATIVO MÓVEL

por

SÉRGIO HIROSHI NONOMURA

Esta monografia foi apresentada em 17 de Dezembro de 2020 como requisito parcial para obtenção do título de Especialista em Desenvolvimento de Aplicativos Móveis e Internet das Coisas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Leandro Batista de Almeida
Orientador

_____
Prof. Dr. Maria Claudia Figueiredo Pereira Emer
Membro titular

_____
Prof. Dr. Adriano Francisco Ronszcka
Membro titular

O Termo de Aprovação assinado encontra-se na Coordenação do Curso.

AGRADECIMENTOS

À minha família, aos meus colegas e professores do Curso. Em especial, ao meu Orientador Prof. Leandro Batista de Almeida.Ao Banco do Brasil que possibilitou a minha participação neste Curso.

RESUMO

NONOMURA, Sérgio Hiroshi. **Fiscalização de Obras Através de Aplicativo Móvel**. 2020. 121 p. Monografia de Especialização em Desenvolvimento de Aplicativos Móveis e Internet das Coisas, Departamento Acadêmico de Informática, Universidade Tecnológica do Paraná. Curitiba, 2020.

Atualmente, a informação está disponível e acessível para todos com o advento da internet. As plataformas web e posteriormente as plataformas iPhone Operational System (iOS) e Android®, passaram a fornecer meios computacionais para desenvolver e executar aplicativos como um serviço via web, em vez de instalar as ferramentas no próprio computador. O Web Service, através das Application Programming Interface (API), como forma de integração e comunicação de sistemas, permite que um sistema realize uma chamada para um serviço de outro sistema a fim de obter informações de uma forma padronizada e independente de linguagem de programação. A computação em nuvem permite que soluções como o Firebase Realtime Database acessem os bancos de dados diretamente de um dispositivo móvel sem a necessidade de uma aplicação do lado do servidor (server-side). Com o surgimento de novas tecnologias, o processo de fiscalização de obras deve acompanhar tal evolução de forma a dar agilidade, confiabilidade, eficiência e eficácia aos serviços de Engenharia. O que se propõe neste estudo é desenvolver um protótipo de aplicativo Android® que acesse dados do Firebase Realtime Database e grave/altere os dados das Ordens de Serviço de Fiscal de Serviço responsável pela fiscalização de obras de agências bancárias. Desenvolvido o protótipo de aplicativo, obteve-se como resultado o acesso e gravação de dados em nuvem.

Palavras-chave: Firebase. Realtime Database. Fiscalização. Obras. Aplicativo Móvel.

ABSTRACT

NONOMURA, Sérgio Hiroshi. **Fiscalização de Obras Através de Aplicativo Móvel**. 2020. 121 p. Monografia de Especialização em Desenvolvimento de Aplicativos Móveis e Internet das Coisas, Departamento Acadêmico de Informática, Universidade Tecnológica do Paraná. Curitiba, 2020.

Currently, information is available and accessible to everyone with the advent of the internet. The web platforms and later the iPhone Operational System (iOS) and Android® platforms started to provide computational means to develop and run applications as a service via the web, instead of installing the tools on the computer itself. The Web Service, through the Application Programming Interface (API), as a way of integrating and communicating systems, allows a system to make a call to a service from another system in order to obtain information in a standardized way and independent of the programming language. . Cloud computing allows solutions such as Firebase Realtime Database to access databases directly from a mobile device without the need for a server-side application. With the emergence of new technologies, the construction inspection process must accompany such evolution in order to provide agility, reliability, efficiency and effectiveness to the engineering services. The purpose of this study is to develop a prototype of an Android® application that accesses data from the Firebase Realtime Database and records / changes the data of the Service Tax Service Orders responsible for the inspection of bank branch works. Once the application prototype was developed, the result was access and recording of data in the cloud.

Keywords: Firebase. Realtime Database. Supervision. Construction. Mobile App.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
App	*Application*
CONFEA	Conselho Federal de Engenharia e Agronomia
HTTP	*Hypertext Transfer Protocol*
iOS	*iPhone Operacional System*
IRB	Instituto Rui Barbosa
JSON	*JavaScript Object Notation*
OS	Ordem de Serviço
PaaS	*Platform as a service*
SDK	*Software Development Kit*
SOAP	*Simple Object Access Protocol*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
UDDI	*Universal Description, Discovery, and Integration tool*
URI	Uniform Resource Identifier
XML	Extensible Markup Language
WSDL	Web Services Description Language

SUMÁRIO

1 INTRODUÇÃO

A coleta de dados é uma atividade rotineira para diversos profissionais. Pesquisadores utilizam formulários para coletar dados de pesquisas. Vendedores utilizam formulários de contato para conhecer os seus clientes. Agricultores criam formulários para acompanhar o resultado de sua produtividade (COLETUM, 2020).

Não só a coleta de dados, mas o acesso e atualização de dados de forma remota são necessários para a maioria das pessoas, profissionais ou não. Seja por meio do Twitter®, Whatsapp®, Facebook® ou aplicativo móvel desenvolvido por uma empresa na qual se trabalha.

"As plataformas digitais com versões móveis são ferramentas muito versáteis. Além de possibilitarem o acesso a qualquer hora e em qualquer lugar, oferecem diversas funcionalidades [...]." (COLETUM, 2020).

Muitas das vezes, profissionais que visitam clientes, realizam vistorias e fiscalizações de obra, necessitam se locomover para lugares distantes demandando um tempo relativamente longo. Ao mesmo tempo, nesse período, suas demandas, pedidos, relatórios, análise de conformidade de pagamentos vão se acumulando e ficam sem dar encaminhamentos gerando problemas de atrasos, perda de eficiência e prejuízos monetários.

O uso de tecnologia móvel para o acesso e registro de dados produzidos se mostra adequado para otimizar etapas de organização e processamento de informações; favorecer gestão do fluxo de dados e oferecer segurança e agilidade de forma a serem disponibilizados rapidamente.

Diante do exposto, tem-se como tema de pesquisa, os procedimentos realizados para desenvolver uma ferramenta que possa suprir as necessidades de tais profissionais.

1.1 TECNOLOGIAS UTILIZADAS NA FISCALIZAÇÃO DE OBRAS DE ENGENHARIA

O Instituto Rui Barbosa (IRB) cita que fiscalizar é uma das principais ações na gestão de obras públicas. Prova disso é a Resolução nº 1010, do Conselho Federal de Engenharia e Agronomia (CONFEA) que versa sobre a

verificação e acompanhamento de execução de obras ou serviços no tocante ao cumprimento às especificações e aos prazos estabelecidos (IRBCONTAS, 2020).

No Brasil, há diferentes cenários que ainda passam pelos profissionais que atuam de forma convencional ou analógica - com as medições e anotações em pranchetas, mas há, também, os que fazem uso de recursos tecnológicos a favor dos resultados. Com a utilização da tecnologia na fiscalização de obras, ela confere precisão na gestão financeira da obra, facilita o processo de acompanhamento e pagamentos, realiza medições em tempo real de forma a permitir realizar comparativos de produtividade, eficiência e agilidade ao trabalho (IRBCONTAS, 2020). Qualquer empresa que seja selecionada para efetuar a fiscalização de obras e serviços de engenharia deve observar a qualidade técnica dos profissionais, análise do portifólio e eficiência no cumprimento de prazos e metas propostas. E o mais importante, que a economia seja bem maior que os gastos efetivamente realizados (ARAUJO, 2020).

A SMARTQUESTION (2020) fornece um tipo de serviço que parece ser semelhante ao que este protótipo pretende desenvolver. Informa prover uma interface web para realizar a criação de pesquisas, planejamento das visitas e análise de resultados. Para executar as pesquisas em campo, informa que é utilizado um Mobile Android que trabalha de forma offline, garantindo que mesmo em locais sem rede de dados, as pesquisas de dados poderão ser realizadas. Ainda, que o sistema ao detectar uma conexão com a Internet, o envio dos dados é realizado automaticamente (SMARTQUESTION, 2020). Cada dispositivo móvel pode realizar quantas pesquisas forem necessários, inclusive com o uso de vários dispositivos ao mesmo tempo (SMARTQUESTION, 2020).

Várias propostas foram levantadas e poucas foram desenvolvidas na forma de utilização que propõe este trabalho. Tais propostas de aplicativos móveis trataram sobre:

- visualização de dados EEG e PDC - exame para diagnóstico de patologias (ANDRADE, 2013);
- localização de estabelecimento que estão em horário de funcionamento (DAPPER, 2017);
- localização de restaurantes (WILL, 2017);

- treinamento pedagógico visual de pacientes com TEA-Transtorno do Espectro do Autismo (MACHADO, 2018);
- aplicativo de games (FILHO, 2012);
- acesso a ambiente de aprendizagem virtual (DALL'OGLIO, 2013);
- usuários de transporte público (LUCIO, 2011);
- cadastramento e classificação de áreas potencialmente contaminadas (FAGUNDES et al., 2015).

O que se percebe diante das pesquisas realizadas, é que as tecnologias digitais e remotas estão se tornando cada vez mais importantes para se obter uma melhor eficiência e qualidade dos serviços fiscalizados.

1.2 OBJETIVOS

O próposito deste trabalho é satisfazer as necessidades dos vários profissionais que necessitem acessar e registrar, de forma remota, dados disponibilizados em nuvem utilizando um aplicativo móvel.

1.2.1 Objetivo Geral

Objetiva-se desenvolver uma ferramenta (protótipo) capaz de auxiliar fiscais de serviços, auditores fiscais e coletores de dados: a realizarem visitas em obras de engenharia de forma eficiente e rápida; eliminar a necessidade de execução de relatórios; agilizar deferimentos/indeferimentos de pagamentos, análises de conformidade de serviços prestados; e a melhoria de processos como um todo.

1.2.2 Objetivo Específico

Especificamente, propõe-se:

- desenvolver um protótipo de aplicativo para fiscalização de obras de engenharia;
- utilizar base de dados NoSQL via plataforma Firebase RealtimeDatabase;
- utilizar a plataforma Android;

1.3 ESTRUTURA

O restante dos capítulos deste trabalho são descritos como segue: a Seção 2 Fundamentação Teórica e Estado da Arte apresenta assuntos teóricos que embasam todo o projeto desenvolvido e que serão utilizados na Metodologia e Desenvolvimento; a Seção 3 Metodologia descreve as etapas gerais do projeto; a Seção 4 Desenvolvimento explana o projeto do software e suas funcionalidades; a Seção 5 Conclusão descreve o resultado alcançado, as aplicabilidades e vantagens na utilização do protótipo de aplicativo, as dificuldades encontradas e sugestões de melhorias; a Seção Referências lista de forma ordenada, as fontes dos fundamentos que embasaram o desenvolvimento deste trabalho; a Seção 7 trata das etapas de execução do protótipo de aplicativo e a Seção 8 apresenta a codificação do protótipo de aplicativo propriamente dito.

2 FUNDAMENTAÇÃO TEÓRICA E ESTADO DA ARTE

A construção de web services e computação em nuvem são pilares importantes no mundo do desenvolvimento de sistemas e um desafio para qualquer desenvolvedor de software se posicionar bem no mercado. Web Services são utilizados como forma de integração e comunicação de sistemas de modo que um sistema possa realizar uma chamada para um serviço de outro sistema a fim de obter informações (LECHETA, 2015).

Existem alguns componentes básicos dos Web Services que facilitam a identificação e a aplicação a aplicativos móveis. O Simple Object Access Protocol (SOAP), a ferramenta Universal Description, Discovery e Integration (UDDI) e a Web Services Description Language (WSDL) são os principais componentes de um serviço da web que o torna disponível para uso. Esses recursos ajudam o aplicativo a se comunicar e ser executado em qualquer serviço baseado na web, como um navegador da Internet ou dispositivo baseado na web (GOODFIRMS, 2020).

São frequentemente categorizados como desenvolvimento de back-end, oferecendo um framework que funciona no desenvolvimento de aplicativos móveis. Outros tipos de aplicativos que não são baseados em dispositivos móveis têm um longo histórico de estruturas e recursos que ajudam a simplificá-los, mas no mundo dos aplicativos móveis, ainda existem muitos desenvolvimentos em fases primárias. É por isso que o surgimento dos Web Services é tão importante para o futuro dos aplicativos móveis (GOODFIRMS, 2020). E o Application Programming Interface (API) é utilizado para consumir quaisquer Web Services (ECLIPSE, 2020).

Quanto às APIs, a RED HAT (2020a) comenta, nenhuma aplicação consegue gerar valor corporativo sozinha, pelo menos não por muito tempo. Para atender os avanços tecnológicos de forma competitiva, as aplicações devem estar conectadas aos investimentos tecnológicos e futuros de uma empresa proporcionando aos clientes novas experiências de forma rápida. A Red Hat (2020a) indaga: para que serve uma aplicação nova se ela não estiver bem integrada aos investimentos legados. Ainda, ao mesmo tempo, questiona como é possível implementar a segurança de APIs de maneira proativa, flexível e consistente. (RED HAT, 2020a).

As APIs bem projetadas e gerenciadas interconectam novas soluções e serviços das empresas rapidamente e permitem que mantenham o controle e a segurança. Através daquelas torna-se possível reutilizar recursos e ser mais versátil ao projetar novas ferramentas e soluções (RED HAT, 2020a).

Os aplicativos móveis estão continuamente evoluindo e mudando como resultado das necessidades crescentes dos usuários de recursos de conectividade e Web Services. Em breve, poder-se-á dizer que, os Web Services são a parte do servidor que trata os dados para aplicativos móveis (GOODFIRMS, 2020).

2.1 PROTÓTIPOS

Segundo OLIVEIRA (2020), é um modelo criado para testar um produto ou sistema. O seu objetivo é o teste e o aprendizado. Ele é construído para saber se uma coisa pode dar certo ou não, por isso todos os recursos devem ser aplicados de forma a responder a pergunta: "Estamos certos ou não?". Ele deve entregar valor, isto é, muitas funcionalidades completas do que muitos recursos inacabados. Não precisa estar completo para ser testado. Ele nasce para ser testado e descartado. O projeto pode ter várias funcionalidades, mas nesse deve ser incluído as mais importantes. Somente após testado, deve ser implementado as melhorias.

2.2 COMPUTAÇÃO EM NUVEM

A computação em nuvem permite à empresa ou desenvolvedor estar totalmente no controle e gerenciar o seu próprio datacenter de forma simples e ágil. É um modelo que permite um pool compartilhado de recursos computacionais configuráveis (redes, servidores, armazanamento, aplicações e serviços) que podem ser rapidamente fornecidos e disponibilizados com um mínimo esforço de gestão ou interação com o fornecedor do serviço. A principal vantagem da nuvem é que não necessita preocupar-se com recursos de Tecnologia da Informação (TI) e focar apenas no desenvolvimento das aplicações (LECHETA, 2015).

O exemplo mais clássico de computação na nuvem é armazenar arquivos na internet, mas segundo a NIST (National Institute of Standarts Technology)

trata-se de algo mais amplo, pois pode-se acessar qualquer recurso de Tecnologia da Informação (TI) sob demanda. Por exemplo, criar vários servidores na nuvem e acessar serviços computacionais para auxiliar a desenvolver aplicações, aumentar o espaço de armazenamento, melhorar a capacidade computacional de seus computadores ou até adicionar mais servidores. Tudo isso, pode ser feito manualmente de forma remota, automaticamente ou até por uma API, utilizando uma linguagem de programação. Outro conceito importante da nuvem é o pagamento conforme demanda, ou seja, paga-se conforme utiliza-se os serviços (LECHETA, 2015).

2.3 API

API é um acronônimo em inglês que significa Application Programming Interface (interface de programação de aplicações) que se traduz para um conjunto de definições e protocolos utilizado no desenvolvimento e na integração de software de aplicações (RED HAT, 2020b).

As Application Programming Interface (API) surgiram nos primeiros dias da computação, mesmo antes do computador pessoal. Elas eram normalmente utilizadas como bibliotecas para sistemas operacionais. Embora a API enviasse mensagens entre mainframes, tais mensagens eram quase sempre locais aos sistemas em que operava. Depois de quase 30 anos, as APIs se expandiram para além dos ambientes locais. No início dos anos 2000, elas passaram a se tornar uma tecnologia importante para a integração remota de dados (RED HAT, 2020b).

Ela permite que uma solução ou serviço se comunique com outros produtos e serviços sem precisar saber como eles foram implementados. Simplifica o desenvolvimento de aplicações gerando economia de tempo e dinheiro e oferecem a flexibilidade necessária para simplificar o design, a administração e o uso, além de fornecer oportunidades de inovação ao desenvolver novas ferramentas e soluções (ou ao gerenciar aquelas já existentes). São como contratos com documentações que representam um acordo entre as partes interessadas. Se uma das partes envia uma solicitação remota estruturada de um forma específica, a outra responderá especificamente da mesma forma . Além disso, permite simplificar a forma como os

desenvolvedores integram novos componentes de aplicações a uma arquitetura preexistente. Muitas vezes, as necessidades empresariais mudam rapidamente para responder aos mercados digitais que estão sempre em transformação. Nesse ambiente, novos concorrentes podem redefinir o setor inteiro com uma nova aplicação. Portanto, é importante oferecer suporte à implantação e desenvolvimento rápidos de serviços inovadores. Por este motivo, as APIs ajudam na colaboração entre as equipes de TI e as empresas (RED HAT, 2020b).

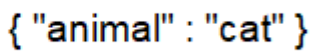
O desenvolvimento de aplicações nativas em cloud (em nuvem) é uma forma conhecida de aumentar a velocidade de criação. Ele depende de uma arquitetura de aplicações de microserviços conectada por meio de uma API. Para tanto podem ser utilizados API Web para captura de dados para ser utilizados pelo aplicativo, isto é, fazer uso das funcionalidades da API sem conhecer as funcionalidades desta aplicação (RED HAT, 2020b).

2.4 JSON

JSON significa JavaScript Object Notation (Notação de Objetos JavaScript) é uma estrutura de dados que representa um objeto em JavaScript (LECHETA, 2015).

Segundo BASSETT (2015), é independente da linguagem de programação. Utiliza o conceito de pares nome-valor muito difundido na computação. Esses pares são conhecidos também por outros nomes: pares chave-valor, pares atributo-valor e pares campo-valor (vide Figura 1). Sendo assim, deve-se usar chaves em torno do par nome-valor para transformá-lo em um objeto (BASSETT, 2015).

Figura 1 - Par chave-valor



Fonte: autor.

Em JSON, vários pares nome-valor são separados por vírgula. Ao esquecer-se de finalizar um objeto com uma chave à direita seu objeto não será reconhecido como um objeto.

O formato JSON pode constituir um documento próprio e residir em um sistema de arquivos e sua extensão é *.json* (BASSETT, 2015).

Conforme BASSETT (2015), os tipos primitivos que a maioria das linguagens de programação compartilham são:

- números inteiros, ponto flutuante e ponto fixo;
- caracteres ("a","A") ou strings ("apple");
- boolenos (verdadeiro ou falso).

O Objeto é um tipo de dado cuja a estrutura composta de dados pode ser decomposta em tipos nativos que o permite ser entendido por linguagens de programação em que não há o objeto como tipo. Logo, a comunicação entre dois sistemas diferentes pode ocorrer desde que tenham o mesmo formato de dados e que o território comum seja expresso nesse formato (BASSETT, 2015).

Segundo BASSETT, o tipo de dados JSON são:

- objeto;
- string;
- número;
- booleano;
- null;
- array.

Em JSON, misturar e combinar tipos de dados é válido, *mas não é uma prática recomendável*. O array pode ser constituído de qualquer um dos tipos de dados possíveis. Sendo assim, pode-se ter um array de strings, um array de números, um array de booleanos, um array de objetos ou um array de arrays. Um *array de arrays* é chamado de array multidimensional (BASSETT, 2015).

Nos formatos com cenários para intercâmbio de dados, os dados são criados para serem enviados de uma rede para outra parte. Esta parte espera receber um formato desejado para o documento que ela estiver esperando, com determinada estrutura e certos tipos de dados. Geralmente, ela fornece documentação que explica o formato esperado e disponibiliza exemplos. Mesmo quando a mais detalhada e bela das documentações é fornecida, não é difícil gerar erros em dados (BASSETT, 2015).

Para que os seus dados não apresentem erros de conformidade, no sentido de que "é enviado algo e é recebido outra coisa", os dados e formato JSON devem ser validados. Por isso surgiu o JSON Schema. Ele é utilizado na extremidade da transação que recebe os dados. Trata-se da primeira linha de defesa na aceitação dos dados para verificar se os dados estão adequados (BASSETT, 2015).

Segundo BASSETT (2015), ele visa responder as seguintes perguntas:

- os tipos de dados dos valores estão corretos?
- essas informações incluem os dados obrigatórios?
- os valores estão nos formatos exigidos?

Um JSON Schema é escrito em JSON, portanto ler ou escrever um é somente um processo diferente. No primeiro par nome-valor de um Schema, deve-se declarar como um Schema document. O nome dessa declaração sempre será *"$schema"* (vide figura 2).

Figura 2 - Exemplo de uma declaração $schema

```
{

    "$schema" : "http://json-schema.org/draft-04/schema#"

}
```

Fonte: BASSETT, 2015.

O segundo par nome-valor será o "*título"*. O terceiro par nome-valor define-se as "*propriedades que se desejam ser incluídas"* no JSON. O quarto par nome-valor tem o nome de "*required"* com um array contendo os valores obrigatórios. Vide figura 3.

Figura 3 - Exemplo de JSON Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Cat",
    "properties" : {
        "name": {
            "type" : "string"
        },
        "age": {
            "type": "number",
            "description": "Your cat's age in years."
        },
        "declawed": {
            "type": "boolean"
        },
        "description": {
            "type": "string"
        }
    },
    "required": [
        "name",
        "age",
        "declawed"
    ]
}
```

Fonte: BASSETT, 2015.

Considerando o JSON Schema da figura 3 acima, tem-se o seguinte JSON válido na figura 4, abaixo:

Figura 4 - Exemplo de JSON válido.

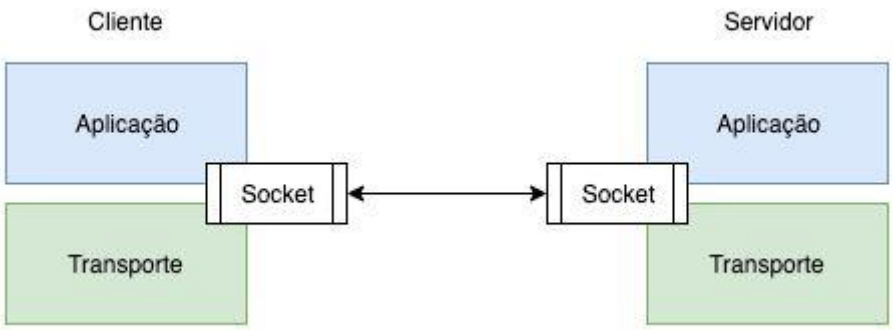
```
{
    "name": "Fluffy",
    "age": 2,
    "declawed": false
}
```

Fonte: BASSETT, 2015.

2.5 SOCKETS

O Java API para WebSocket é que fornece suporte para criação de aplicações WebSocket. Vide figura 5.

Figura 5 - Esquema sockets



Fonte: TREINAWEB, 2020.

Segundo ORACLE (2014), webSocket é um protocolo de aplicação que fornece comunicação full-duplex (recebe e envia dados) entre dois pontos através do protocolo Transmission Control Protocol (TCP).

No modelo tradicional de requisição-resposta usado no Hypertext Transfer Protocol (HTTP), o cliente realiza requisições e o servidor responde a elas. A troca é sempre inicializada pelo cliente; o servidor não pode enviar qualquer dado sem que o cliente requisite, primeiramente. Este modelo trabalhou bem para World Wide Web (internet) quando cliente realizava requisições por documentos que sofriam mudanças ocasionais, mas as limitações foram se tornando relevantes a partir do momento que os documentos mudavam rapidamente e os usuários desejavam maior interatividade com a Web. O protocolo WebSocket lida com essas limitações fornecendo um canal de comunicação entre o cliente e o servidor. Combinado com outras tecnologias, tais como JavaScript e HTML5, WebSocket permite às aplicações entregar uma experiência mais rica ao usuário (ORACLE, 2014).

Em uma aplicação WebSocket, o servidor publica um ponto WebSocket, e o cliente usa o URI de nó para conectar-se ao servidor. O Protocolo WebSocket apresenta as mesmas funções para ambos os nós, uma vez que a comunicação esteja estabelecida. O cliente e servidor podem enviar mensagens de um para o outro a qualquer hora, enquanto a conexão estiver aberta. O encerramento pode ser executado a qualquer hora. Clientes geralmente conectam-se somente a um servidor e servidores aceitam conexões de vários clientes ao mesmo tempo (ORACLE, 2014).

2.6 FIREBASE

O Firebase é uma plataforma criada sob a infraestrutura da Google em nuvem com produtos que podem trabalhar compartilhando dados e insights entre si, em conjunto (FIREBASE, 2020a).

Dentre os vários produtos, pode-se citar o Cloud Firestore (armazena e sincroniza dados do aplicativo em escala global) e Realtime Database (armazena e sincroniza dados de aplicativos em milisegundos) (FIREBASE, 2020a).

O Realtime Database armazena e sincroniza os dados com um banco de dados NoSQL na nuvem. Eles são sincronizados em todos os clientes em tempo real e permanecem disponíveis, quando o aplicativo está offline, utilizando o cache do dispositivo. Vide figura 6.

Figura 6 - Atualização de dados simultânea para todos os usuários

Fonte: FIREBASE, 2020a.

Isto quer dizer, que a partir do momento em que o usuário torna-se online, todos os dados são sincronizados mesclando qualquer conflito automaticamente (FIREBASE, 2020a).

O Firebase Realtime Database pode ser acessado diretamente de um dispositivo móvel ou navegador da Web, sem um servidor de aplicativos. A segurança e a validação de dados estão disponíveis por meio de regras de segurança, baseadas em expressões do Firebase Realtime Database, executadas quando os dados são lidos ou gravados. Permite o controle, o acesso às informações em cada banco de dados com regras personalizadas para cada instância de banco de dados. Os dados são armazenados como JSON (FIREBASE, 2020a).

FIREBASE (2020e) apresenta os principais conceitos da arquitetura de dados e práticas recomendadas para estruturar dados JSON no Firebase Realtime Database. Cita que para criar corretamente um banco de dados estruturado exige planejamento e afirma que o mais importante é planejar como os dados serão salvos e depois recuperados da forma mais fácil possível.

Todos os dados do Firebase Realtime Database são armazenados como objetos JSON na nuvem. Ao contrário de um banco de dados SQL, não há tabelas nem registros. Ao adicionar dados à árvore JSON, eles se tornam um nó na estrutura JSON com uma chave associada. É possível fornecer suas próprias

chaves, como códigos de usuário e nomes semânticos ou até gerá-las utilizando o método push() (FIREBASE, 2020e).

FIREBASE (2020e) observa que, caso se queira criar as próprias chaves, devem ser codificadas em UTF-8. Elas podem ter até 768 bytes e não podem conter os caracteres ., $, #, [,], / nem os caracteres de controle ASCII de 0 a 31 ou 127. Não é possível usar caracteres de controle ASCII nos próprios valores.

FIREBASE (2020e) recomenda as seguintes práticas para a estruturação de dados NoSQL (Realtime Database):

- evitar o aninhamento de dados com grandes níveis de profundidade;
- simplificar a estrutura de dados;
- criar dados escalonáveis.

Segundo FIREBASE (2020e), o Realtime Database apresenta 32 níveis de profundidade, mas informa que deve-se procurar tornar a estrutura mais simples possível, uma vez que, quando se busca os dados do nó, se recupera todos os dados dos nós-filhos, também. Pode-se observar, ainda, que quando o usuário tem acesso a leitura e gravação de um nó do banco de dados, ele passa a acessar todos os dados desse nó. O exemplo da figura 7, mostra que para listar os títulos de conversas de bate-papo, é necessário fazer o download de toda a árvore chats para o cliente, inclusive os membros e as mensagens (FIREBASE, 2020e).

Figura 7 - Aninhamento de dados que deve ser evitado

```
{
  // This is a poorly nested data architecture, because iterating the children
  // of the "chats" node to get a list of conversation titles requires
  // potentially downloading hundreds of megabytes of messages
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message": "Relay malfunction found. Cause: moth." },
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}
```

Fonte: FIREBASE, 2020f.

Menciona FIREBASE (2020e), a exemplo da figura 8, que deve-se adotar a simplificação da estrutura de dados dividindo os dados em caminhos ou ramos separados, como um processo de desnormalização, permitindo que o download possa ser feito em chamadas separadas, conforme a necessidade.

Figura 8 - Simplificação de estrutura de dados.

```
{
  // Chats contains only meta info about each conversation
  // stored under the chats's unique ID
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth.",
      "timestamp": 1459361875666
    },
    "two": { ... },
    "three": { ... }
  },

  // Conversation members are easily accessible
  // and stored by chat conversation ID
  "members": {
    // we'll talk about indices like this below
    "one": {
      "ghopper": true,
      "alovelace": true,
      "eclarke": true
    },
    "two": { ... },
    "three": { ... }
  },

  // Messages are separate from data we may want to iterate quickly
  // but still easily paginated and queried, and organized by chat
  // conversation ID
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems to be malfunctioning.",
        "timestamp": 1459361875337
      },
      "m2": { ... },
      "m3": { ... }
    },
    "two": { ... },
    "three": { ... }
  }
}
```

Fonte: FIREBASE, 2020f.

FIREBASE (2020e) recomenda que ao criar um aplicativo, faça o download somente de subconjuntos de lista, uma vez que, a lista pode conter milhares de registros. Quando a relação é estática e unidirecional, basta agrupar os objetos filhos subordinados ao nó pai. Às vezes, essa relação é mais dinâmica e convém desnormalizar esses dados usando uma consulta para recuperar um subconjunto de dados. Entretanto, até isso pode ser insuficiente em uma relação bidirecional entre usuários e grupos. Os usuários podem pertencer a um grupo, e os grupos incluem uma lista de usuários. A dificuldade aparece na hora de decidir a quais grupos um usuário pertence (FIREBASE, 2020e).

FIREBASE (2020e) cita que seria necessário encontrar um meio eficiente de listar os grupos a que um usuário pertence e buscar dados somente desses grupos. Um *índice* dos grupos pode ser muito útil neste caso. Vide figura 9.

Figura 9 - Exemplo de uma criação de um índice de Grupos.

```
// An index to track Ada's memberships
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      // Index Ada's groups in her profile
      "groups": {
         // the value here doesn't matter, just that the key exists
         "techpioneers": true,
         "womentechmakers": true
      }
    },
    ...
  },
  "groups": {
    "techpioneers": {
      "name": "Historical Tech Pioneers",
      "members": {
        "alovelace": true,
        "ghopper": true,
        "eclarke": true
      }
    },
    ...
  }
}
```

Fonte: FIREBASE, 2020f.

Percebe-se que alguns dados são duplicados quando a relação é armazenada tanto no registro de Ada como no grupo. Agora, alovelace é indexado em um grupo e techpioneers é listado no perfil de Ada. Então, *para excluir Ada do grupo, é necessário que isso seja atualizado nos dois locais*. Essa é uma **redundância necessária nas relações bidirecionais**. Ela permite que as associações de Ada sejam carregadas de maneira rápida e eficiente, mesmo quando a lista de usuários ou grupos atinjam milhões ou quando as regras de segurança do Realtime Database impedem o acesso a alguns dos registros (FIREBASE, 2020e).

Essa abordagem, ao inverter os dados listando os códigos como chaves e definir o valor como true, torna a verificação de uma chave, uma tarefa tão simples quanto ler /users/$uid/groups/$group_id e verificar se é null. A indexação é mais rápida e muito mais eficiente do que o envio de consultas ou a varredura dos dados (FIREBASE, 2020e).

2.7 ANDROID STUDIO

O Android Studio (ANDROID, 2020a) é o ambiente de desenvolvimento integrado oficial para o desenvolvimento de Aplicativo (app) para Android e é baseado na ferramenta IntelliJ IDEA da JetBrains (desenvolvedora de ferramentas para desenvolvedores). Além do editor de código e das ferramentas de desenvolvedor avançadas do IntelliJ, o Android Studio oferece ainda mais recursos para aumentar sua produtividade na compilação de Aplicativo Android, como:

- um sistema de compilação flexível baseado em Gradle;
- um emulador rápido com inúmeros recursos;
- um ambiente unificado que possibilita o desenvolvimento para todos os dispositivos Android;
- a aplicação de alterações para enviar alterações de código e recursos ao aplicativo em execução sem reiniciar o aplicativo;
- modelos de código e integração com GitHub para ajudar a criar recursos comuns de apps e importar exemplos de código;
- frameworks e ferramentas de teste cheios de possibilidades;
- ferramentas de lint para detectar problemas de desempenho, usabilidade, compatibilidade com versões, entre outros;
- compatibilidade com C++ e NDK;
- compatibilidade integrada com o Google Cloud Platform, facilitando a integração do Google Cloud Messaging e do App Engine.

O Android Studio não usa espaços de trabalho. Portanto, projetos separados são abertos em janelas distintas dele. Ele organiza código em projetos, que contêm tudo o que define o app Android, desde o código-Fonte até configurações da compilação e código de teste. Cada projeto contém um ou mais módulos, que permitem dividir o projeto em unidades distintas de funcionalidade. Os módulos podem ser compilados, testados e depurados independentemente (ANDROID, 2020b).

O Sistema de compilação do Android Studio (ANDROID, 2020d) é baseado no Gradle possibilitando que os projetos tenham disponíveis os seguintes recursos:

- compatibilidade com bibliotecas binárias (Android Archive - AAR), o que evita a necessidade de copiar bibliotecas nos próprios projetos, bastando declarar uma dependência e a biblioteca é automaticamente transferida por download e combinada ao projeto;
- compatibilidade com variantes de compilação, o que permite compilar versões diferentes do app para o mesmo projeto;
- facilidade de configuração e personalização de compilações;
- poder ser utilizado o Gradle a partir do IDE, por linha de comando e por meio de servidores de integração contínua, como o Jenkins, o que gera a mesma compilação a qualquer momento e de qualquer lugar.

2.8 ANDROID SOFTWARE DEVELOPMENT KIT (SDK)

A linguagem Java® surgiu em 1995 pela Sun Microsystems, Inc.® (SCHILDT, 2015) e foi adquirida pela Oracle ® em 27 de janeiro de 2010 (WIKIPEDIA, 2010). É a linguagem mais utilizada para desenvolver aplicativos Android, portanto será essa que utilizar-se-á (GRIFFITHS *et al.*, 2019).

Os dispositivos Android não executam os arquivos .class e *.jar*, o que inviabiliza o uso de um ambiente de desenvolvimento Java normal (GRIFFITHS *et al.*, 2019).

Para que fosse possível aumentar a velocidade e o desempenho da bateria, os dispositivos utilizam formatos compilado e otimizados de código. Para isso foi necessário a criação do Software Development Kit (SDK) Android que contém as bibliotecas e ferramentas necessárias para o desenvolvimento de aplicativos Android (GRIFFITHS *et al*., 2019).

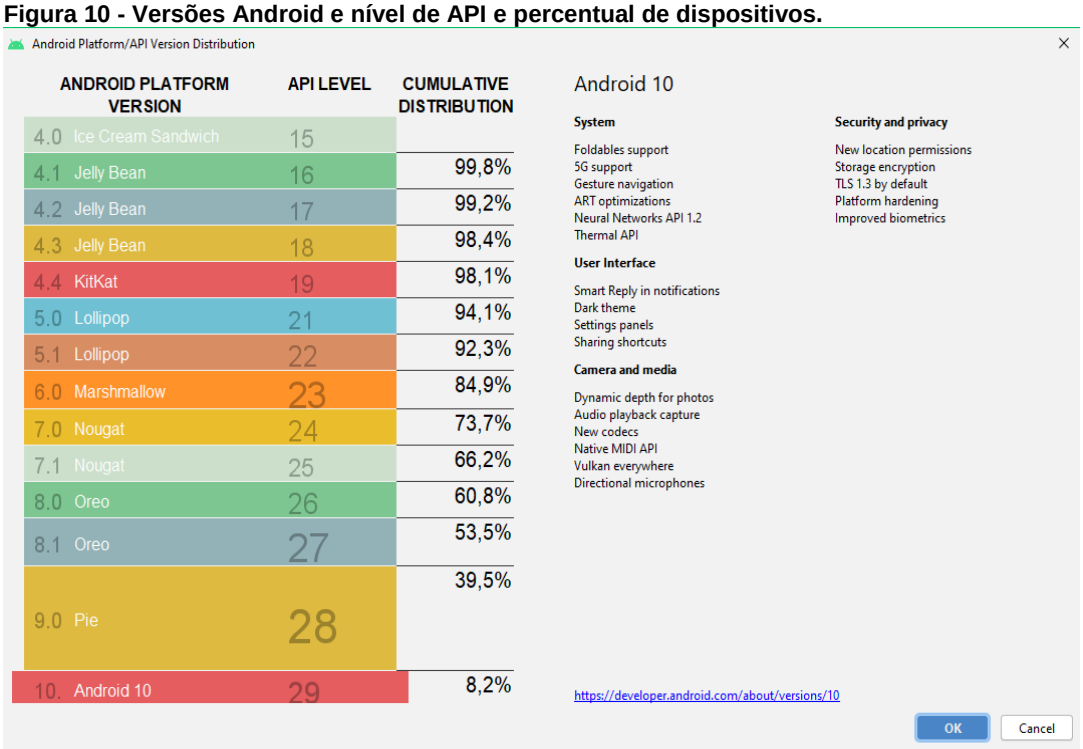
O SDK Android:

- possui uma plataforma diferente para cada versão;
- possui ferramentas para depuração e teste e outros utilitários práticos dependentes da plataforma;
- possui amostras de aplicativos práticos de código e usos de algumas API;
- possui acesso offline à documentação da API mais recente;
- possui suporte para Android cujas APIs extras não estão disponíveis na plataforma padrão;
- permite integrar serviços de faturamento no aplicativo.

Ao desenvolver aplicativos Android é preciso definir a versão do SDK mínimo com as quais versões do Android, os aplicativos serão compatíveis. Se determinar a compatibilidade do aplicativo com a versão mais recente do SDK, talvez ele não possa ser executado em muitos aplicativos (GRIFFITHS *et al.*, 2019).

O nível de API é um valor inteiro que identifica unicamente a revisão da API da biblioteca oferecida por uma versão da plataforma Android (ANDROID, 2020c). Vide esquema na figura 10.

Figura 10 - Versões Android e nível de API e percentual de dispositivos.



Fonte: autor.

A plataforma Android oferece uma API da biblioteca que pode ser usada pelos aplicativos para interagir com o sistema Android (ANDROID, 2020c).

Segundo ANDROID (2020c), a API da biblioteca consiste em:

- um conjunto de pacotes e classes principais;
- um conjunto de elementos e atributos Extensible Markup Language (XML) para declarar um arquivo manifesto;
- um conjunto de elementos e atributos Extensible Markup Language (XML) para declarar e acessar recursos;
- um conjunto de intents;

- um conjunto de permissões que podem ser solicitadas pelos aplicativos, bem como, a aplicação das permissões incluídas no sistema.

Cada versão da plataforma Android pode conter atualizações para a API da biblioteca de aplicativos Android. A API da biblioteca fornecida por uma plataforma Android é especificada por um identificador inteiro denominado “nível de API” e que é única para cada versão. ANDROID (2020e) menciona que o identificador de nível de API desempenha um papel essencial para garantir a melhor experiência possível para os usuários e desenvolvedores de aplicativos de forma que permita que:

- a plataforma Android correlacione com a revisão máxima compatível de biblioteca de API;
- os aplicativos informem a revisão de biblioteca de API exigida;
- o sistema impeça a instalação de aplicativos com versões incompatíveis no dispositivo do usuário.

Segundo ANDROID (2020e), os aplicativos podem usar um elemento de manifesto fornecido pela API da biblioteca (<uses-sdk>) para descrever os níveis mínimo e máximo de API em que podem ser executados, bem como o nível de API preferencial para que foram projetados. O elemento oferece três atributos principais:

- **android:minSdkVersion** — Especifica o nível de API mínimo em que o aplicativo pode ser executado. O valor padrão é “1”.
- **android:targetSdkVersion** — Especifica o nível da API em que o aplicativo foi projetado para executar. Em alguns casos, isso permite que o aplicativo use elementos do manifesto ou comportamentos definidos no nível da API direcionada, em vez de estar restrito ao uso exclusivo daqueles definidos para o nível mínimo da API.
- **android:maxSdkVersion** — Especifica o nível de API máximo em que o aplicativo pode ser executado.

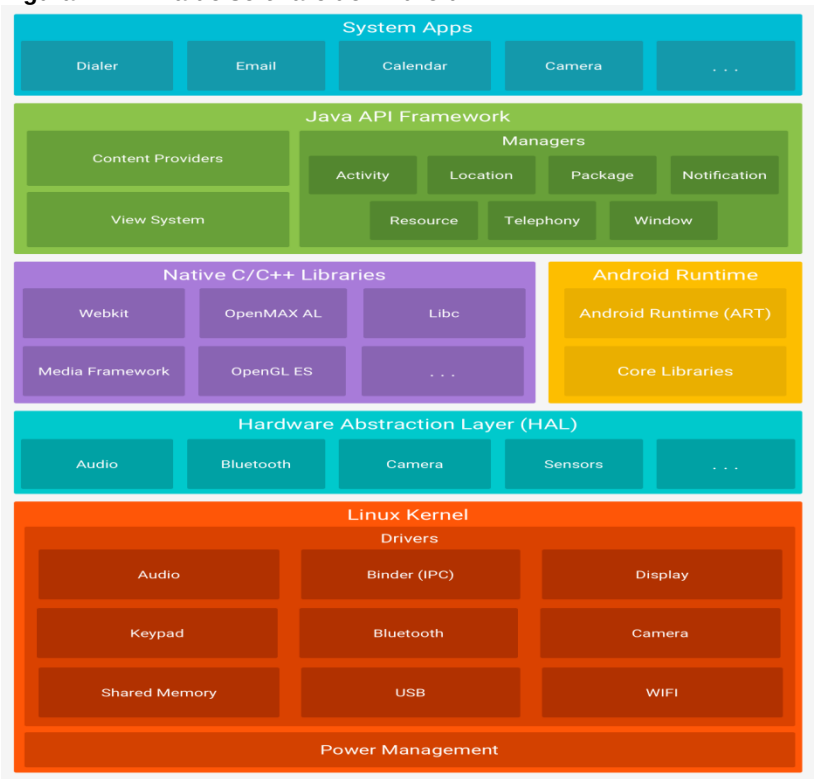
2.9 ANDROID

O Android é uma plataforma móvel mais popular do momento. Atualmente, há mais de dois bilhões de dispositivos Android ativos no mundo, e

esse número vem aumentando rapidamente. Foi desenvolvido pelo Google® e é uma plataforma de código-aberto baseado no Linux. É um poderoso framework de desenvolvimento e contém os elementos essenciais para a criação de excelentes aplicativos com uma combinação de Java e XML (GRIFFITHS *et al*., 2019).

É uma pilha de software com base em Linux de código aberto criada para diversos dispositivos e fatores de forma. A figura 11, mostra a maioria dos componentes da plataforma Android (ANDROID, 2020f).

Figura 11 - Pilha de software do Android.



Fonte: ANDROID, 2020f.

Ao construir aplicativos, tem-se acesso às API usadas pelos aplicativos básicos como contatos, telefone, calendário e um navegador. Estas APIs servem para controlar a aparência e o comportamento do aplicativo.

Abaixo da camada JAVA Application Framework há um conjunto de bibliotecas C e C++ que são acessadas através daquelas API. Já o Android Runtime vem com algumas bibliotecas básicas que implementam a maior parte da linguagem de programação Java onde cada aplicativo é executado em seu próprio processo.

Abaixo de todos os componentes está o kernel Linux. O Android utiliza o kernel para drivers e serviços básicos, como segurança e gerenciamento de memória (GRIFFITHS *et al*., 2019).

Uma atividade é uma classe especial do Java que determina o layout a ser usado e orienta o aplicativo sobre como responder ao usuário. Por exemplo, se um layout contém um botão, é necessário escrever código em Java na atividade para definir o que o botão deve fazer quando pressionado.

Além das atividades e layouts, os aplicativos Android precisam de recursos adicionais, como arquivos de imagem e dados de aplicativo, geralmente. Pode-se inserir arquivos adicionais no aplicativo, sempre. Toda atividade tem um ciclo de vida cujos os eventos acontecem de forma cíclica. Além dos métodos onCreate() e onDestroy() pertinentes ao ciclo de vida global da atividade, existem outros métodos de ciclo de vida que atuam sobre a visibilidade da atividade. Pode-se citar os cinco métodos mais importantes do ciclo de vida de uma atividade (GRIFFITHS *et al*., 2019):

- onCreate();
- onStart();
- onStop();
- onRestart();
- onStart();
- onStop().

No método onCreate(), o código de inicialização da atividade é executada. Nesse ponto, a atividade ainda não está visível, pois onStart() ainda não foi chamado. No método onStart(), após sua execução, o usuário visualiza a atividade na tela. Portanto, nesse instante é que foi inserido o código de consulta ao bando de dados Firebase, a captura dos dados e armazenamento utilizando a classe FiscalModel.class do protótipo desenvolvido. No método onStop(), após a sua execução a atividade deixa de estar visível. Os métodos onRestart e onStart() serão chamados sucessivamente, se a atividade tornar visível para o usuário novamente. A atividade pode passar por esse ciclo muitas vezes se ficar invisível e visível reiteradamente. No método onDestroy(), a atividade é destruída, mas onStop() será chamado antes (GRIFFITHS *et al*., 2019).

Na realidade, os aplicativos Android são apenas vários arquivos em diretórios específicos. Quando se constrói um aplicativo, todos esses arquivos são reunidos para formar o aplicativo que será executado em um dispositivo (GRIFFITHS *et al*., 2019).

3 METODOLOGIA

O protótipo de aplicativo a ser desenvolvido consiste em que tela que relacione os usuários previamente cadastrados. De forma que esses possam buscar as suas ordens de serviço e realizar a análise de conformidade dos itens de serviços (quantidades executadas), liberando ou não para pagamento. Tudo isso de forma remota, com acesso à nuvem por meio de um dispositivo móvel.

No Apêndice A, demonstra-se as etapas de execução do protótipo de aplicativo e no Apêndice B, tem-se a codificação propriamente dita.

3.1 REQUISITOS

Os requisitos do protótipo são:

- apresentar uma relação de fiscais de serviço;
- permitir que o fiscal de serviço selecione o item correspondente ao fiscal desejado;
- ao clicar no fiscal de serviço desejado será apresentado a relação de Ordem de Serviço (OS) que pertence ao mesmo;
- ao clicar na OS selecionada será disponibilizado uma lista de itens de serviços correspondente à OS;
- disponibilizado os itens da OS, o usuário poderá alterar o quantitativo do itens de serviço desejado;
- deve-se-á clicar no ícone que comanda a gravação do quantitativo alterado no cache do dispositivo;
- terminado as alterações desejadas, deverá clicar no botão finalizar que comandará a gravação em nuvem e retornará a tela de OS possibilitando a seleção de nova OS.

3.2 PREMISSAS

O protótipo de aplicativo assume as seguintes premissas:

- ser desenvolvido em linguagem Java;
- ser projetado para funcionar na plataforma Android;
- consumir dados do Firebase RealtimeDatabase;
- ser desnecessário login ou informações do usuário.

3.3 PROTÓTIPO

O protótipo de aplicativo deverá ser desenvolvido para ambiente em nuvem utilizando o Firebase Realtime Database. Para que se torne possível o acesso dever-se-á criar um projeto dentro do Firebase e realizada as configurações de forma a definir o plano desejado. O plano Spark é gratuito até o limite de 1GB de armazenamento. As regras de uso permitem estabelecer o processo de leitura e escrita do database, quem pode acessar, bem como, criar as chaves de índices para aprimoramento de pesquisa, segurança e desempenho.

Opcionalmente, pode ser informado a chave hash *sha1* do protótipo de aplicativo dentro do projeto Firebase de forma a vincular o aplicativo ao projeto.

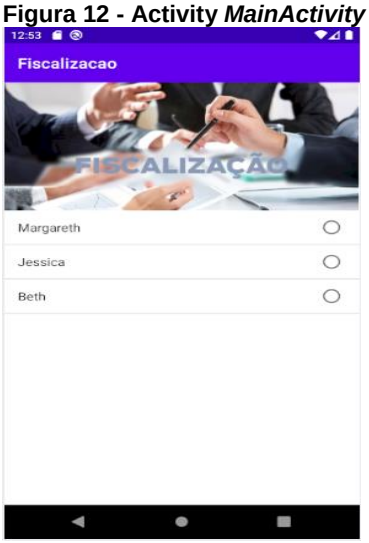
Deve-se realizar o download do arquivo *google-services.json* gerado na criação do projeto e anexado ao diretório *app* do aplicativo o que permitirá o seu acesso ao Firebase RealtimeDatabase.

O Android Studio deve ser inicialmente configurado com a inserção de dependências no build-gradle do projeto e do app e que permitirão o mesmo conectar à internet, ao Firebase e fazer uso das bibliotecas necessárias para acesso, leitura e gravação de dados na nuvem.

Três activities serão criadas onde serão disponibilizadas as informações em forma de lista. A primeira e a principal, a *MainActivity* corresponde a relação de fiscais de serviço; a segunda, a activity *OrdensFiscal*, corresponde a lista de OS-Ordens de Serviços do fiscal selecionado; a terceira, a activity *ItensOs* corresponde a lista de Itens de Serviços da OS selecionada.

A activity *MainActivity* é composta de uma *ImageView e ListView*. Vide figura 12. A ImageView conterá uma imagem simbólica da fiscalização e o listView relacionará os fiscais de serviço.

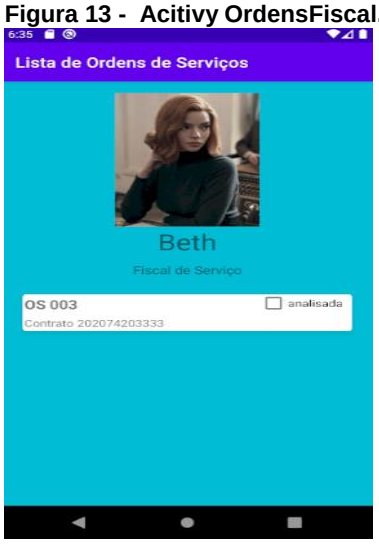
Figura 12 - Activity *MainActivity*



Fonte: autor

A activity *OrdensFiscal* é composta de uma *ImageView, um TextView e um RecyclerView*. Vide figura 13. A ImageView conterá de forma estática e única para todos os fiscais a imagem *anyataylorjoy.png.* O recycler view conterá dentro de um cardview, as informações da OS, do contrato e situação da OS (analisada ou não).

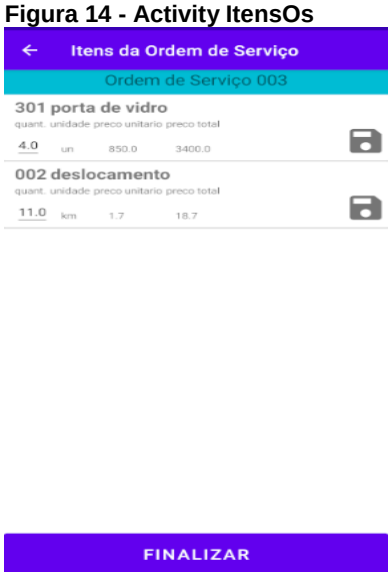
Figura 13 - Acitivy OrdensFiscal.



Fonte: autor.

A activity *ItensOs* é composta de *TextView, um RecyclerView e um Button*. Vide figura 14.

Figura 14 - Activity ItensOs



Fonte: autor.

O textview conterá o número da OS selecionada. O recycler view conterá dentro de um cardview, os dados do item de OS. Esse cardview terá 3 linhas que conterão as seguintes informações. A primeira linha terá: o código do item; a descrição do item; a segunda linha: textos-títulos quant., unidade, preço unitário e preço total; a terceira linha: quantidade (editável), a unidade, preço unitário, preço total e, finalmente, o ícone de salvar (salvará a alteração em cache). Abaixo do recycler view, terá o botão finalizar que gravará todas as alterações gravadas em cache na nuvem.

Na parte superior das activities há uma *actionbar* que retorna para activity anterior.

4 DESENVOLVIMENTO

4.1 REQUISITOS DE SISTEMA

O projeto visa desenvolver um protótipo de aplicativo móvel para fiscalização de obras de forma remota com recursos para gravar/alterar quantitativos de serviços realizados por fornecedores de serviços contratados.

4.1.1 Convenções, termos e abreviações

A correta interpretação deste documento exige o conhecimento de algumas convenções e termos específicos, que são descritos a seguir.

4.1.2 Identificação dos requisitos

Por convenção, a referência a requisitos é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do requisito, de acordo com a especificação a seguir:

requisito funcional
[nome da subseção. identificador do requisito]
[RF:01]
[Acesso ao sistema :RF01]
[Capturar foto :RF02]
[Gravar videos :RF03]

requisito não-funcional
[Restrição de acesso:NF01]

Por exemplo, o requisito funcional [Recuperação de dados.RF016] deve estar descrito em uma subseção chamada “Recuperação de dados”, em um bloco identificado pelo número [RF016].

Já o requisito não-funcional [Confiabilidade.NF008] deve estar descrito na seção de requisitos não-funcionais de Confiabilidade, em um bloco identificado por [NF008].

Os requisitos devem ser identificados com um identificador único. A numeração inicia com o identificador [RF001] ou [NF001] e prossegue sendo incrementada à medida que forem surgindo novos requisitos.

4.1.3 Prioridades dos requisitos

Para estabelecer a prioridade dos requisitos, adotou-se as seguintes denominações “essencial”, “importante” e “desejável”.

- **essencial** é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.
- **importante** é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.
- **desejável** é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

4.2 ABRANGÊNCIA E SISTEMAS RELACIONADOS

A abrangência envolverá rede em nuvem utilizando o Firebase Realtime Database.

4.3 REQUISITOS FUNCIONAIS

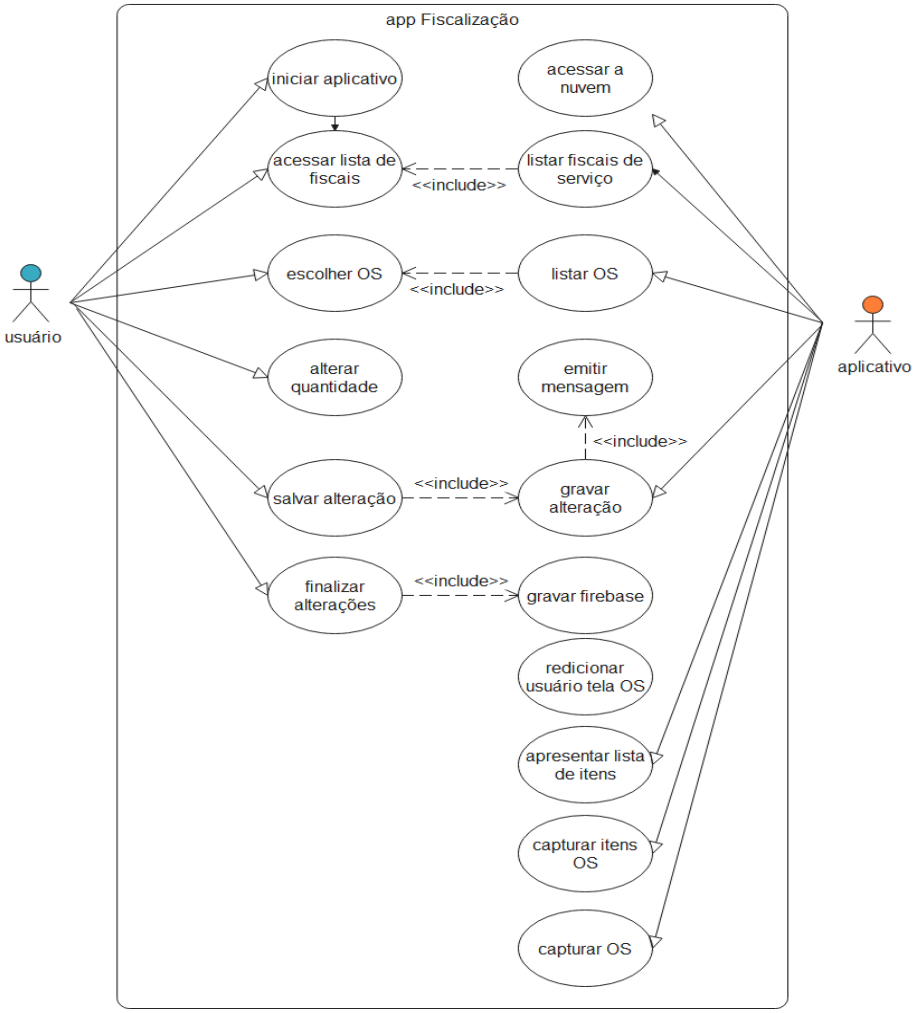
Este deverá:

- acessar via aplicativo móvel lista de fiscais de serviços (essencial);
- listar as Ordens de Serviço (OS) pertencentes ao fiscal de serviço selecionado (essencial)
- acessar os itens da OS selecionada (essencial)
- alterar quantitativo de serviço realizado (essencial)
- gravar quantitativo de serviço alterado em cache (essencial)
- gravar quantitativos de serviços alterados na nuvem(essencial)

4.3.1 Diagramas de Casos de Uso

Na figura 15, tem-se o diagrama de caso de usos do sistema.

Figura 15 - Diagrama de casos de usos do sistema.



Fonte: autor.

4.3.1.1 Acesso

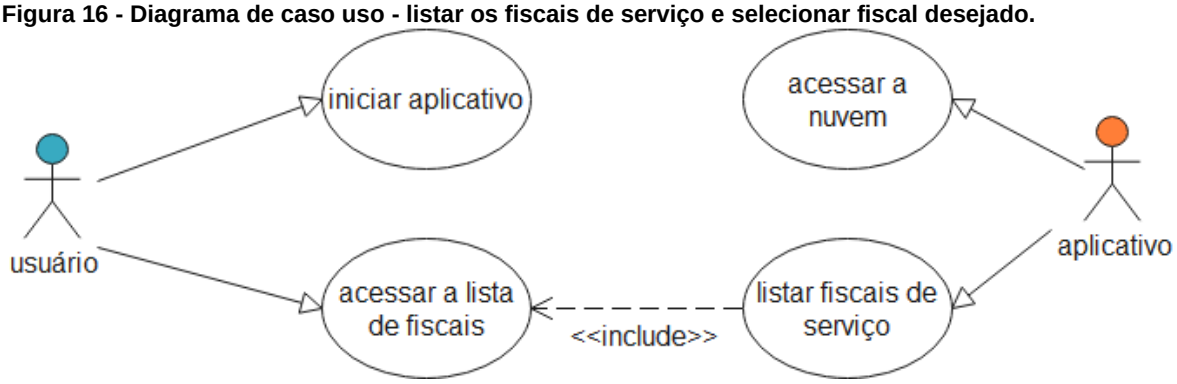
[RF01] Listar os fiscais de serviço

O usuário deverá acessar a rede remota. Este passo é importante, pois garante que cada usuário da aplicação terá acesso à relação de fiscais.

Descrição do caso de uso:

Vide o diagrama de caso de uso na figura 16.

Figura 16 - Diagrama de caso uso - listar os fiscais de serviço e selecionar fiscal desejado.



Fonte: autor.

A Tabela 1 apresenta o caso de uso - listar os fiscais de serviços e selecionar o fiscal desejado.

Tabela 1 - UC01 – Listar os fiscais de serviço e selecionar fiscal desejado

Nome	UC01 – Listar os fiscais de serviço e selecionar fiscal desejado
Descrição	o aplicativo lista os fiscais de serviço e o usuário seleciona o fiscal de serviço.
Eventos	• inicialização do aplicativo; • conexão com a nuvem; • captura dos dados dos fiscais de serviço pelo aplicativo; • visualização da lista de fiscais de serviço; • escolha do fiscal de serviço.
Atores	aplicativo, usuário.
Pré-Condições	• acesso a rede de comunicação em nuvem; • usuário inicia o aplicativo.
Pós-Condições	1. Conclusões com sucesso: • usuário acessa lista de fiscais de serviço e seleciona o fiscal desejado. 2. Conclusões sem sucesso: • o usuário não acessa a lista de fiscais de serviços(A3).
Sequência	1. iniciar o aplicativo. 1.1. o usuário inicia o aplicativo (A1).

principal	2. acessar a rede em nuvem. 2.1. aplicativo acessa a nuvem (A2). 3. listar os fiscais de serviço. 3.1. o aplicativo lista os fiscais de serviços (A3). 4. escolher o fiscal de serviço. 4.1. usuário seleciona o fiscal desejado (A4). 5. fim
Sequência alternativa	A1. em 1.1, caso o usuário não conseguir iniciar o aplicativo. A1.1.informar a mensagem de erro. A2. em 2.1, caso o aplicativo não conseguir acessar a rede: A2.1 utilizar dados disponíveis em cache. A3. em 3.1, caso o aplicativo não liste a relação de fiscais de serviço: A3.1 utilizar dados disponíveis em cache. A4. em 4.1, caso o usuário selecione o fiscal: A4.1. direcionar para tela de OS do fiscal selecionado.

Fonte: autor.

Prioridade: Essencial.

Entradas e pré-condições:

- o usuário deve ter realizado a instalação do aplicativo no celular;
- o usuário deve estar cadastrado.

Saídas e pós-condições:

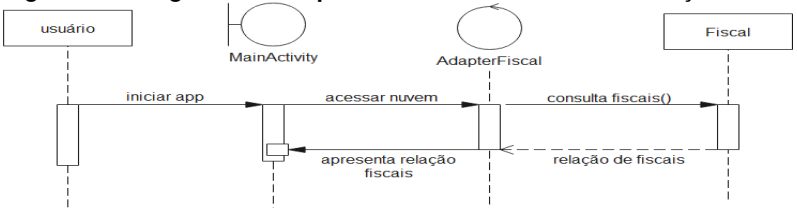
A. sucesso:

- O usuário acessa lista de fiscais de serviços e seleciona o fiscal desejado.

B. sem sucesso: usuário não acessa a lista de fiscais de serviços.

Vide o respectivo diagrama de sequencia na figura 17.

Figura 17 - Diagrama de sequencia - lista de fiscais de serviço.



Fonte: autor.

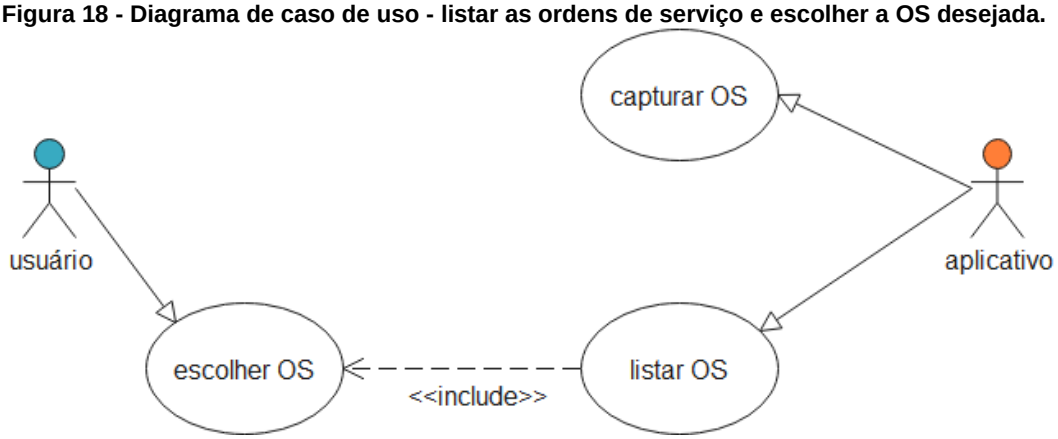
4.3.1.2 Seleção

[RF02] Listar as Ordens de Serviço (OS)

Para listar as OS pertencentes ao fiscal de serviço, as OS devem estar cadastradas na nuvem em nome do fiscal.

Descrição do caso de uso:

Figura 18 - Diagrama de caso de uso - listar as ordens de serviço e escolher a OS desejada.



Fonte: autor.

Vide o diagrama de caso de uso na figura 18.

A Tabela 2 apresenta o caso de uso - listar as ordens de serviço e escolher a OS desejada.

Tabela 2 - UC02 – listar as ordens de serviço e escolher a OS desejada.

Nome	UC02 – listar as ordens de serviço e escolher a OS desejada
Descrição	O usuário deseja visualizar as OS-Ordens de Serviço do fiscal selecionado e escolher a OS para realização de análise de conformidade.
Eventos	• aplicativo captura os dados da OS do fiscal de serviço; • aplicativo disponibiliza a lista de OS do fiscal de serviço; • usuário escolhe a OS desejada.
Atores	aplicativo, usuário.
Pré-Condições	• as ordens de serviço devem estar cadastradas; • fiscal de serviço foi selecionado.

Pós-Condições	1. Conclusões com sucesso: • lista de OS do fiscal é disponibilizada; • usuário escolhe a OS desejada. 2. Conclusões sem sucesso: • lista de OS do fiscal não é disponibilizada; • usuário não tem escolher OS.
Fluxo Básico	1. Capturar da nuvem as OS dos fiscal de serviço. 1.1. o aplicativo captura as OS do fiscal (A1). 2. Apresentar a lista de OS do fiscal de serviço. 2.1 o aplicativo lista as OS do fiscal (A2). 3. Escolher a OS desejada. 3.1 usuário escolhe OS (A3). 4. Fim.
Fluxos Alternativos	A1. em 1.1, caso não capture as OS do fiscal: A1.1. apresenta lista de OS disponível em cache. A2. em 2.1, caso o aplicativo não apresente lista de OS do fiscal: A2.1. apresenta a lista de OS disponível em cache. A3. em 3.1, caso usuário escolha uma OS: A3.1. o usuário é direcionado para lista de itens de OS.

Fonte: autor.

Prioridade: Essencial

Entradas e pré-condições:

- usuário deve ter selecionado o nome do fiscal de serviço desejado.

Saídas e pós-condições:

A. sucesso:

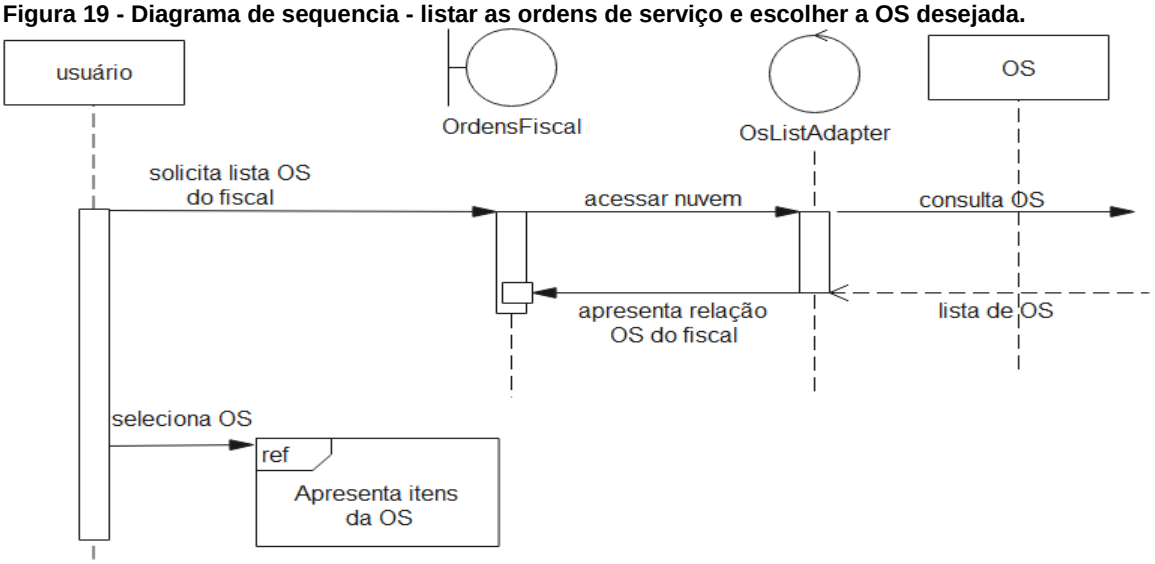
- aplicativo captura a lista de OS do fiscal de serviço da nuvem;
- o aplicativo disponibiliza a lista de OS do fiscal de serviço.

B. sem sucesso:

- lista de OS não aparece para o usuário.

Vide o respectivo diagrama de sequencia na figura 19.

Figura 19 - Diagrama de sequencia - listar as ordens de serviço e escolher a OS desejada.



Fonte: autor.

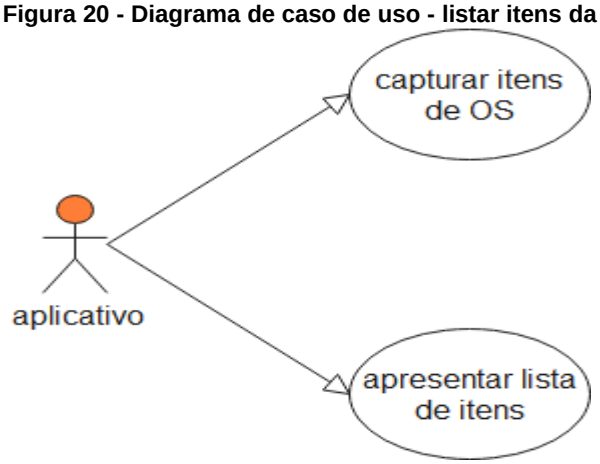
[RF03] Listar itens da OS-Ordens de Serviço

Para listar os itens da OS selecionada, estes devem estar cadastrado na nuvem.

Descrição do caso de uso:

Vide o diagrama de caso de uso na figura 20.

Figura 20 - Diagrama de caso de uso - listar itens da OS-Ordens de Serviço.



Fonte:autor.

A Tabela 3, apresenta o caso de uso - *Listar itens da OS-Ordens de Serviço*.

Tabela 3 - UC03 – Listar itens da OS-Ordens de Serviço

Nome	UC03 – Listar itens da OS-Ordens de Serviço

Descrição	O usuário deseja visualizar os itens de OS selecionado.
Eventos	• aplicativo captura os itens de serviço da nuvem; • aplicativo disponibiliza os itens da OS selecionada.
Atores	aplicativo.
Pré-Condições	• os itens de serviço da ordens de serviço devem estar cadastradas; • OS foi selecionada.
Pós-Condições	1. Conclusões com sucesso: • usuário visualiza a lista de itens da OS selecionada. 2. Conclusões sem sucesso: • usuário não visualiza a lista de itens OS. selecionada.
Fluxo Básico	1. Capturar da nuvem os itens de OS selecionada. 1.1. o aplicativo captura os itens de OS (A1). 2. Apresentar a lista de itens de OS. 2.1 o aplicativo lista os itens da OS selecionada (A2). 3. Fim.
Fluxos Alternativos	A1. em 1.1, caso não seja realizada a captura dos itens de OS: A1.1. apresenta lista de itens da OS disponível em cache. A2. em 2.1, caso o aplicativo não apresente lista de itens da OS: A2.1. apresentar a lista de itens da OS disponível em cache.

Fonte: autor.

Prioridade: Essencial

Entradas e pré-condições:

- usuário deve ter selecionado uma OS.

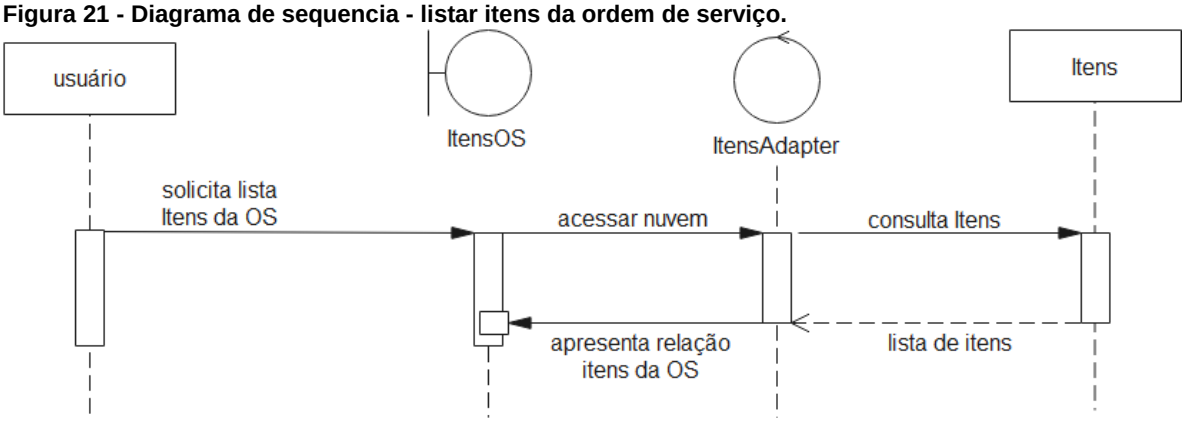
Saídas e pós-condições:

- sucesso:
  - aplicativo captura a lista de itens de OS da nuvem;
  - o aplicativo disponibiliza a lista de itens da OS selecionada.
- sem sucesso:

- o aplicativo disponibiliza a lista de itens da OS selecionada disponível em cache.

Vide diagrama de sequencia na figura 21.

Figura 21 - Diagrama de sequencia - listar itens da ordem de serviço.



Fonte: autor.

4.3.1.3 Alteração

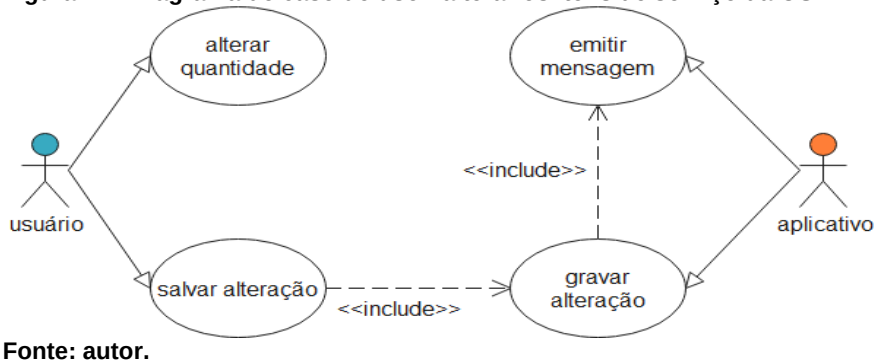
[RF04] Alterar os itens de serviço da OS

Deve ser disponibilizado ao usuário, a lista de itens de serviço da OS para que ele possa realizar a alterar os itens de serviços.

Descrição do caso de uso:

Vide o diagrama de caso de uso na figura 22.

Figura 22 - Diagrama de caso de uso - alterar os itens de serviço da OS



Fonte: autor.

A Tabela 4, apresenta o caso de uso - *Alterar os itens de serviço da OS*.

Tabela 4 - UC04 – Alterar os itens de serviço da OS

Nome	UC04 – Alterar os itens de serviço da OS

Descrição	O usuário deseja acessar os quantitativos de serviços dos itens de serviço da OS selecionada para realizar a alterações.
Eventos	• usuário seleciona o item de serviço desejado; • usuário altera o quantitativo do item de serviço; • usuário grava alteração do quantitativo de serviço.
Atores	usuário, aplicativo.
Pré-Condições	os itens de serviço devem estar disponibilizados.
Pós-Condições	1. Conclusões com sucesso: • usuário grava a alteração de quantitativo de serviço. 2. Conclusões sem sucesso: • usuário não consegue gravar a alteração de quantidade do serviço.
Fluxo Básico	1. Acessar o item desejado. 1.1. usuário acessa o item desejado. 2. Realizar a alteração da quantidade do serviço realizado. 2.1. usuário realiza alteração da quantidade. 2.2. teclado digital é recolhido. 3. Clicar no botão salvar. 3.1. usuário clica no botão salvar . 4. Gravar a alteração do item alterado. 4.1. aplicativo salva alteração realizada na memória cache. (A1). 5. Emitir mensagem informando que foi realizada a alteração. 5.1. aplicativo emite mensagem informando que foi realizada a alteração; 5.2. aplicativo salva a alteração na memória cache. 6. Fim.
Fluxos Alternativos	A1. Em 4.1, caso não seja salva a alteração: A1.1. aplicativo informa mensagem de erro.

Fonte: autor.

Prioridade: Essencial

Entradas e pré-condições:

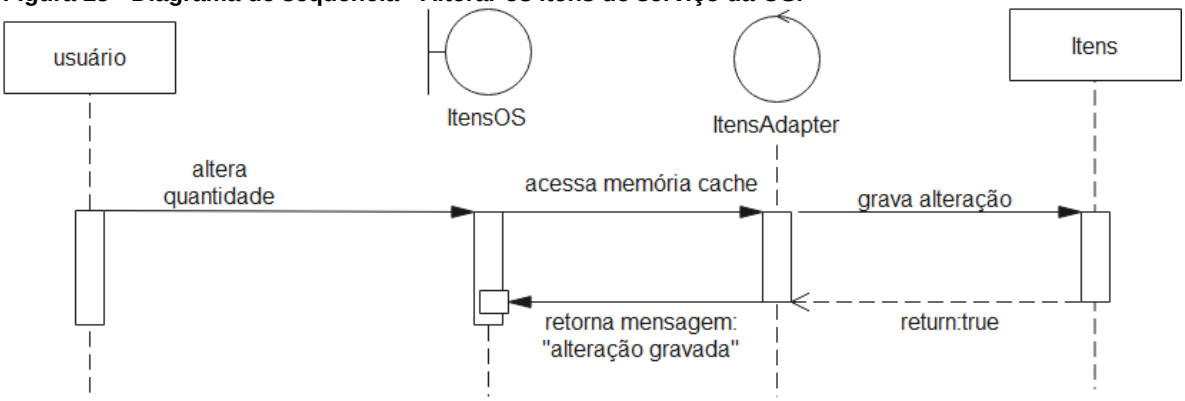
- o usuário deve ter realizado a alteração do quantitativo e selecionado o ícone de gravação.

Saídas e pós-condições:

- realizada o salvamento e apresentado informação "Alteração realizada".

Vide o diagrama de sequencia na figura 23.

Figura 23 - Diagrama de sequência - Alterar os itens de serviço da OS.



Fonte: autor.

4.3.1.4 Gravação

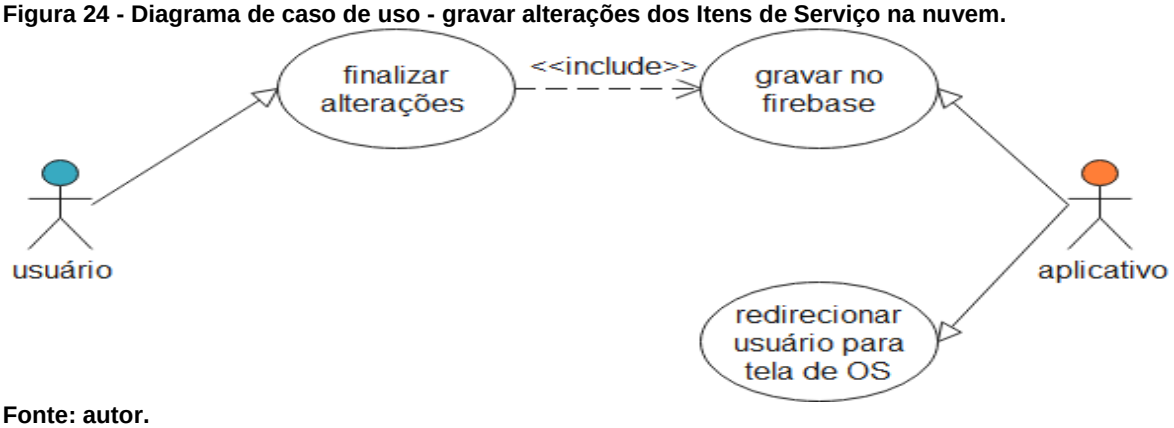
[RF05] Gravação das alterações dos Itens de Serviço na nuvem

O usuário deve salvar a alteração do item de serviço localmente para que seja possível gravar na nuvem quando da finalização da análise de conformidade.

Descrição do caso de uso:

Vide o diagrama de caso de uso na figura 24.

Figura 24 - Diagrama de caso de uso - gravar alterações dos Itens de Serviço na nuvem.



Fonte: autor.

A Tabela 5 apresenta o caso de uso - Gravar alterações dos Itens de Serviço na nuvem.

Tabela 5 - UC05 – Gravar alterações dos Itens de Serviço na nuvem.

Nome	UC05 – Gravar alterações dos Itens de Serviço na nuvem
Descrição	O usuário deseja gravar a alterações dos itens de serviço na nuvem.
Eventos	• usuário clica no ícone de gravação; • usuário recebem mensagem "Alteração gravada!"; • usuário é direcionado para tela de OS do fiscal.
Atores	Usuário.
Pré-Condições	• usuário realizou a alterações de quantitativo. • usuário clicou o botão de gravação.
Pós-Condições	1. Conclusões com sucesso: - usuário recebe mensagem: "Alteração gravada!". 2. Conclusões sem sucesso: - usuário recebe mensagem de erro.
Fluxo Básico	1. Clicar no botão "Finalizar". 1.1. usuário clica no botão finalizar. 2. Gravar alterações no Firebase. 2.1. aplicativo grava alterações no Firebase (A1). 3. Redirecionar para tela de OS do fiscal. 3.1. aplicativo redireciona para a tela de OS do fiscal. 4. Fim.
Fluxos Alternativos	A1: em 2.1, caso não seja possível gravar: A1.1: aplicativo envia mensagem de erro.

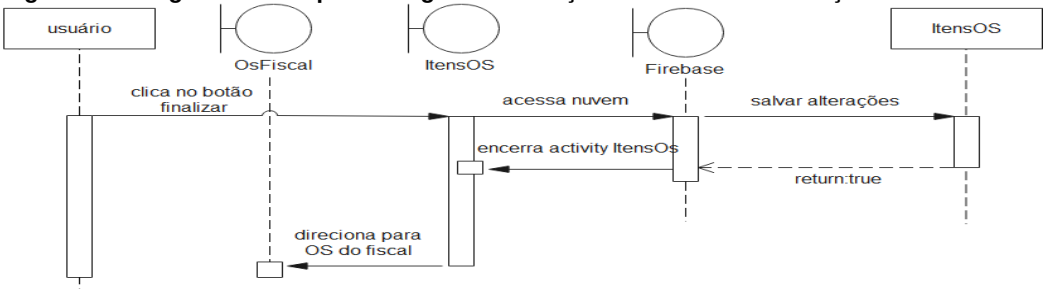
Fonte: autor.

- Prioridade: Essencial.
- Entradas e pré-condições: o usuário deve realizado a alteração e clicado no ícone salvar.

- Saídas e pós-condições: o aplicativo realiza a gravação na nuvem e redireciona usuário para tela de OS..

Vide o diagrama de sequencia na figura 25.

Figura 25 - Diagrama de sequência - gravar alterações dos Itens de Serviço na nuvem.

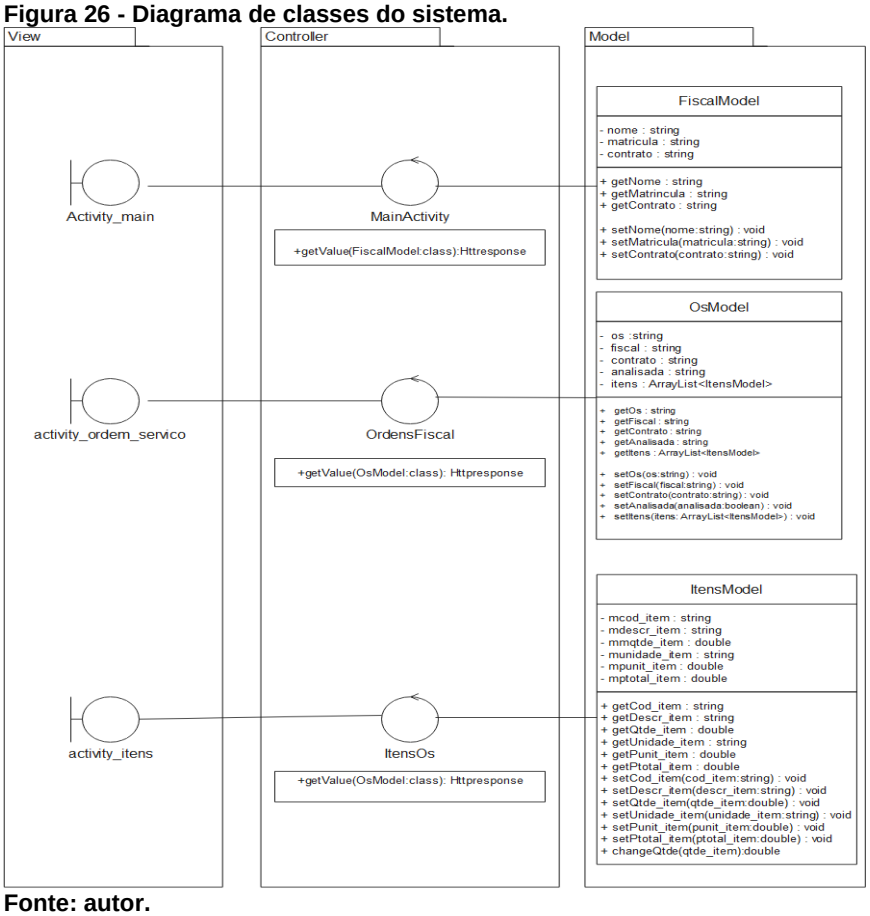


Fonte: autor.

4.3.2 Diagrama de classes

A partir dos diagramas de casos de uso obtém-se o diagrama da classes do protótipo. Vide figura 26.

Figura 26 - Diagrama de classes do sistema.



Fonte: autor.

4.4 REQUISITOS NÃO - FUNCIONAIS (NF)

[NF01] Usabilidade

O sistema possuirá uma interface de fácil navegação, acesso sem login e senha. Após acessar a tela com a lista de fiscais de serviço poderá obter a lista de OS-Ordens de Serviços correspondente ao fiscal selecionado, bem como, a lista de itens da OS. A interface é constituída de *list view* e *recycler views* que permitirão alterar o quantitativo ou não dos serviços realizados pela contratada. Caso seja necessário a alteração de quantitativos dos serviços será possível gravar a alteração. Após a análise de conformidade, os dados da OS serão gravados na nuvem e sinalizado a respectiva OS como analisada. Prioridade: A usabilidade do sistema para usuários.

[NF002] Desempenho

O sistema terá acesso 24 horas de forma rápida e segura. A gravação dos dados será de forma instantânea quando estiver on-line. Caso não seja possível, informação será gravada em cache, para gravação a posteriori de forma automática. Ainda, o sistema trabalhará de forma independente sem afetar o desempenho dos demais sistemas do dispositivo móveis. *Prioridade:* Acesso estável e seguro.

[NF003] Hardware e Software

A linguagem para desenvolvimento de sistema será o Java e os dispositivos móveis a serem utilizados deverão ter o sistema operacional Android. *Prioridade:* Linguagem de programação Java e sistema operacional Android.

[NF004] Autenticação Usuário

Por tratar-se de um protótipo de aplicativo não será necessário realizar a autenticação. Prioridade: Desejável e deverá ser realizado após aprovação do protótipo.

5 CONCLUSÃO

5.1 APLICABILIDADE

Este protótipo de aplicativo para fiscalização de obras por meio de dispositivo móvel permitiu concluir que o proposta é viável com vantagens e traz de forma quase instantânea a alteração/atualização dos dados disponíveis em nuvem. Ainda, o próprio Firebase permite gravar em cache os dados quando a nuvem não encontra-se disponível. No momento que o sistema torna-se online, a nuvem é automaticamente atualizada.

5.2 DIFICULDADES ENCONTRADAS

Foram encontradas as seguintes dificuldades durante o desenvolvimento do projeto:

- o método public void onDataChange(@NonNull DataSnapshot snapshot) do addListenerForSingleValueEvent é assíncrono, portanto não preenchia o RecyclerView. Solução: configurou-se o adapter e o recycler view dentro do método onDataChange().
- dificuldade na montagem na estrutura de dados, uma vez que, trata-se de NoSQL.
- dificuldade na utilização do recycler view, que apesar de poderosa, diferentemente do adapter de um List view, necessita-se de uma codificação mais robusta para que funcione adequadamente.
- dificuldade em capturar o nó-pai hasheado, de forma programática, onde seriam gravados a alterações realizadas no respectivos nós filhos.

5.3 SUGESTÕES DE MELHORIAS

Sugere-se o autor, como fiscal de serviço da instituição bancária, para aprimoramento do protótipo, os seguintes itens:

- criar módulo de autenticação do usuário. Para tanto, será também necessário, de forma simples, a configuração do projeto Firebase na definição de regras;
- permitir a apresentação / o inclusão de imagens no protótipo.
- permitir a inclusão de itens de serviços na OS a ser analisada quando não contempladas na OS analisada, mas que encontram-se executados pela contratada e que foram constatados na ocasião da fiscalização.

- possibilidade de utilizar o Cloud Firestore que é um banco de dados flexível e escalonável para desenvolvimento de dispositivos móveis, Web e servidores a partir do Firebase e do Google Cloud Platform. Como o Firebase Realtime Database, ele mantém seus dados em sincronia em aplicativos cliente por meio de listeners em tempo real.
- criar uma API que permita o acesso remoto a banco de dados Schema.
- criar opção para gerar relatório técnicos fotográficos e disponibilizar para os demais intervenientes do processo. Com adaptações, essa funcionalidade pode ser utilizada para fornecedores de serviços.
- permitir capturar as coordenadas geográficas do local fiscalizado.

6 REFERÊNCIAS

1. ABHIANDROID **Adapters in Android.** Disponível em: <https://abhiandroid.com/ui/adapter>. Acesso em: 25/11/2020a.

2. ALMEIDA, L. B. **Demonstração - banco firebase.** Disponível em: <https://www.youtube.com/watch?v=53fmOn9oEIk&feature=youtu.be>. Acesso em: 28/10/2020b. 18:24min.

3. ANDRADE, C. R. **Proposta e desenvolvimento de aplicativo móvel de representação de dados de EEG e PDC**. Disponível em: <https://teses.usp.br/teses/disponiveis/3/3142/tde-11072014-003224/pt-br.php>. Acesso em: 09/10/2020.

4. ANDROID**. Android Studio . Conheça o Android Studio.** Disponível em: <https://developer.android.com/studio/intro>. Acesso em: 08/10/2020a.

5. ANDROID**. Documentation for app developers**. Disponível em : <https://developer.android.com/docs>. Acesso em 08/10/2020b.

6. ANDROID. **O que é um nível de API?** . Disponível em : <https://developer.android.com/guide/topics/manifest/uses-sdk-element>. Acesso em 16/10/2020c.

7. ANDROID. **Sistema de compilação baseado em Gradle.** <https://developer.android.com/studio/intro/migrate.html?authuser=0&hl=pt-br#gradle-based_build_system>. Acesso em 16/10/2020d.

8. ANDROID. **Usos de nível de API no Android.** <https://developer.android.com/guide/topics/manifest/uses-sdk-element#uses>. Acesso em 16/10/2020e.

9. ANDROID. **Arquitetura da plataforma.** <https://developer.android.com/guide/platform?hl=pt-br>. Acesso em 17/10/2020f.

10. ARAUJO E. A. **Principais Características da Fiscalização de Obras e Serviços de Engenharia.** Disponível em: <http://www.araujoengenheiros.com.br/fiscalizacao-obras-e-servicos-engenharia>. Acesso em 02/12/2020.

11. BASSETT, L. **Introdução ao JSON - um guia para Json que vai direto ao ponto.** São Paulo: Novatec, 2015.

12. BORGES, B. R. **Desenvolvimento de Aplicação Mobile Utilizando Metodologia Ágil SCRUM.** Disponível em : <https://repositorio.ufu.br/handle/123456789/20098> . Acesso em: 09/10/2020.

13. COLETUM. **Como preencher formulários em lugares remotos onde a internet não chega . Coletum**,**Formulários.** Disponível em: <http://blog.coletum.com/preencher-formularios-sem-internet> . Acesso em 07/10/2020.

14. DALL'OGLIO, M. **Aplicativo Android para o Ambiente Univates Virtual.** Disponível em: <https://www.univates.br/bdu/handle/10737/382>. Acesso em: 09/10/2020.

15. DAPPER, A. R. **Aplicativo Mobile para Localização de Farmácias.** Disponível em : <https://www.riuni.unisul.br/handle/12345/5250>. Acesso em: 09/10/2020.

16. DEITEL, P. *et al*. **Java : como programar.** São Paulo: Pearson Education do Brasil, 2017.

17. DEITEL, P. *et al*. **Android para Programadores** - Uma abordagem baseada em aplicativos. Porto Alegre: Bookman, 2015.

18. ECLIPSE. **Documentation - Chapter 5. Client API.** Disponível em: <https://eclipse-ee4j.github.io/jersey.github.io/documentation/latest/client.html#d0e4210>. Acesso em 08/10/2020.

19. EVOLVE, L. **Prototipagem.** Disponível em : <http://moodle.dainf.ct.utfpr.edu.br/course/view.php?id=508>. Acesso em : 05/11/2020a.

20. FAGUNDES, H. *et al*. **Desenvolvimento de Ferramenta Móvel de Cadastramento e Classificação de Áreas Potencialmente Contaminadas pela Disposição de Resíduos de Construção e Demolição.** Disponível em : <http://abes-dn.org.br/anaiseletronicos/trabalhos.php?evento=36&grupo=1&pagina=116> Acesso em: 09/10/2020.

21. FILHO, F. M. G. **Desenvolvimento de um Game para Android OS.** Disponível em : <https://repositorio.unesp.br/handle/11449/119358>. Acesso em: 09/10/2020.

22. FIREBASE. **Firebase Realtime Database.** Disponível em: <https://firebase.google.com/docs/database>. Acesso em 08/10/2020a.

23. FIREBASE. **Adicionar o Firebase ao projeto para Android.** Disponível em: <https://firebase.google.com/docs/android/setup#console>. Acesso em 22/10/2020b.

24. FIREBASE. **Fazer o download do arquivo de configuração para seu app do Android.** Disponível em: <https://firebase.google.com/docs/android/setup#console>. Acesso em 22/10/2020c.

25. FIREBASE. **Bibliotecas disponíveis.** Disponível em: <https://firebase.google.com/docs/android/setup#available-libraries>. Acesso em 23/10/2020d.

26. FIREBASE. **Práticas recomendadas para a estruturação dos dados.** Disponível em: <https://firebase.google.com/docs/database/android/structure-data#best_practices_for_data_structure>. Acesso em 23/10/2020e .

27. GOODFIRMS. **What is Web Services for Mobile App?** Disponível em : <https://www.goodfirms.co/glossary/web-services-mobile-app/>. Acesso em 08/10/2020.

28. GRIFFITHS, D. *et. al.* **Use a cabeça! Desenvolvendo para Android.** Rio de Janeiro: Alta Books, 2019.

29. IRBCONTAS. **Soluções tecnológicas tornam-se aliadas na fiscalização de obras públicas.** Disponível em: <https://irbcontas.org.br/tecnologia-na-fiscalizacao-de-obras-publicas/>. Acesso em 02/12/2020.

30. JETBRAINS. **The drive to develop.** Disponível em: <https://www.jetbrains.com/pt-br/company> . Acesso em 24/08/2020.

31. LECHETA, R. R. **Web Services Restful.** São Paulo: Novatec, 2015.

32. LUCIO, D. R. **Um aplicativo para dispositivos móveis voltados para usuários de transporte público.** Disponível em : <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/247>. Acesso em: 09/10/2020.

33. MACHADO, A. P. **Desenvolvimento de aplicativo para pacientes com Transtorno do Espectro do Autismo História Social: Indo ao Dentista.** Disponível em : <https://repositorio.ufsc.br/handle/123456789/187350>. Acesso em: 09/10/2020.

34. OLIVEIRA, W. **Protótipo: o que você precisa ter em mente para montar o seu?** <https://evolvemvp.com/prototipo/>. Acesso em 30/11/2020.

35. ORACLE **Java API for WebSocket - Java Documentation.** Disponível em <https://docs.oracle.com/javaee/7/tutorial/websocket.htm#GKJIQ5_>. Acesso em 09/10/2020.

36. RED HAT. **APIs - Por que escolher a Red Hat para o gerenciamento de APIs?** Disponível em: <https://www.redhat.com/pt-br/topics/api/why-choose-red-hat-apis>. Acesso em 05/06/2020a.

37. RED HAT. **Interface de Programação de Aplicações. O que é API?** Disponível em : <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em 05/06/2020b.

38. SCHILDT, H. **Java para Iniciantes.** Porto Alegre: Bookman, 2015.

39. SMARTQUESTION. **Criando as pesquisas e formulários.** Disponível em: <https://smartquestion.com.br/howworks.html>. Acesso em 02/12/2020.

40. TREINAWEB. **O que é um Socket?** <https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets>. Acesso em 17/10/2020.

41. VIRALANDROID. **Android Linear Layout.** Disponível em: <https://www.viralandroid.com/2015/09/android-linear-layout.html>. Acesso em 17/10/2020.

42. WIKIA. **MP_Beth_Harmon.jpg.** Disponível em: <https://static.wikia.nocookie.net/the-queens-gambit/images/e/ea/MP_Beth_Harmon.jpg/revision/latest?cb=20201012200522> Acesso em 01/12/2020.

43. WIKIPEDIA. **List of acquisitions by Oracle.** Disponível em: <https://en.wikipedia.org/wiki/List_of_acquisitions_by_Oracle>. Acesso em: 16/10/2020.

44. WILL, J. C. **Aplicativo Mobile Para Busca De Restaurantes.** Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/8166>. Acesso em: 09/10/2020.

7 APÊNDICE A - As etapas de execução do protótipo

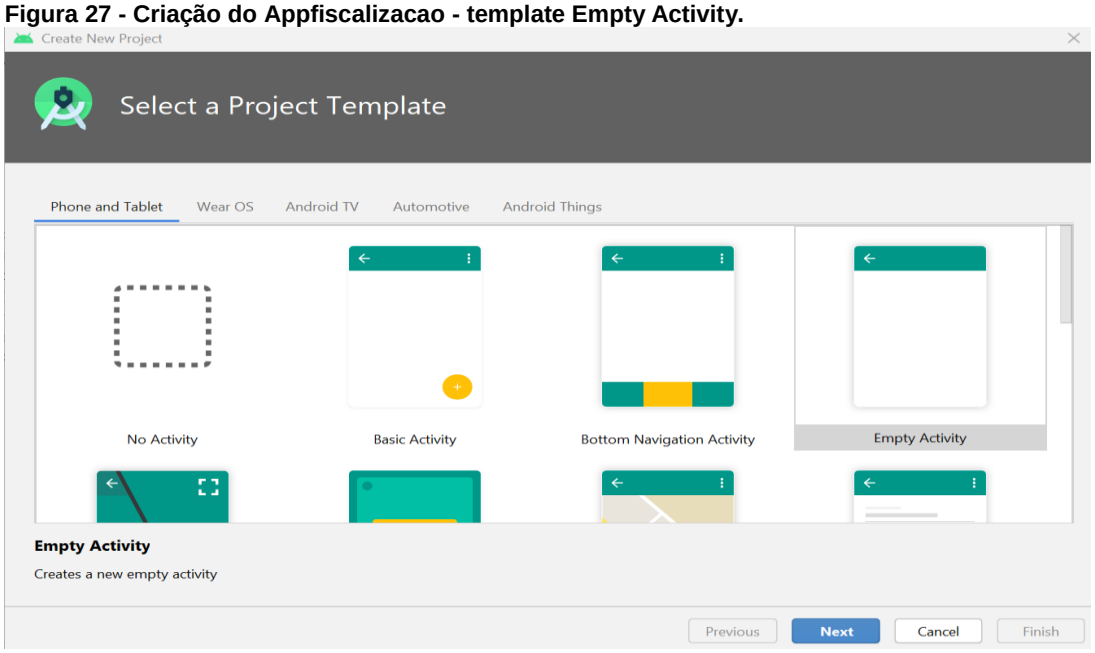
Neste apêndice são disponibilizadas as etapas de execução do Protótipo que são compreendidas por:

- Etapa 01 - criar o projeto do protótipo de aplicativo para Appfiscalizacao (aplicativo móvel) utilizando o Android Studio;
- Etapa 02 - criar um projeto do Firebase;
- Etapa 03 - vincular o projeto Firebase ao projeto de aplicativo AppFiscalizacao (protótipo Android);
- Etapa 04 - estruturar e alimentar o Realtime Database;
- Etapa 05 - desenvolver o protótipo Appfiscalizacao utilizando o Android Studio e a linguagem Java;
- Etapa 06 - teste e validação do protótipo.

ETAPA 01 - CRIAR O PROJETO DO PROTÓTIPO DE APLICATIVO PARA ANDROID APPFISCALIZACAO UTILIZANDO O ANDROID STUDIO

Segundo FIREBASE (2020b), deve ser instalado o Android Studio mais recente, que o protótipo de app para Android segmente o nível 16 da API (jelly Bean) ou versões posteriores, utilize-se o Gradle 4.1 ou versões posteriores e ser configurado um emulador ou dispositivo para executar o aplicativo. Para a criação do protótipo de aplicativo para Android Appfiscalizacao**,** cria-se um novo projeto com o Template *Empty Activity* (vide figura 27).

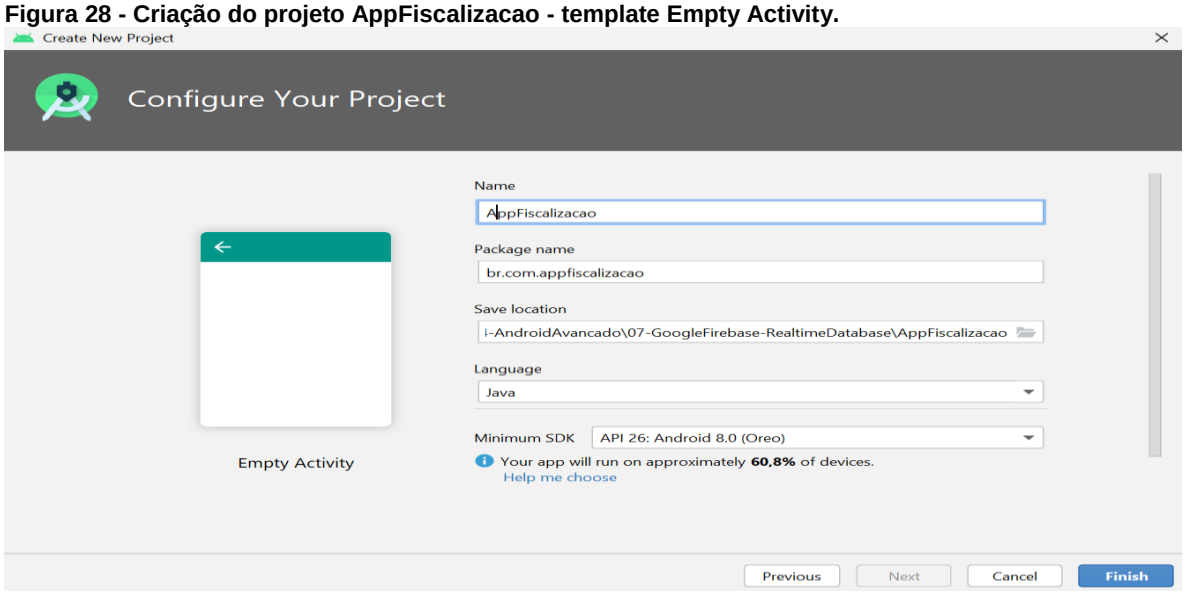
Figura 27 - Criação do Appfiscalizacao - template Empty Activity.



Fonte: Android Studio 4.1.1.

Define-se o nome do protótipo (AppFiscalizacao), nome do pacote (br.com.appfiscalizacao) e local onde será salvo o projeto, a linguagem a ser utilizada (Java) para desenvolvimento e o SDK mínino (API 26: Android 8.0-Oreo) que será atendido (vide figura 28). Após a criação do projeto AppFiscalização (protótipo) deve-se criar o projeto do Firebase.

Figura 28 - Criação do projeto AppFiscalizacao - template Empty Activity.



Fonte: Android Studio 4.1.1

ETAPA 02 - CRIAR UM PROJETO DO FIREBASE

Segue-se os procedimentos necessários para a criação do projeto Firebase (fiscalizacao). A criação do projeto Firebase envolve tarefas no **Console do Firebase** e no projeto de Android aberto. No Console do Firebase, adiciona-se um projeto e insere-se o Nome do Projeto (fiscalizacao). Ao clicar em **Criar Projeto**, o Firebase provisiona recursos automaticamento para o projeto *fiscalizacao* (vide figura 29).

Figura 29 - Console do firebase.

Fonte: FIREBASE, 2020b.

Quando o processo é concluído, o Firebase redireciona para página do projeto no Console do Firebase (vide figura 30). Criados os projetos AppFiscalizacao (protótipo Android) e Fiscalizacao (Firebase) segue para a próxima etapa que consiste na vinculação dos dois projetos.

Figura 30 - Projeto fiscalizacao criado no Firebase.

Fonte: FIREBASE, 2020c.

ETAPA 03 - VINCULAR O PROJETO FIREBASE AO PROJETO DE APLICATIVO ANDROID APPFISCALIZACAO (PROTÓTIPO)

A vinculação do projeto Firebase ao aplicativo Android (protótipo) consiste dos seguintes passos:

- registrar o AppFiscalizacao no Firebase;
- adicionar o arquivo de configuração do Firebase no AppFiscalizacao;
- adicionar o SDK básico do Firebase (core) ao AppFiscalizacao;
- sincronizar o AppFiscalizacao para que todas as dependências tenham as versões necessárias.
- executar o AppFiscalizacao para enviar ao Firebase a confirmação de integração entre os mesmos.

PASSO 1 - registrar o AppFiscalizacao no Firebase.

Na página visão geral do projeto do Console do Firebase, clica-se no ícone **Android** para iniciar-se a configuração de vinculação. No campo **Nome do pacote do Android** informa-se o nome do pacote informado na criação do projeto Android (AppFiscalizacao). Pode ser encontrado no arquivo Gradle do módulo (nível do app), geralmente app/build.gradle, na informação *applicationId* (vide figura 31).

Figura 31 - Identificação do pacote Android - módulo build.gradle (nível do app).

Fonte: Android Studio 4.1.1

A seguir clica-se em Registrar aplicativo.

PASSO 2- adicionar o arquivo de configuração do Firebase no AppFiscalizacao.

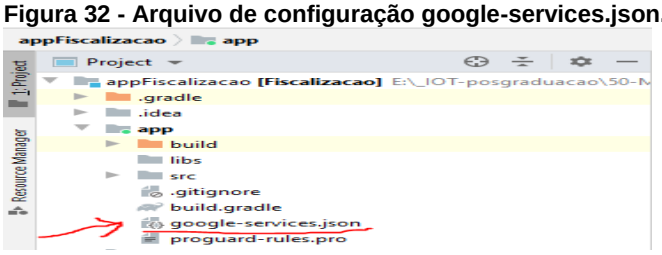
A adição do arquivo de configuração fornecido pelo Firebase realiza-se através do download do arquivo *google-services.json* quando da construção do projeto Firebase utilizando o console do Firebase.

Pode-se fazer o download do arquivo de configuração para um app do Android a qualquer momento conforme abaixo:

- Faça login no Firebase e abra seu projeto.
- Clique em  e selecione Configurações do projeto.
- No cartão Seus apps, selecione o nome do pacote na lista para o app para o qual precisa de um arquivo de configuração.
- Clique em  google-services.json.

Em seguida, mova o arquivo *google-services.json* para diretório do aplicativo (nível do app) (vide figura 32).

Figura 32 - Arquivo de configuração google-services.json.



Fonte:FIREBASE, 2020b.

Para ativar os produtos do Firebase no AppFiscalizacao (protótipo Android), adiciona-se o *plug-in google services* aos arquivos do Gradle nos nível de projeto e nível do app. No nível raiz do projeto do arquivo *Gradle (build.gradle)* (vide figura 33) dever-se-á adicionar as seguintes regras para incluir o plug-in do Google Services.

Figura 33 - Regras do arquivo Gradle(build-gradle).

```
buildscript {

  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }

  dependencies {
    // ...

    // Add the following line:
    classpath 'com.google.gms:google-services:4.2.0'  // Google Services plugin
  }
}

allprojects {
  // ...

  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    // ...
  }
}
```

Fonte: FIREBASE, 2020b.

No arquivo Gradle do seu módulo *(nível do app - app/build.gradle)*, adiciona-se a seguinte linha no *fim do arquivo* (vide figura 34).

Figura 34 - Plug-in adicionado Gradle módulo (nível app/build.gradle).

```
apply plugin: 'com.android.application'

android {
  // ...
}

// Add the following line to the bottom of the file:
apply plugin: 'com.google.gms.google-services'  // Google Play services Gradle plugin
```

Fonte: FIREBASE, 2020b.

Como utilizar-se-á o Firebase Realtime Database dever-se-á adicionar a dependência/biblioteca Android do Gradle *com.google.firebase:firebase-database:18.0.0* (vide figura 35).

Figura 35 - Dependência gradle - produto Realtime Database.

Serviço ou produto	Linha de dependência do Gradle
Plug-in do Google Play Services	com.google.gms:google-services:4.2.0
AdMob	com.google.firebase:firebase-ads:18.0.0
Analytics	com.google.firebase:firebase-core:17.0.0
App Indexing	com.google.firebase:firebase-appindexing:19.0.0
Authentication	com.google.firebase:firebase-auth:18.0.0
Cloud Firestore	com.google.firebase:firebase-firestore:20.0.0
SDK de cliente do Cloud Functions para Firebase	com.google.firebase:firebase-functions:18.0.0
Cloud Messaging	com.google.firebase:firebase-messaging:19.0.0
Cloud Storage	com.google.firebase:firebase-storage:18.0.0
Crashlytics	com.crashlytics.sdk.android:crashlytics:2.10.1
Dynamic Links	com.google.firebase:firebase-dynamic-links:18.0.0
Mensagens no app	com.google.firebase:firebase-inappmessaging:18.0.0
Exibição de mensagens no app	com.google.firebase:firebase-inappmessaging-display:18.0.0
Kit de ML: APIs Vision	com.google.firebase:firebase-ml-vision:21.0.0
Kit de ML: modelo de rotulagem de imagens	com.google.firebase:firebase-ml-vision-image-label-model:18.0.0
Kit de ML: modelo de reconhecimento facial	com.google.firebase:firebase-ml-vision-face-model:18.0.0
Kit de ML: modelo de detecção e rastreamento de objetos	com.google.firebase:firebase-ml-vision-object-detection-model:17.0.0
Kit de ML: APIs Natural Language	com.google.firebase:firebase-ml-natural-language:20.0.0
Kit de ML: modelo de identificação de linguagem	com.google.firebase:firebase-ml-natural-language-language-id-model:20.0.0
Kit de ML: modelo de tradução	com.google.firebase:firebase-ml-natural-language-translate-model:20.0.0
Kit de ML: modelo de resposta inteligente	com.google.firebase:firebase-ml-natural-language-smart-reply-model:20.0.0
Kit de ML: APIs Custom Model	com.google.firebase:firebase-ml-model-interpreter:20.0.0
Kit de ML: API AutoML Vision Edge	com.google.firebase:firebase-ml-vision-automl:17.0.0
Monitoramento de desempenho	com.google.firebase:firebase-perf:18.0.0
Realtime Database	com.google.firebase:firebase-database:18.0.0

Fonte: FIREBASE, 2020d.

PASSO 3 - adicionar o SDK básico do Firebase (core) ao AppFiscalizacao.

É possível adicionar qualquer um dos produtos Firebase compatíveis com Android, a começar pelo SDK básico do firebase (*com.google.firebase:firebase-core),* que oferece a funcionalidade do Google Analytics para Firebase. Funcionalidade recomendada pelo Firebase.

No arquivo Gradle do módulo (nível do app - app/build.gradle), adiciona-se a dependência do SDK básico do Firebase (vide figura 36). A partir daqui poder-se-á fazer a sincronização.

Figura 36 - Dependência do SDK básico - nível-app/build.gradle.

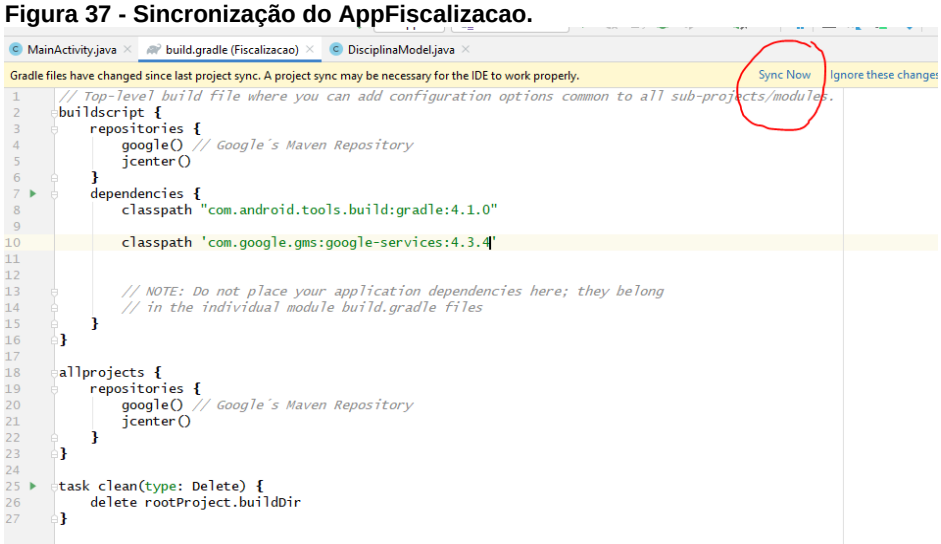
```
dependencies {
 // ...
 implementation 'com.google.firebase:firebase-core:17.0.0'

 // Getting a "Could not find" error? Make sure that you've added
 // Google's Maven repository to your root-level build.gradle file
}
```

Fonte: FIREBASE, 2020b.

PASSO 4 - sincronizar o AppFiscalizacao para que todas as dependências tenham as versões necessárias.

Dever-se-á sincronizar o AppFiscalizacao para garantir que todas as dependências tenham as versões necessárias (vide figura 37).

Figura 37 - Sincronização do AppFiscalizacao.



Fonte: FIREBASE, 2020b.

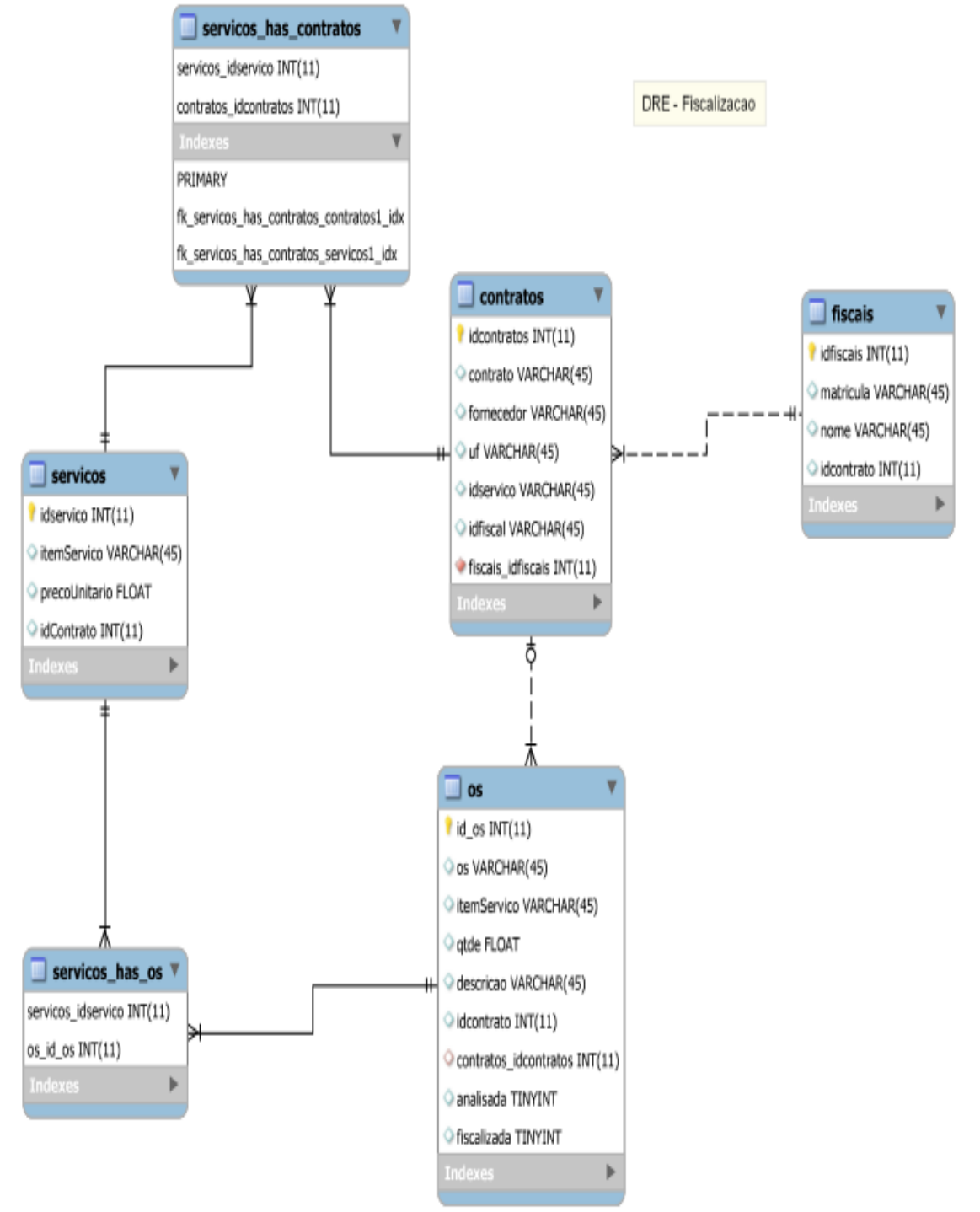
PASSO 5 - executar o AppFiscalizacao para enviar ao Firebase a confirmação de integração entre os mesmos.

Ao executar o AppFiscalizacao, os registros do dispositivo exibirão a verificação do Firebase de que a inicialização foi concluída. Executado o app no emulador com acesso à rede, o Console do Firebase notifica que a conexão do aplicativo está completa.

ETAPA 04 - ESTRUTURAR E ALIMENTAR O REALTIME DATABASE

Para gerar a estrutura de dados e alimentar o Realtime Database tomar-se-á como referência o diagrama de relacionamento de entidades DRE - fiscalizacao (vide figura 38).

Figura 38 - DRE-Diagrama de Relacionamento de Entidades base.



Fonte: autor

Considerando as práticas recomendadas para a estruturação de dados mencionada no item 3.9.1, adotar-se-á a seguinte estrutura de dados JSON apresentada na figura 39 :

Figura 39 - Estrutura de dados JSON adotada para o Realtime Database.


```
{
  "fiscal" : {
    "-MMYuXsWe7I5_NuU2ZRv" : {
      "contrato" : "202074201111",
      "matricula" : "111111",
      "nome" : "Margareth"
    },
    "-MMYuXsuJKU7l9naKxJf" : {
      "contrato" : "202074202222",
      "matricula" : "222222",
      "nome" : "Jessica"
    },
    "-MMYuXsvKvKly0W6ZPKj" : {
      "contrato" : "202074203333",
      "matricula" : "333333",
      "nome" : "Anya"
    }
  },
  "os" : [ {
    "analisada" : true,
    "contrato" : "202074201111",
    "fiscal" : "Margareth",
    "itens" : [ {
      "cod_item" : "101",
      "descr_item" : "mola hidráulica",
      "punit_item" : 120,
      "qtde_item" : 1,
      "unidade_item" : "un"
    }, {
      "cod_item" : "002",
      "descr_item" : "deslocamento",
      "punit_item" : 1.7,
      "qtde_item" : 10,
      "unidade_item" : "km"
    } ],
    "os" : "001"
  }, {
    "analisada" : true,
    "contrato" : "202074202222",
    "fiscal" : "Jessica",
    "itens" : [ {
      "cod_item" : "201",
      "descr_item" : "pintura acrílica",
      "punit_item" : 2.5,
```

```
45          "qtde_item" : 10,
46          "unidade_item" : "m2"
47        }, {
48          "cod_item" : "002",
49          "descr_item" : "deslocamento",
50          "punit_item" : 1.7,
51          "qtde_item" : 10,
52          "unidade_item" : "km"
53        } ],
54        "os" : "002"
55      }, {
56        "analisada" : true,
57        "contrato" : "202074203333",
58        "fiscal" : "Anya",
59        "itens" : [ {
60          "cod_item" : "301",
61          "descr_item" : "porta de vidro",
62          "punit_item" : 850,
63          "qtde_item" : 4,
64          "unidade_item" : "un"
65        }, {
66          "cod_item" : "002",
67          "descr_item" : "deslocamento",
68          "punit_item" : 1.7,
69          "qtde_item" : 11,
70          "unidade_item" : "km"
71        } ],
72        "os" : "003"
73      } ]
```

Fonte: autor.

ETAPA 05 - DESENVOLVER O PROTÓTIPO APPFISCALIZACAO UTILIZANDO O ANDROID STUDIO E A LINGUAGEM JAVA.

Serão desenvolvidas as seguintes telas que permitirão a navegação do protótipo de aplicativo.

- **Tela inicial** - apresenta a relação de fiscais de serviços que apresentam OS-ordens de serviço pendentes de análise, fiscalização e aprovação ou não dos pagamentos dos serviços realizados.
- **Tela de OS** - apresenta a relação de ordens de serviços do fiscal com a situação analisada ou não.
- **Tela de itens de OS** - apresenta os itens da OS-ordem de serviços que serão analisadas e dadas ou não conformidade de pagamento dos serviços fiscalizados.

CRIAÇÃO DO PROJETO APPFISCALIZAÇÃO

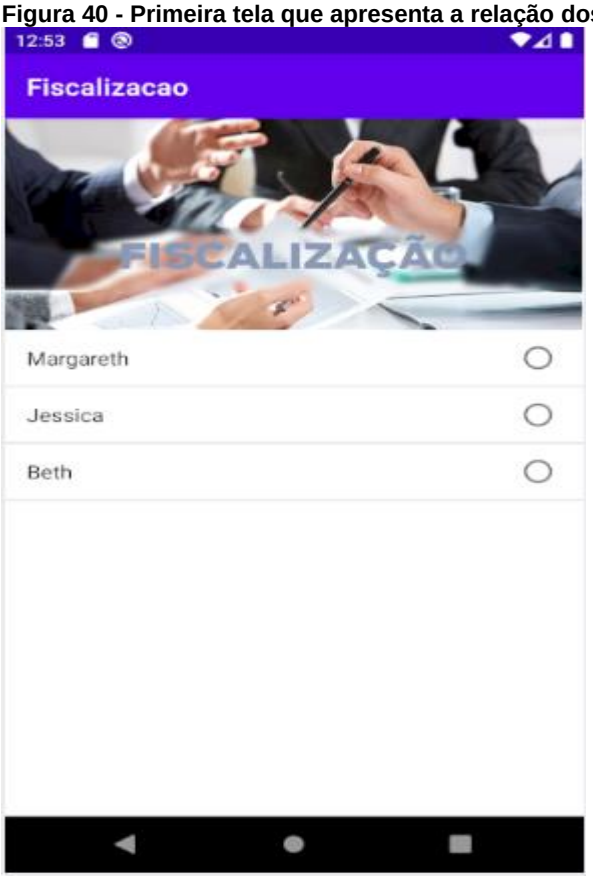
A criação do projeto *appFiscalizacao* será feita a partir de Empty activity cujo o *MainActivity.java* terá a tela inicial do projeto. Foram adotadas as seguintes configurações e implementações:

- IDE Android Studio 4.1.1

- Linguagem Java.
- minSdkVersion 26
- targetSdkVersion 29
- implementation platform('com.google.firebase:firebase-bom:25.12.0');
- implementation 'com.google.firebase:firebase-analytics';
- implementation 'com.google.firebase:firebase-core: 18.0.0';
- implementation 'com.google.firebase:firebase-database:19.5.1';

Segue na figura 40, a tela inicial do protótipo de aplicativo.

Figura 40 - Primeira tela que apresenta a relação dos fiscais de serviço.



Fonte: autor

Apresenta a relação de fiscais de serviços que tem OS-ordens de serviço pendente de análise, fiscalização e aprovação ou não dos pagamentos dos serviços realizados.

Nesta tela, cria-se uma Activity chamada *MainActivity.java* e um layout *activity_main.xml* que receberá todas as view dela.

No arquivo subdiretório *themes* do diretório *values*, é definido as cores primárias e secundárias que será utilizado na aplicação, bem como o tema do ActionBar que será mostrada em todas activities do protótipo.

O tema adotado para a ActionBar é Theme.MaterialComponents.DayNight.DarkActionBar.

Ao ser criada a activity, a activity é mencionada no arquivo AndroidManifest.xml. Como está activity é a principal, ela recebe as especificações <intent-filter> e <action> abaixo.

```
<activity
        android:name=".activity.MainActivity"
        android:label="@string/app_name"
        android:theme="@style/Theme.Fiscalizacao.ActionBar">
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
</activity>
```

É inserido no layout activity_main um imageview e um listview.

O imageview constará a imagem *img_equipe_fiscalizacao.png* e o texto "Fiscalização" (vide figura 41).

Figura 41 - Imagem inserida no componente ImageView da Tela Inicial.

Fonte: autor.

No ListView será apresentado a lista dos fiscais de serviços que será capturado do Firebase Realtime Database.

Antes de consultar o Firebase, cria-se uma variável *database* para armazenar a instância do Firebase.

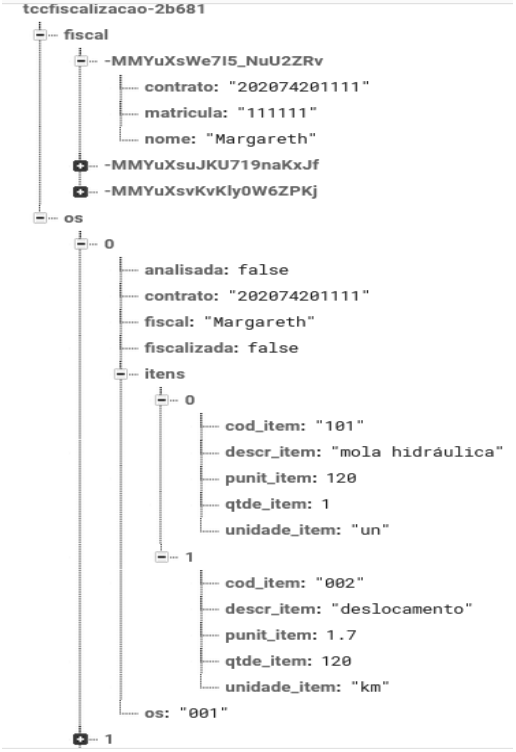
Declara-se uma variável *ref* da classe DatabaseReference para armazena como referência o nó-raiz do banco de dados Firebase-RealtimeDatabase.

Para receber os fiscais capturados, cria-se a classe FiscalModel.class com a seguinte estrutura:

```
public class FiscalModel {

private String nome;
private String matricula;
private String contrato;

// construtores
public FiscalModel() {
}

public FiscalModel( String nome, String matricula, String contrato){
    this.nome = nome;
        this.matricula = matricula;
        this.contrato = contrato;
    }
    // getters
    public String getNome(){
        return this.nome;
    }
    public String getMatricula(){
        return this.matricula;
    }
    public String getContrato(){
        return this.contrato;
    }
    // setters
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void setMatricula(String matricula){
        this.matricula = matricula;
    }
    public void setContrato(String contrato){
        this.contrato = contrato;
}

    @Override
    public String toString() {
        return "FiscalModel{" +
                "nome='" + nome + '\'' +
                ", matricula='" + matricula + '\'' +
                ", contrato='" + contrato + '\'' +
                '}';
    }
} // fim da classe FiscalModel
```

Na figura 42, tem-se a estrutura do banco de dados no Firebase Realtime Database.

Figura 42 - Estrutura do banco de dados no Firebase Realtime Database.



Fonte: FIREBASE, 2020a.

No Layout *activity_main.xml* foram declarados o ImageView e ListView.

Foi inserida uma imagem *img_equipe_fiscalizacao.png* no diretório *drawable* do projeto para que o ImageView possa acessá-la pelo atributo <*android:src="@drawable/img_equipe_fiscalizacao"*>.

Para o ListView, após a declaração de uma variável do tipo ListView e a atribuição de id para o mesmo no arquivo *Layout activity_main.xml*, vincula-se a variável objeto ao view utilizando o método *findViewById()* no método *onCreate()* da Activity *MainActivity.java*.

No método *onStart()* do *MainActivity*, após sua execução, o usuário visualiza a activity na tela do aplicativo. Portanto, nesse instante é inserido o código de consulta ao bando de dados Firebase e a captura dos dados por meio da classe *FiscalModel.class*.

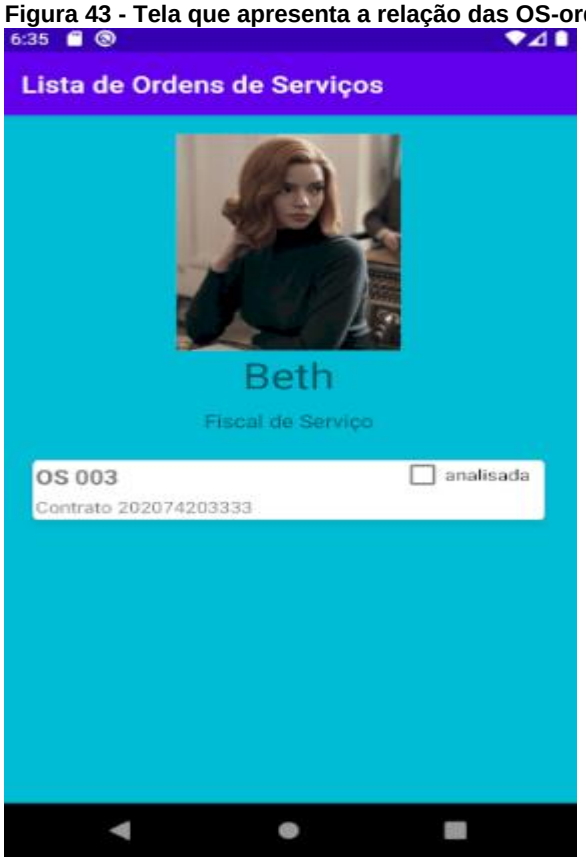
Todo nome de fiscal é adicionado ao *Arraylist<String>* declarado no início da public class *MainActivity()*.

Uma vez adicionado todos os fiscais existentes no ArrayList, vincula-se ao Adapter para que ele possa fazer a ponte entre os dados e o ListView. Vinculado o adapter tem-se a lista dos fiscais cadastrados no ListView. Para que se possa selecionar o fiscal desejado através do ListView é criado o método *aguardaClickListView*().

No método *aguardaClickListView()*, é criado uma variável do tipo *Adapter.OnItemClickListener* que ao ser instanciada aguarda um click a ser executado pelo usuário. Ao selecionar o fiscal, o método *OnItemClick()* é acionado e criado uma Intent que levará a uma nova Acitvity, a *OrdensFiscal.java Activity*. Nessa intent será enviada o nome do fiscal selecionado de forma a permitir que a Activity destino apresente as respectivas Ordens de Serviço.

Na figura 43, a tela de OS-Ordens de Serviços apresenta a relação de ordens de serviços do fiscal selecionado com um checkbox demonstrando se a OS foi analisada ou não.

Figura 43 - Tela que apresenta a relação das OS-ordens de serviço pendentes.



Fonte: autor.

No instante em que é criada a activity é executado o método onCreate(), onde foram disponibilizadas dois métodos:

- mostrarNomeFiscal();
- listarOs(nome);

O método *mostrarNomeFiscal()* resgata o nome do fiscal enviado pela Activity *MainActivity* e o apresenta em um TextView.

Na criação da activity *OrdensFiscal.java* foi criado conjuntamente o layout *activity_ordem_servico.xml*.

No layout *activity_ordem_servico.xml* foi definido os componentes view que compõem a tela da activity. De forma estática, foi definida a imagem dos fiscais de serviço para efeito de ilustração e apresentada na ImageView.

O método *listarOS(fiscal)* captura as OS-Ordens de Serviço ao definir uma query abaixo do nó *os* cujo nome do fiscal é passado pelo parâmetro *fiscal*. Vide figura 44.

Figura 44 - query realizada para captura das OS do fiscal de serviço

```
Query query1 =
ref.child("os").orderByChild("fiscal").equalTo(nome);
```

Fonte: autor.

Através do método *addListenerForSingleValueEvent* aplicado ao objeto Query definido obtém-se um DataSnapshot dos dados encontrados pela query.

Esse DataSnapshot é processado pelo método void *onDataChange()* onde se percorre todo Snapshot utilizando um loop for cujo o tamanho da pesquisa é dado pelo *snapshot.getChildren*. Vide figura 45.

Figura 45 - loop for que percorre todo Snapshot.

```
for(DataSnapshot objSnapshot:snapshot.getChildren()) {...}
```

Fonte: autor.

Ao percorrer o DataSnapshot, resgata-se o nó-pai em que estão dispostas todas as OS do fiscal cujo o nome foi passado como parâmetro ao método *listarOs(String fiscal)* através do método *getKey()*. Vide figura 46.

Figura 46 - método getKey() utilizado para resgatar o nó-pai.

```
String key = objSnapshot.getKey();
```

Fonte: autor.

Criou-se antecipadamente, a classe *OsModel.class* que será utilizado para resgatar os dados de cada OS.

Para cada OS encontrada é capturada todos os seus dados através do método *getValue(OSModel.class)* e atribuída a uma variável obj do tipo *OSModel*.

Todo objeto é adicionado a uma *ArrayList <OsModel> listOs* que será enviado ao objeto OsListAdapter adapter que será instanciado.

O adapter é responsável em fazer a ligação entre os dados e o RecyclerView que deve ser criado e configurado para que liste todas as OS pertencentes ao fiscal selecionado. O RecyclerView é populado com as OS-Ordens de Serviço do fiscal de serviço selecionado.

Para que sejam disponibilizados todos as OS do fiscal selecionado, cria-se o Layout *cardview_adapter_os.xml* onde estão encapsulados pelo componente CardView, as views que disponibiliza os textos e checkBox. Este informa se a OS já foi analisada ou não.

Além disso, cria-se a classe *OsListAdapter.java* que estende a *RecyclerView.Adapter* cujo o objeto de tipo é um *OsListAdapter.OsViewHolder*. Esta contém os métodos necessários para a configuração do Adapter a ser vinculado ao RecyclerView.

- método construtor OsListAdapter(List<OsModel> lista);
- método público OsViewHolder onCreateViewHolder(ViewGroup parent, int viewType);
- método público onBindViewHolder(OsViewHolder holder, final int position);
- método public int *getItemCount()*;
- public interface OnItemClickListener;
- public void setOnItemClickListener(OnItemClickListener listener);
- public static class OsViewHolder extends RecyclerView.ViewHolder{};
-

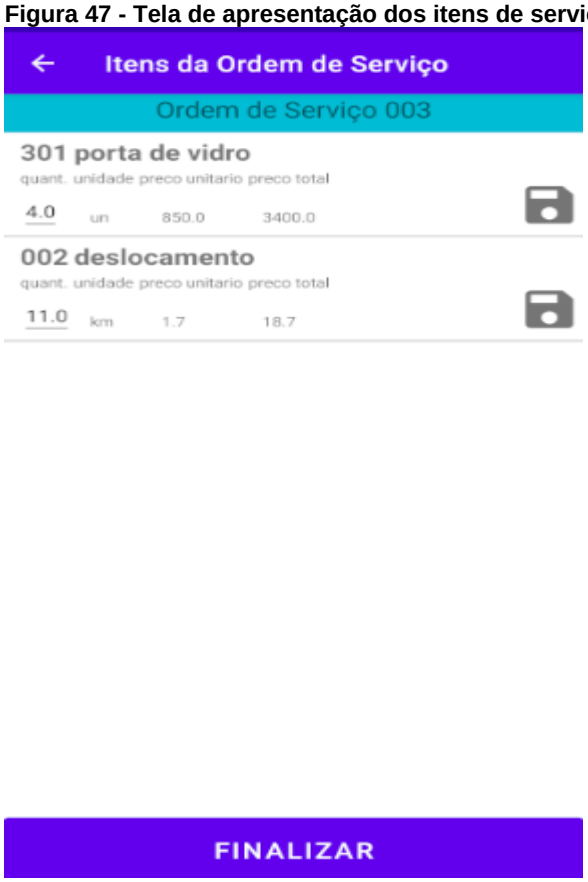
Feito essas configurações do *OsListAdapter*, instancia-se o objeto *OsListAdapter* e realiza-se as seguintes definições:

```
RecyclerView recyclerView =
        findViewById(R.id.osList_recycler);
RecyclerView.LayoutManager layoutManager =
        new LinearLayoutManager(getApplicationContext());
recyclerView.setLayoutManager(layoutManager);
recyclerView.setHasFixedSize(true);
recyclerView.addItemDecoration(new
DividerItemDecoration(getApplicationContext(), LinearLayout.VERTICAL));
recyclerView.setAdapter(adapter);
adapter.setOnItemClickListener(new OsListAdapter.OnItemClickListener()
{..}
```

O método *setOnItemClickListener* na implementação do *onItemClick(int position)* encaminha por intent à activity *ItensOS.java*, os dados da OS selecionada (nome do fiscal, número da OS e nó-pai da OS onde estão disponibilizados o itens da OS abaixo do nó-filho items).

Segue a tela de apresentação dos itens de serviço da Ordem de Serviço (vide figura 47).

Figura 47 - Tela de apresentação dos itens de serviço da Ordem de Serviço.



Fonte: autor.

A Tela de Itens de OS-Ordem de Serviço a ser analisada apresenta os itens da OS-ordem de serviços que serão analisadas e dadas ou não conformidade de pagamento pelo fiscal de serviço.

A activity *ItensOS.java* é acompanhada de Layout *activity_itens.xml* onde estão encapsulados dentro de um constraint Layout, um TextView (*os_number*), um RecyclerView (*osList_recycler*) e um Button (*button_conforme*). O TextView conterá o número da OS e o RecyclerView, os dados do itens da OS selecionada.

Para iniciar a activity *ItensOS.java* criam-se o Layout *cardview_adapter_os.xml* e o Adapter *ItensAdapter*. O Layout serve para disponibilizar os dados dos itens da OS, no TextView, na RecyclerView e consequentemente, na activity e o Adapter para fazer a conexão dos dados dos itens da OS com o RecyclerView.

No Layout *cardview_adapter_os.xml* define-se os views que contém os dados, textos e campo editável (quantidade do item) e drawable (ícone de salvar) que serve de container para cada item de OS a ser apresentado pelo RecyclerView *osList_recycler*.

Para o Adapter, cria-se a classe *ItensAdapter.java* que estende a RecyclerView.Adapter cujo o objeto-tipo é um *ItensAdapter.OsViewHolder*. Essa contém os métodos necessários para a configuração do Adapter a ser vinculado ao RecyclerView.

- interface onItemClickListener{ ... };
- public void setOnItemClickListener(ItensAdapter.OnItemClickListener listener);
- public ItensAdapter(List<ItensModel> detailList) {...};
- public static class OsDetailsViewHolder extends RecyclerView.ViewHolder{};
- método construtor OsDetailViewHolder (List<ItensModel> detailsList);
- método público OsDetailViewHolder onCreateViewHolder(ViewGroup parent, int viewType);
- método público onBindViewHolder(OsDetailViewHolder holder, int position);
- método public int *getItemCount()*;
- public interface OnItemClickListener{..}.

Uma classe *ItensModel* é criada onde conterá os atributos de itens da OS e seus getters e setters para armazenar os dados obtidos das queries realizadas no Firebase.

No método *onCreate* da activity *ItensOS.java* são executados 3 (três) métodos:

- mostraOsSelecionada()
- mostraItens(os_selecionada)
- mostraButtonConforme()

O método *mostraOsSelecionada()* através do método *getStringExtra* de um objeto Intent, recupera o nome do fiscal, a OS selecionada e a chave key sob o qual estão os itens da OS-Ordem de Serviço selecionada. O número da OS é atribuída ao TextView (*os_number*) criado.

O método *mostraItens(os_selecionada)* realiza uma *query* que captura todos os itens da OS selecionada. Vide figura 48.

Figura 48 - Query que captura todos os itens da OS selecionada.

```
Query query1 = ref.child("os")
                  .orderByChild("os")
                  .equalTo(os_selecionada);
```

Fonte: autor.

Ao se aplicar o método *addListenerForSingleValueEvent(new ValueEventListener(){...}* este retorna um DataSnapshot dos itens da OS selecionada. Vide figura 49.

Figura 49 - Query que retorna um DataSnapshot dos itens da OS selecionada.

```
query1.addListenerForSingleValueEvent(new ValueEventListener() {...}
```

Fonte: autor.

No método de implementação void *onDataChange(@NonNull DataSnapshot snapshot){...}*, instancia-se um iterable *it* que captura todos os filhos do snapshot. Vide figura 50.

Figura 50 - Iterable que captura todos os filhos de um snapshot.

```
Iterable<DataSnapshot> it = snapshot.getChildren();
```

Fonte: autor.

Apesar de ser uma OS, essa foi a forma de obter os itens do dado da OS selecionada.

Aplica-se o método *getItens.size()* para determinar a quantidade de itens da OS e utiliza-se loop for para obter o código do item, descrição, quantidade, unidade, preço unitário do item e com tais dados, calcular o preço total do item.

Com os dados capturados, instancia-se o objeto *ItensModel* e o atribui-se a variável *item*. E finalmente, adiciona-se o objeto *item* ao Arraylist *listItens* que será passado ao objeto ItensAdapter criado. O ItensAdapter é que realizará a conexão dos dados com o componente Recycler View *oslist_recycler.*

Instancia-se o objeto ItensAdapter e realiza-se as seguintes definições:

```
ItensAdapter mAdapter = new ItensAdapter(listItens);
RecyclerView recyclerView = findViewById(R.id.osList_recycler);
RecyclerView.LayoutManager layoutManager =
new LinearLayoutManager(getApplicationContext());
recyclerView.setLayoutManager(layoutManager);
recyclerView.setHasFixedSize(true);
recyclerView.addItemDecoration(new
DividerItemDecoration(getApplicationContext(), LinearLayout.VERTICAL));
recyclerView.setAdapter(mAdapter);
mAdapter.setOnItemClickListener(new OsListAdapter.OnItemClickListener()
{..}
```

O método *setOnItemClickListener* na implementação do *onSaveClick(int position, double qtde)* efetiva a alteração realizada e comunica por meio de uma Toast (de pouca duração) a alteração realizada.

O método *mostraButtonConforme()* executa através do método *buttonConforme.setOnClickListener* e seu método de implementação *onClick*, a gravação das quantidades alteradas no Firebase e através de uma intent informa o nome do fiscal e retorna para a activity *OrdensFiscal.java*.

Durante o desenvolvimento e teste apresentava-se a mensagem:

> *W/PersistentConnection: pc_0 - Using an unspecified index. Your data will be downloaded and filtered on the client. Consider adding '".indexOn": "fiscal"' at os to your security and Firebase Database rules for better performance.*

Na realidade, devido aos constantes acessos aos dados de OS, o firebase informa a necessidade da criação de índice para segurança e melhor performance. Portanto, criou-se o índice solicitado no Firebase conforme figura 51 abaixo:

Figura 51 - Criação de índices para o nós os.

```
{
  "rules": {
    ".read": true,
    ".write": true,
      "os" : {
        ".indexOn" : ["fiscal","itens","os"]
      }
  }
}
```

Fonte: FIREBASE, 2020a.

Durante a alteração dos itens da ordens de serviço o usuário não percebe que foi gravado a alteração de quantidades alteradas. Para solucionar o problema adicionou-se um Toast para informar que a alteração foi realizada ao clicar no ícone de salvar.

Percebeu-se que estava tendo uma relativa demora na apresentação da activity OrdensFiscal.java. Ao verificar o problema, percebeu-se que o nome do fiscal não estava sendo informado para a activity *OrdensFiscal.java*. Corrigido o problema, a velocidade de renderização melhorou consideravelmente.

Verificou-se a necessidade de inserir o button UP. Ao navegar pelo protótipo de aplicativo, percebeu-se a necessidade movimentar entre as telas, portanto inseriu-se o button UP para voltar de uma activity para outra.

Percebeu-se que o botão *finalizar* não retornava para a activity OrdensFiscal. Inclui-se uma intent no método *mostraButtonConforme* permitindo assim, o retorno a activity *OrdensFiscal* de forma direta.

8 APÊNDICE B - codificação do protótipo de aplicativo

Este apêndice demonstra de forma direta, a codificação do protótipo desenvolvido para a plataforma Android® utilizando o Android Studio®, como IDE (Integrated Development Enviroment), e a linguagem Java®.

Segue as configurações necessárias para o desenvolvimento do protótipo:

A versão Java SE a ser utilizada será a:

- java 11.0.7 2020-04-14 LTS
- Java(TM) SE Runtime Environment 18.9 (build 11.0.7+8-LTS)
- Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.7+8-LTS, mixed mode)

A versão do Android Studio a ser utilizada será o 4.1.1. para 64 bits.

- O nível da API a ser utilizado será minSdkVersion 26 e targetSdkVersion 26
- Versão do gradle 4.1.0

O desenvolvimento de protótipo de aplicativo para fiscalização de obras utilizando um Firebase Realtime Database realizou-se conforme os seguintes passos:

- Criação de Projeto no Firebase TCC-fiscalizacao, inclusão da base de dados no Firebase Realtime Database, configuração do Android Studio;
- Vinculação dos drawables img_equipe_fiscalizacao.png e anyataylorjoy.png e ic_save.xml;
- Criação das Classes FiscalModel.java, ItensModel.java, OsModel.java
- Criação da activity MainActivity, layout activity_main.xml, definição do themes.xml, definição de strings.xml do projeto;
- Criação da activity OrdensFiscal.java, layout activity_ordem-servico.xml, cardview_adapter_os.xml, OsListAdapter;
- Criação da activity ItensOS.java, layout activity_itens.xml, cardview_adapter_itens.xml, ItensAdapter;
- Inclusão do button UP e criação de índices no Realtime Database.

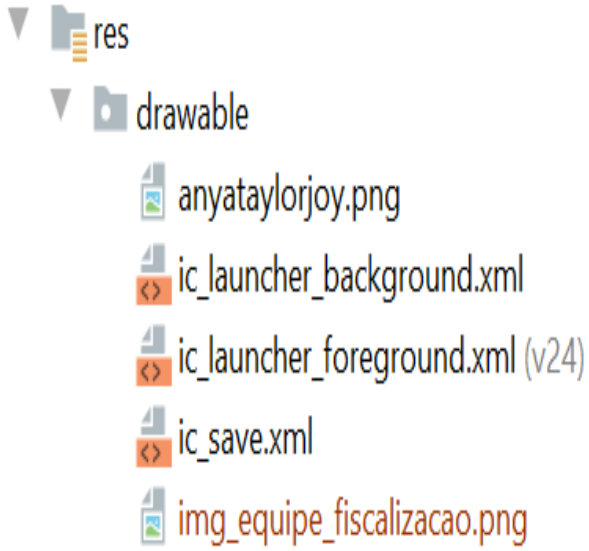
CRIAÇÃO DE PROJETO NO FIREBASE TCC-FISCALIZACAO, INCLUSÃO DA BASE DE DADOS NO FIREBASE REALTIME DATABASE, CONFIGURAÇÃO DO ANDROID STUDIO

Essa etapa está bem detalhada na Etapa 01 - Criar o projeto do protótipo de aplicativo para Android Appfiscalizacao utilizando o Android Studio disponibilizado no apêndice A.

VINCULAÇÃO DOS DRAWABLES IMG_EQUIPE_FISCALIZACAO.PNG E ANYATAYLORJOY.PNG E IC_SAVE.XML;

Vinculou-se ao pacote *res\drawable* do projeto os *img_equipe_fiscalizacao.png* e *anyataylorjoy.png* e *ic_save.xml* (vide figuras 52, 53, 54 e 55).

Figura 52 - Arquivos drawables utilizados.



Fonte: Android Studio 4.1.1

Figura 53 - Drawable img_equipe_fiscalizacao.png - 825 x 367 pixels.

Fonte: Android Studio 4.1.1

Figura 54 - Drawable anyataylorjoy.png - 319x380 pixels.

Fonte: WIKIA, 2020.

Figura 55 - Vector - 24 x 24dp.

Fonte: Android Studio 4.1.1

CRIAÇÃO DAS CLASSES FISCALMODEL.JAVA, ITENSMODEL.JAVA, OSMODEL.JAVA

As classes criadas Classes *FiscalModel.java, ItensModel.java, OsModel.java* estão representadas na figura 56, 57 e 58, respectivamente.

Figura 56 - Classe FiscalModel.java.

```
package br.com.fiscalizacao.model;

public class FiscalModel {

    private String nome;
    private String matricula;
    private String contrato;

    public FiscalModel() {
    }

    public FiscalModel( String nome, String matricula, String contrato){
        this.nome = nome;
        this.matricula = matricula;
        this.contrato = contrato;
    }

    // getters
    public String getNome(){
        return this.nome;
    }

    public String getMatricula(){
        return this.matricula;
    }

    public String getContrato(){
        return this.contrato;
    }

    // setters
    public void setNome(String nome) {
        this.nome = nome;
    }

    public void setMatricula(String matricula){
        this.matricula = matricula;
    }

    public void setContrato(String contrato){
        this.contrato = contrato;
    }

} // fim da classe FiscalModel
```

Fonte: autor.

Figura 57 - Classe OsModel.java.

```
package br.com.fiscalizacao.model;

import java.util.ArrayList;
import java.util.List;

public class OsModel {

    private String os;
    private String fiscal;
    private String contrato;
    private Boolean fiscalizada;
    private Boolean analisada;
    private List<ItensModel> itens = new ArrayList<>();

    public OsModel(String os, String fiscal, String contrato, Boolean fiscalizada, Boolean analisada, List<ItensModel> itens) {
        this.os = os;
        this.fiscal = fiscal;
        this.contrato = contrato;
        this.fiscalizada = fiscalizada;
        this.analisada = analisada;
        this.itens = itens;
    }

    public String getOs() {
        return os;
    }

    public String getFiscal() {
        return fiscal;
    }

    public String getContrato() {
        return contrato;
    }
```

```
public Boolean getFiscalizada() {
    return fiscalizada;
}

public Boolean getAnalisada() {
    return analisada;
}

public List<ItensModel> getItens() {
    return itens;
}

// setters

public void setOs(String os) {
    this.os = os;
}

public void setFiscal(String fiscal) {
    this.fiscal = fiscal;
}

public void setContrato(String contrato) {
    this.contrato = contrato;
}

public void setFiscalizada(Boolean fiscalizada) {
    this.fiscalizada = fiscalizada;
}

public void setAnalisada(Boolean analisada) {
    this.analisada = analisada;
}
   public void setItens(List<ItensModel> itens) {
       this.itens = itens;
   }

} // fim da classe OsModel
```

Fonte: autor.

Figura 58 - Classe ItensModel.java.

```
package br.com.fiscalizacao.model;
public class ItensModel {

    private String mcod_item;
    private String mdescr_item;
    private Double mqtde_item;
    private String munidade_item;
    private Double mpunit_item;
    private Double mptotal_item;

    public ItensModel(String cod_item, String descr_item, Double qtde_item,
             String unidade_item, Double punit_item, Double ptotal_item) {
        mcod_item = cod_item;
        mdescr_item = descr_item;
        mqtde_item = qtde_item;
        munidade_item = unidade_item;
        mpunit_item = punit_item;
        mptotal_item = qtde_item*punit_item;
    }
    public String getCod_item() { return mcod_item; }
    public String getDescr_item() { return mdescr_item; }
    public Double getQtde_item() { return mqtde_item; }
    public String getUnidade_item() { return munidade_item; }
    public Double getPunit_item() { return mpunit_item; }
    public Double getPtotal_item() { return mptotal_item; }
    public void setCod_item(String cod_item) { mcod_item = cod_item; }
    public void setDescr_item(String descr_item) { mdescr_item = descr_item; }
    public void setQtde_item(Double qtde_item) { mqtde_item = qtde_item; }
    public void setUnidade_item(String unidade_item) { munidade_item = unidade_item; }
    public void setPunit_item(Double punit_item) { mpunit_item = punit_item; }
    public void setPtotal_item(Double ptotal_item) { mptotal_item = ptotal_item; }
    public Double changeQtde(Double qtde_item){
        mqtde_item = qtde_item;
        mptotal_item = qtde_item*mpunit_item;
        return mqtde_item;
    }
} // fim da classe ItensModel
```

Fonte: autor.

CRIAÇÃO DA ACTIVITY MAINACTIVITY, LAYOUT ACTIVITY_MAIN.XML, DEFINIÇÃO DO THEMES.XML, DEFINIÇÃO DE STRINGS.XML DO PROJETO.

Primeiramente, definiu-se o tema a ser utilizado no protótipo de aplicativo para Theme.MaterialComponents.DayNight.DarckActionBar no arquivo values\themes\themes.xml.

```
<resources xmlns:tools="http://schemas.android.com/tools">
<!-- Base application theme. -->
 <style name="Theme.Fiscalizacao"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
      <!-- Primary brand color. -->
      <item name="colorPrimary">@color/purple_500</item>
      <item name="colorPrimaryVariant">@color/purple_700</item>
      <item name="colorOnPrimary">@color/white</item>
      <!-- Secondary brand color. -->
```

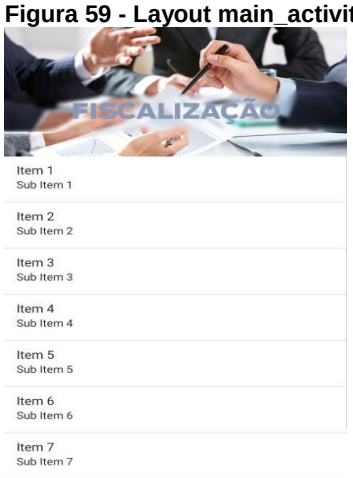
```
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
<!-- Status bar color. -->
        <item name="android:statusBarColor"
tools:targetApi="l">?attr/colorPrimaryVariant</item>
<!-- Customize your theme here. -->
 </style>
 <style name="Theme.Fiscalizacao.ActionBar">
        <item name="windowActionBar">true</item>
        <item name="windowNoTitle">false</item>
    </style>
</resources>
```

Definiu-se os recursos de strings a serem utilizados em todo projeto.

```
<resources>
    <string name="app_name">Fiscalizacao</string>
    <string name="os_activity">Ordem de Serviço</string>
    <string name="image_description">Equipe Fiscalizacao</string>
    <string name="anyataylorjoy_description">Anya Taylor Joy</string>
    <string name="fiscalservico">Fiscal de Serviço</string>
    <string name="os">OS</string>
    <string name="contrato">Contrato</string>
    <string name="analisada">analisada</string>
    <string name="altera_qtde">qtde</string>
    <string name="save_img">icone salvar</string>
    <string name="tot_os">preco total OS</string>
    <string name="cod_item">cod. item</string>
    <string name="descr_item">descricao</string>
    <string name="quant_item">quant.</string>
    <string name="unidade">unidade</string>
    <string name="punit_item">preco unitario</string>
    <string name="ptot_item">preco total</string>
    <string name="dar_conformidade">Finalizar</string>
</resources>
```

Criou-se o layout *activity_main.xml* (vide figura 59).

Figura 59 - Layout main_activity.xml.



Fonte: autor.

Segue abaixo, a codificação do arquivo do Layout *main_activity.xml.*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".activity.MainActivity">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true"
        android:contentDescription="@string/image_description"
        android:cropToPadding="true"
        android:scaleType="fitStart"
        android:src="@drawable/img_equipe_fiscalizacao" />
    <ListView
        android:id="@+id/lv_fiscais"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

Criou-se a activity *MainActivity* e gravou-se no pacote *activity*.

Para melhor organização foi adicionado os pacotes activity, adapter e model dentro do pacote br.com.fiscalizacao.

Segue o código da *MainActivity.java*:

```
package br.com.fiscalizacao.activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;
import java.util.List;
import br.com.fiscalizacao.R;
import br.com.fiscalizacao.model.FiscalModel;

public class MainActivity<Fiscal> extends AppCompatActivity {

 private final FirebaseDatabase database = FirebaseDatabase.getInstance();
 private final DatabaseReference ref = database.getReference();
 private final List<String> listFiscal = new ArrayList<>();
 private ArrayAdapter<String> arrayAdapterfiscal;
 private ListView lv_fiscais;

 @Override
```

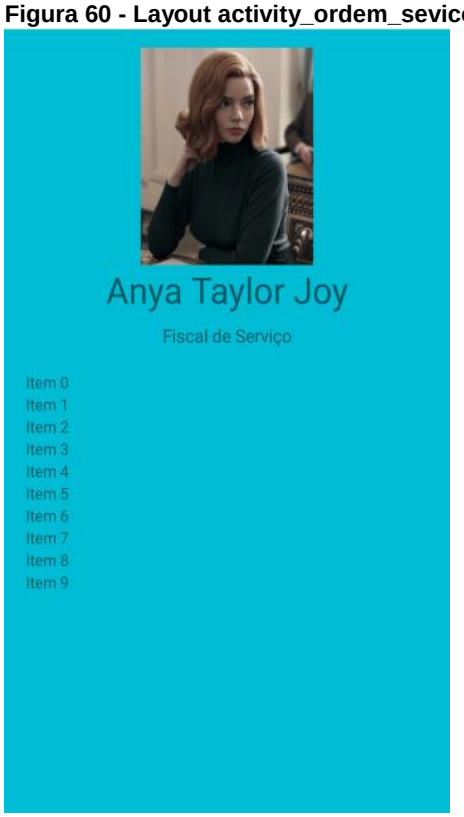
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    lv_fiscais = findViewById(R.id.lv_fiscais);

    aguardaClickListView();

} // fim do método onCreate

@Override
protected void onStart(){
    super.onStart();
    ref.child("fiscal").addListenerForSingleValueEvent(
      new ValueEventListener() {

      @Override
      public void onDataChange(@NonNull DataSnapshot snapshot) {
        listFiscal.clear();
        for(DataSnapshot objSnapshot:snapshot.getChildren()){
          FiscalModel f = objSnapshot.getValue(FiscalModel.class);
          listFiscal.add(f.getNome());
        }
        arrayAdapterfiscal = new ArrayAdapter<>(MainActivity.this,
        android.R.layout.simple_list_item_single_choice,listFiscal);
        lv_fiscais.setAdapter(arrayAdapterfiscal);
      }

      @Override
      public void onCancelled(@NonNull DatabaseError error) {
        }
    });
}

  public void aguardaClickListView(){

    AdapterView.OnItemClickListener itemClickListener =
      new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> lv_fiscais, View itemView,
          int position, long id) {
            Intent intent =
              new Intent(MainActivity.this, OrdensFiscal.class);
              String nomeFiscal =
                lv_fiscais.getItemAtPosition(position).toString();
            intent.putExtra("nome",nomeFiscal );
            startActivity(intent);
        }
    };
    ListView lv_fiscais = findViewById(R.id.lv_fiscais);
    lv_fiscais.setOnItemClickListener(itemClickListener);
}// fim do método aguardaClickListView

} // fim do MainActivity
```

Dentro do método *onCreate* da Activity, informa-se o método *aguardaClickListView* que é responsável por direcionar para activity *OrdensFiscal* após o usuário selecionar o Fiscal desejado.

CRIAÇÃO DA ACTIVITY ORDENSFISCAL.JAVA, LAYOUT ACTIVITY_ORDEM-SERVICO.XML, CARDVIEW_ADAPTER_OS.XML, OSLISTADAPTER

Criou-se o layout *activity_ordem-servico.xml* (vide figura 60)

Figura 60 - Layout activity_ordem_sevico.xml.



Fonte: autor.

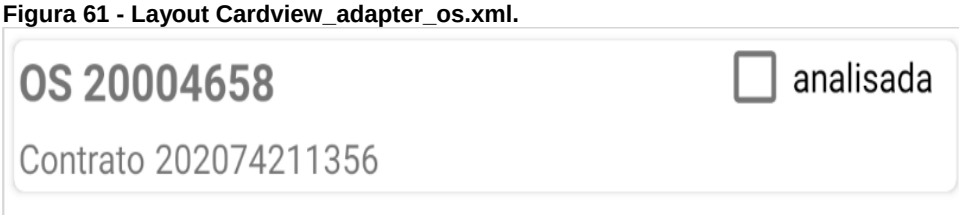
Segue a codificação do Layout *activity_ordem_sevico.xml.*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/fundo_fiscal"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".activity.OrdensFiscal">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="190dp"
        android:background="@color/fundo_fiscal"
        android:contentDescription="@string/anyataylorjoy_description"
        android:cropToPadding="true"
        android:scaleType="fitCenter"
        android:src="@drawable/anyataylorjoy" />

    <TextView
```

```
        android:id="@+id/nomefiscal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/fundo_fiscal"
        android:paddingBottom="8sp"
        android:textAlignment="center"
        android:textSize="32sp" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/fundo_fiscal"
        android:paddingBottom="16sp"
        android:text="@string/fiscalservico"
        android:textAlignment="center"
        android:textSize="16sp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="?android:attr/selectableItemBackground">
        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/osList_recycler"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="4dp"
            android:scrollbars="vertical"/>
    </LinearLayout>
</LinearLayout>
```

Criou-se o cardview_adapter_os.xml (vide figura 61).

Figura 61 - Layout Cardview_adapter_os.xml.



Fonte: autor.

Segue a codificação do Layout *Cardview_adapter_os.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cardviewOs"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="4dp"
    app:cardCornerRadius="4dp"
    app:cardElevation="2dp">

    <RelativeLayout
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/titleos"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="2dp"
            android:layout_marginTop="2dp"
            android:text="@string/os"
            android:textSize="18sp"
            android:textStyle="bold"/>

        <TextView
            android:id="@+id/textOs"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="4dp"
            android:layout_marginTop="2dp"
            android:layout_toEndOf="@+id/titleos"
            android:text="20004658"
            android:textSize="18sp"
            android:textStyle="bold"/>

        <TextView
            android:id="@+id/titleContract"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/titleos"
            android:layout_marginStart="2dp"
            android:layout_marginTop="5dp"
            android:layout_marginBottom="2dp"
            android:text="@string/contrato"
            android:textSize="14sp"/>

        <TextView
            android:id="@+id/textContrato"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/textOs"
            android:layout_marginStart="4dp"
            android:layout_marginTop="5dp"
            android:layout_marginBottom="2dp"
            android:layout_toEndOf="@+id/titleContract"
            android:text="202074211356"
            android:textSize="14sp" />

        <CheckBox
            android:id="@+id/checkAnalisada"
            android:layout_width="wrap_content"
            android:layout_height="25dp"
            android:layout_alignParentEnd="true"
            android:layout_marginEnd="10dp"
            android:layout_marginBottom="10dp"
            android:clickable="false"
            android:text="@string/analisada"
            android:textSize="14sp" />
    </RelativeLayout>
</androidx.cardview.widget.CardView>
```

Criou-se o Adapter *OsListAdapter. Segue a codificação do OsListAdapter*

```
package br.com.fiscalizacao.adapter;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.CheckBox;
import android.widget.TextView;
import androidx.cardview.widget.CardView;
import androidx.recyclerview.widget.RecyclerView;
import org.jetbrains.annotations.NotNull;
import java.util.List;
import br.com.fiscalizacao.R;
import br.com.fiscalizacao.model.OsModel;
public class  OsListAdapter extends
RecyclerView.Adapter<OsListAdapter.OsViewHolder>  {
   private final List<OsModel> listaOs;
   private int position;
   private OnItemClickListener mListener;
   public OsListAdapter(List<OsModel> lista) {
      this.listaOs = lista;
   }

   @NotNull
   public OsViewHolder onCreateViewHolder(ViewGroup parent, int
       viewType)
   {
     CardView cv = (CardView) ayoutInflater
     .from(parent.getContext())
     .inflate(R.layout.cardview_adapter_os, parent,false);

      return new OsViewHolder(cv, mListener);
   }

   public void onBindViewHolder(OsViewHolder holder,
       final int position)
   {
     final CardView cardView = holder.cardView;
      OsModel dadosOs = listaOs.get(position);
      holder.os.setText(dadosOs.getOs());
      holder.contrato.setText(dadosOs.getContrato());
      holder.analisada.setChecked(dadosOs.getAnalisada());
   }

   public int getItemCount() {
      return listaOs.size();
   }

   public interface OnItemClickListener
   { void onItemClick(int position); }

   public void setOnItemClickListener(OnItemClickListener listener)
   { mListener = listener; }

   public static class OsViewHolder extends RecyclerView.ViewHolder
   {
```

```
    private CardView cardView;
    public TextView os;
    public TextView contrato;
    public CheckBox analisada;

     public OsViewHolder(View itemView, OnItemClickListener listener)
     {
        super(itemView);
          os = itemView.findViewById(R.id.textOs);
          contrato = itemView.findViewById(R.id.textContrato);
          analisada = itemView.findViewById(R.id.checkAnalisada);

          itemView.setOnClickListener(new View.OnClickListener()
          {
              @Override
              public void onClick(View v) {
                  if(listener!= null){
                      int position = getAdapterPosition();
                      if(position!= RecyclerView.NO_POSITION){
                          listener.onItemClick(position);
                      }
                  }
              }
          });
      } // fim do construtor OsViewHolder -tem como parâmetro uma view
   } // fim da class OsViewHolder
}
```

Criou-se e codificou-se o activity *OrdensFiscal.java*

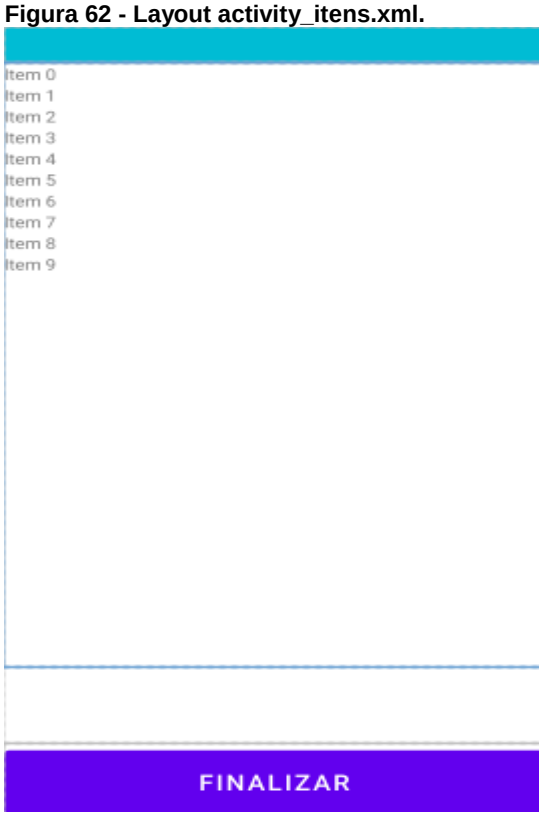
```
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.DividerItemDecoration;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.widget.LinearLayout;
import android.widget.TextView;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.Query;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;
import br.com.fiscalizacao.R;
import br.com.fiscalizacao.adapter.OsListAdapter;
import br.com.fiscalizacao.model.OsModel;

public class OrdensFiscal extends AppCompatActivity  {

    public DatabaseReference ref =
      FirebaseDatabase.getInstance().getReference();
    public static final String NUM_OS = "num_os";
    public static final String TAG = "Fisc";
    public ArrayList<OsModel> listOs = new ArrayList<>();
```

```
    public String fiscal;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ordem_servico);
        mostrarNomeFiscal();
        listarOs(fiscal);

    } // fim do método onCreate

    public void mostrarNomeFiscal(){
        Intent intent = getIntent();
        fiscal = intent.getStringExtra("nome");
        TextView nomeFiscal = findViewById(R.id.nomefiscal);
        nomeFiscal.setText(fiscal);
    } // fim do método mostrar_nome_fiscal

    public void listarOs(String nome){
        Query query1 = ref.child("os").orderByChild("fiscal").equalTo(nome);
        query1.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                for(DataSnapshot objSnapshot:snapshot.getChildren()) {
                    String key = objSnapshot.getKey();
                    OsModel obj = objSnapshot.getValue(OsModel.class);
                    listOs.add(obj);
                    OsListAdapter adapter = new OsListAdapter(listOs);
                    RecyclerView recyclerView =
findViewById(R.id.osList_recycler);
                    RecyclerView.LayoutManager layoutManager = new
LinearLayoutManager(getApplicationContext());
                    recyclerView.setLayoutManager(layoutManager);
                    recyclerView.setHasFixedSize(true);
                    recyclerView.addItemDecoration(new
DividerItemDecoration(getApplicationContext(), LinearLayout.VERTICAL));
                    recyclerView.setAdapter(adapter);
                    adapter.setOnItemClickListener(new
OsListAdapter.OnItemClickListener() {
                        @Override
                        public void onItemClick(int position) {
                            Intent intent = new Intent(getBaseContext(),
ItensOS.class);
                            intent.putExtra("nome", fiscal);
                            intent.putExtra(ItensOS.NUM_OS,
listOs.get(position).getOs());
                            intent.putExtra(ItensOS.KEY, key);
                            startActivity(intent);
                        }
                    });
                }
            } // fim método onDataChange()

            @Override
            public void onCancelled(@NonNull DatabaseError error) {    } // fim
método onCancelled().
        });
    } // fim do método listarOs
} // fim da classe OrdensFiscal
```

CRIAÇÃO DA ACTIVITY ITENSOS.JAVA, LAYOUT ACTIVITY_ITENS.XML, CARDVIEW_ADAPTER_ITENS.XML, ITENSADAPTER.

Criou-se o layout *activity_itens.xml* (vide figura 62)

Figura 62 - Layout activity_itens.xml.



Fonte: autor.

Segue a codificação do Layout *activity_itens.xml.*

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activity.ItensOS"
    android:orientation="vertical">

    <TextView
        android:id="@+id/os_number"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/fundo_fiscal"
        android:paddingBottom="8sp"
        android:textAlignment="center"
        android:textSize="20sp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
```

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/osList_recycler"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_above="@+id/button_conforme"
    android:layout_below="@+id/os_number"
    android:scrollbars="vertical"
    android:layout_marginBottom="70dp"
    app:layout_constraintBottom_toTopOf="@id/button_conforme"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/os_number" />

<Button
    android:id="@+id/button_conforme"
    android:layout_width="match_parent"
    android:layout_height="70dp"
    android:text="@string/dar_conformidade"
    android:textAlignment="center"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/osList_recycler" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Criou-se o cardview_adapter_itens.xml (vide figura 63)

Figura 63 - Cardview_adapter_itens.xml.

Fonte: autor.

Segue abaixo a codificação do Layout Cardview_adapter_itens.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_column="100">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:padding="2sp">

        <TextView
            android:id="@+id/cod_item"
            android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_marginStart="10sp"
    android:layout_marginTop="2dp"
    android:gravity="center"
    android:text="@string/cod_item"
    android:textSize="20sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/descr_item"
    android:layout_width="300sp"
    android:layout_height="wrap_content"
    android:layout_marginStart="4sp"
    android:layout_marginTop="2sp"
    android:gravity="start|center"
    android:text="@string/descr_item"
    android:textSize="20sp"
    android:textStyle="bold"
    app:layout_constraintStart_toEndOf="@+id/cod_item"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/tit_qtde"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="10sp"
    android:layout_marginTop="2sp"
    android:gravity="center"
    android:text="@string/quant_item"
    android:textSize="12sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/descr_item" />

<TextView
    android:id="@+id/tit_unidade"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="4sp"
    android:layout_marginTop="2sp"
    android:gravity="center"
    android:text="@string/unidade"
    android:textSize="12sp"
    app:layout_constraintStart_toEndOf="@id/tit_qtde"
    app:layout_constraintTop_toBottomOf="@+id/descr_item" />

<TextView
    android:id="@+id/tit_pr_unit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="4sp"
    android:layout_marginTop="2sp"
    android:gravity="center"
    android:text="@string/punit_item"
    android:textSize="12sp"
    app:layout_constraintStart_toEndOf="@id/tit_unidade"
    app:layout_constraintTop_toBottomOf="@+id/descr_item"/>

<TextView
```

```
    android:id="@+id/tit_pr_total"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="4sp"
    android:layout_marginTop="2sp"
    android:gravity="center"
    android:text="@string/ptot_item"
    android:textSize="12sp"
    app:layout_constraintStart_toEndOf="@id/tit_pr_unit"
    app:layout_constraintTop_toBottomOf="@+id/descr_item"/>

<EditText
    android:id="@+id/qtde_item"
    android:layout_width="wrap_content"
    android:layout_height="40sp"
    android:layout_marginStart="10sp"
    android:layout_marginTop="2sp"
    android:gravity="center"
    android:imeOptions="actionDone"
    android:inputType="numberDecimal"
    android:padding="4sp"
    android:textSize="15sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tit_qtde" />

<TextView
    android:id="@+id/unidade_item"
    android:layout_width="wrap_content"
    android:layout_height="30sp"
    android:layout_marginStart="12sp"
    android:layout_marginTop="4sp"
    android:gravity="center"
    android:textSize="12sp"
    app:layout_constraintStart_toStartOf="@id/tit_unidade"
    app:layout_constraintBottom_toBottomOf="@id/qtde_item" />

<TextView
    android:id="@+id/punit_item"
    android:layout_width="wrap_content"
    android:layout_height="30sp"
    android:layout_marginStart="16sp"
    android:layout_marginTop="4sp"
    android:gravity="center"
    android:textSize="12sp"
    app:layout_constraintStart_toStartOf="@id/tit_pr_unit"
    app:layout_constraintBottom_toBottomOf="@id/qtde_item"/>

<TextView
    android:id="@+id/ptotal_item"
    android:layout_width="wrap_content"
    android:layout_height="30sp"
    android:layout_marginStart="10sp"
    android:gravity="center"
    android:textSize="12sp"
    app:layout_constraintBottom_toBottomOf="@id/qtde_item"
    app:layout_constraintStart_toStartOf="@id/tit_pr_total" />
<ImageView
    android:id="@+id/image_save"
    android:layout_width="45dp"
    android:layout_height="45dp"
```

```
            android:layout_marginStart="4sp"
            android:layout_marginTop="4sp"
             android:layout_marginBottom="3sp"
             android:src="@drawable/ic_save"
             android:contentDescription="@string/save_img"
            app:layout_constraintBottom_toBottomOf="@id/qtde_item"
             app:layout_constraintEnd_toEndOf="parent"/>

    </androidx.constraintlayout.widget.ConstraintLayout>

</androidx.cardview.widget.CardView>
```

Criou-se o Adapter *ItensAdapter.*

```
package br.com.fiscalizacao.adapter;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
import br.com.fiscalizacao.R;
import br.com.fiscalizacao.model.ItensModel;

public class ItensAdapter extends
RecyclerView.Adapter<ItensAdapter.OsDetailViewHolder> {
    private final List<ItensModel> mItensList;
    private OnItemClickListener mListener;
    public interface OnItemClickListener{
        void onClick(int position);
        void onSaveClick(int position, double qtde_item);
    }

    public void setOnItemClickListener(ItensAdapter.OnItemClickListener
listener){
        mListener = listener;
    }

    public ItensAdapter(List<ItensModel> detailsList){
        mItensList = detailsList;
    }

    public static class OsDetailViewHolder extends RecyclerView.ViewHolder{
    public TextView mcod_item;
    public TextView mdescr_item;
    public EditText mqtde_item;
    public TextView munidade;
    public TextView mpunit_item;
    public TextView mptot_item;
    public ImageView msave_item;

        public OsDetailViewHolder(View itemView, final OnItemClickListener
listener){
            super(itemView);
            mcod_item = itemView.findViewById(R.id.cod_item);
            mdescr_item = itemView.findViewById(R.id.descr_item);
            mqtde_item = itemView.findViewById(R.id.qtde_item);
            munidade = itemView.findViewById(R.id.unidade_item);
```

```
            mpunit_item = itemView.findViewById(R.id.punit_item);
            mptot_item = itemView.findViewById(R.id.ptotal_item);
            msave_item = itemView.findViewById(R.id.image_save);

            msave_item.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    if(listener != null){
                        double qtde_item =
Double.parseDouble(mqtde_item.getText().toString());
                        int position = getAdapterPosition();
                        if(position != RecyclerView.NO_POSITION){
                            listener.onSaveClick(position, qtde_item);
                        }
                    }
                }
            });

        } // fim do construtor da classe OsDetailViewHolder
    } // fim da class OsDetailViewHolder

    @NonNull
    @Override
    public OsDetailViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
        View v =
LayoutInflater.from(parent.getContext()).inflate(R.layout.cardview_adapter_it
ens,parent,false);
        OsDetailViewHolder detailViewHolder = new OsDetailViewHolder(v,
mListener);
        return detailViewHolder;
    }

    @Override
    public void onBindViewHolder(@NonNull OsDetailViewHolder holder, int
position) {
        ItensModel currentItem = mItensList.get(position);
        holder.mcod_item.setText(currentItem.getCod_item());
        holder.mdescr_item.setText(currentItem.getDescr_item());
        holder.mqtde_item.setText(String.valueOf(currentItem.getQtde_item()));
        holder.munidade.setText(currentItem.getUnidade_item());

holder.mpunit_item.setText(String.valueOf(currentItem.getPunit_item()));

holder.mptot_item.setText(String.valueOf(currentItem.getPtotal_item()));
    }

    @Override
    public int getItemCount() {
        return mItensList.size();
    }
} // fim da class ItensAdapter
```

INCLUSÃO DO BUTTON UP

Bastou especificar a atividade-pai usando o atributo *android:parentActivityName* em cada activity presente no *AndroidManifest.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
package="br.com.fiscalizacao">

<uses-permission android:name="android.permission.INTERNET" />

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.Fiscalizacao">
    <activity
        android:name=".activity.ItensOS"
        android:label="Itens da Ordem de Serviço"
        android:parentActivityName=".activity.OrdensFiscal">

    </activity>

    <activity
        android:name=".activity.OrdensFiscal"
        android:label="Lista de Ordens de Serviços"
        android:parentActivityName=".activity.MainActivity"/>
    <activity
        android:name=".activity.MainActivity"
        android:label="@string/app_name"
        android:theme="@style/Theme.Fiscalizacao.ActionBar">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>
```

CRIAÇÃO DE ÍNDICES NO REALTIMEDATABASE

Basta realizar a seguinte configuração via console do Firebase na aba regras do projeto TCC-fiscalizacao. Vide figura 64.

Figura 64 - Criação de índices para o nós os, fiscal e itens.

```
{
  "rules": {
    ".read": true,
    ".write": true,
      "os" : {
        ".indexOn" : ["fiscal","itens","os"]
      }
  }
}
```

Fonte: FIREBASE, 2020a.