UNIVERSIDADE DO ESTADO DE SANTA CATARINA
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT

Alan Hoffmann
Aliffe Bezerra Kauling
Ramon Herrero Martins
Tainara dos Santos

ESTUDO DAS APIS E RECURSOS UTILIZADOS NO DESENVOLVIMENTO DE UMA APLICAÇÃO PARA SISTEMA OPERACIONAL ANDROID 6 BASEANDO-SE NO ESTUDO DE CASO DO APLICATIVO CORONA WARN.

JOINVILLE, SC
2020

Trabalho apresentado como um requisito parcial para obter a aprovação na disciplina de Sistemas Operacionais.

Orientador: Charles Christian Miers

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS E QUADROS

SUMÁRIO

1. INTRODUÇÃO

A quantidade de dispositivos móveis hoje funcionando e conectados na Internet é grande. Segundo Butcher (2020), a internet tem se tornado um meio de comunicação e troca de informações, e atrelado aos dispositivos móveis, os usuários podem acessá-las de qualquer lugar com rede de WiFi ou internet móvel 3G ou 4G.

Um dos sistemas operacionais usados nesses dispositivos móveis é o Android, que segundo Zuriarrain (2017) é um dos SO mais usados na atualidade. É uma plataforma considerada aberta aos fabricantes, o que a torna ainda mais atrativa para desenvolvedores. Além disso, o Android não é restrito a smartphones, mas também pode ser usado em dispositivos como tablets, netbooks e relógios.

Como parte da disciplina de Sistemas Operacionais e visto o contexto citado, este trabalho tem por objetivo um estudo de caso da arquitetura e API utilizada no desenvolvimento e funcionamento do aplicativo para sistema operacional Android 6 "Corona Warn". Com base no objetivo definido, um estudo científico é realizado a fim de identificar qual API é utilizada pelo aplicativo “Corona Warn”. No estudo deve ser observado pontos da arquitetura Android, as classes utilizadas no desenvolvimento do aplicativo e a API de exposição.

No capítulo 2, é descrito sobre o sistema operacional Android, sua história, propriedades, a arquitetura presente no sistema, os componentes e os funcionamentos básicos do sistema e as suas APIs e SDKs . Também é apresentado o aplicativo Corona Warn, introduzido seu funcionamento e recursos necessários para uso dele. Neste capítulo são abordadas as definições do sistema Android e introdução ao sistema Corona Warn, visando a compreensão dos capítulos seguintes.

O capítulo 3 é crucial pois, entra-se mais a fundo no aplicativo Corona Warn. É explicado sobre a API de notificação de exposição desenvolvida pela Google e Apple e outras bibliotecas utilizadas, é abordado a arquitetura descentralizada do aplicativo e também explicado sobre a tecnologia do

Bluetooth de baixa energia, que permite a comunicação e troca de informação entre os aparelhos.

Por último, no capítulo 4 é realizado um estudo de caso onde é descrito como é utilizada a API de notificação de exposição no aplicativo. Neste capítulo, é tenta-se ao máximo mostrar na prática como é implementada a API de notificação de exposição e as principais tecnologias usadas no aplicativo, para que cada etapa das tarefas do sistema seja realizada.

A metodologia utilizada foi realizada com base em consultas em sítios online, selecionando assim fontes necessárias. O estudo de caso ocorreu através do estudo da documentação e da análise de partes do código do aplicativo Corona Warn, ambos fornecidos pelo próprio consorcio de empresas e demais envolvidos no projeto Corona Warn, na plataforma GitHub.

2. CONCEITOS

O sistema operacional Android foi criado em 2003 e é atualmente o sistema operacional mais usado no mundo, segundo Zuriarrain (2017). A plataforma é usada principalmente em aparelhos celulares, sendo que existem quantidades consideráveis de APIs e bibliotecas desenvolvidas para aplicativos *mobile*.

2.1. SISTEMA OPERACIONAL ANDROID

Segundo Fernandes et al. (2012), o sistema operacional Android pode ser definido como uma máquina virtual Java que é baseada no Linux. Ou seja, é uma plataforma desenvolvida em Java que opera no sistema Linux.

De acordo com a Android Inc. (2020), ao fazer uso de um núcleo do Linux, o Android aproveita os principais recursos de segurança, além de os fabricantes dos dispositivos poderem desenvolver *drivers* de *hardware* para o núcleo que é conhecido. Ainda para Fernandes et al. (2012), embora tenha sido desenvolvido para *smartphones,* também é utilizado em aplicações como *tablets*, *netbooks* e até mesmo relógios. Na visão de Pereira e Simões (2014), o Android tornou-se atrativo para a Google justamente porque tem plataforma

aberta aos fabricantes. De acordo com Kleina (2014), Android é a plataforma móvel mais usada no mundo, e segundo Zuriarrain (2017) tornou-se em 2017 o sistema operacional mais usado no mundo. Isso se deve principalmente por sua capacidade de funcionar em diversos dispositivos. Na afirmação de Lecheta (LECHETA 2015 apud HUBSCH, 2012), a plataforma possui diversos componentes, grande disponibilidade de bibliotecas e interface gráfica e conta com diversas ferramentas para o desenvolvimento dos aplicativos, inclusive da própria Google, o Android Studio. Por conta disso, o uso do sistema Android aumenta a cada ano, tanto pela versatilidade para aparelhos diferentes quanto pelo acesso fácil aos desenvolvedores, disponibilizando bibliotecas, interface e demais ferramentas, de forma a possibilitar cada vez mais o desenvolvimento de APIs e aplicações para o sistema.

2.2. HISTÓRICO

Segundo Meyer (2015), a Android Inc., uma iniciativa de criar uma plataforma aberta para *smartphones*, surgiu em 2003 pelas mãos de Andy Rubin, Rich Miner, Nick Sears e Chris White. O lançamento da primeira versão estável surgiu em 2008 e hoje se encontra na versão 10. Inicialmente a ideia era lançar um sistema para câmeras digitais, mas com o declínio do mercado, focaram no mercado mobile que estava crescendo com o surgimento do Apple iPhone em 2007. Em 2005 a Google adquiriu a Android Inc. e foi criada então a *Google Mobile Division*. Ainda segundo Meyer (2015), a criação do Android deu um passo grande quando em 2007 fabricantes como Samsung, Sony, HTC, operadoras como as americanas Sprint Nextel e T-Mobile, e fabricantes de hardware como Qualcomm e a Texas Instruments, além do próprio Google, reuniram-se em um consórcio de tecnologia e fundaram a *Open Handset Alliance*. O objetivo da união de marcas era a criação de uma plataforma de código aberto para *smartphones*. O resultado foi o primeiro Android comercial do mercado, rodando em um HTC Dream, lançado oficialmente em 22 de outubro de 2008.

Cada versão do sistema Android está relacionada a uma API (Interface de Programação de Aplicações) que implementa um conjunto de rotina e

funcionalidades de forma incremental, eventualmente substituindo algumas funcionalidades conforme consulta à ANDROID DEVELOPERS (2020). Para um melhor entendimento das melhorias implementadas em cada uma das APIs, deve-se consultar o Anexo I.

2.3. PROPRIEDADES DA VERSÃO 8

Segundo o ANDROID DEVELOPERS (2020), as funcionalidades do Android são disponibilizadas desde a versão 1.0 com incremento de funcionalidades. Portanto além das funcionalidades implementadas até a versão 7.0 (API 25), a versão 8.0 apresenta, segundo os critérios relevantes ao projeto, os seguintes elementos:

- Sistema de Notificações ao usuário;
- Possibilidade de uso em sistemas *touchscreen*, com teclado virtual que aceita palavras e dicionários customizados pelo usuário;
- Suporte a *widgets*;
- Rotação automática;
- Motor de fala que permitia aos apps falarem uma sequência de texto;
- *Bluetooth*;
- Suporte ao *Android Cloud Computing*;
- Suporte à tecnologia NFC (*Near Field Communication* – Comunicação de pequenos pacotes sem contato);
- Suporte nativo para sensores, como giroscópio e barômetro;
- Capacidade de criptografar todos os dados do usuário;
- *Android Beam*, recurso de comunicação de baixo alcance que permite a troca rápida de favoritos, informações de contato, direções, vídeos do YouTube e outros dados;
- API de redes neurais (sistema aprende com o seu uso diário).

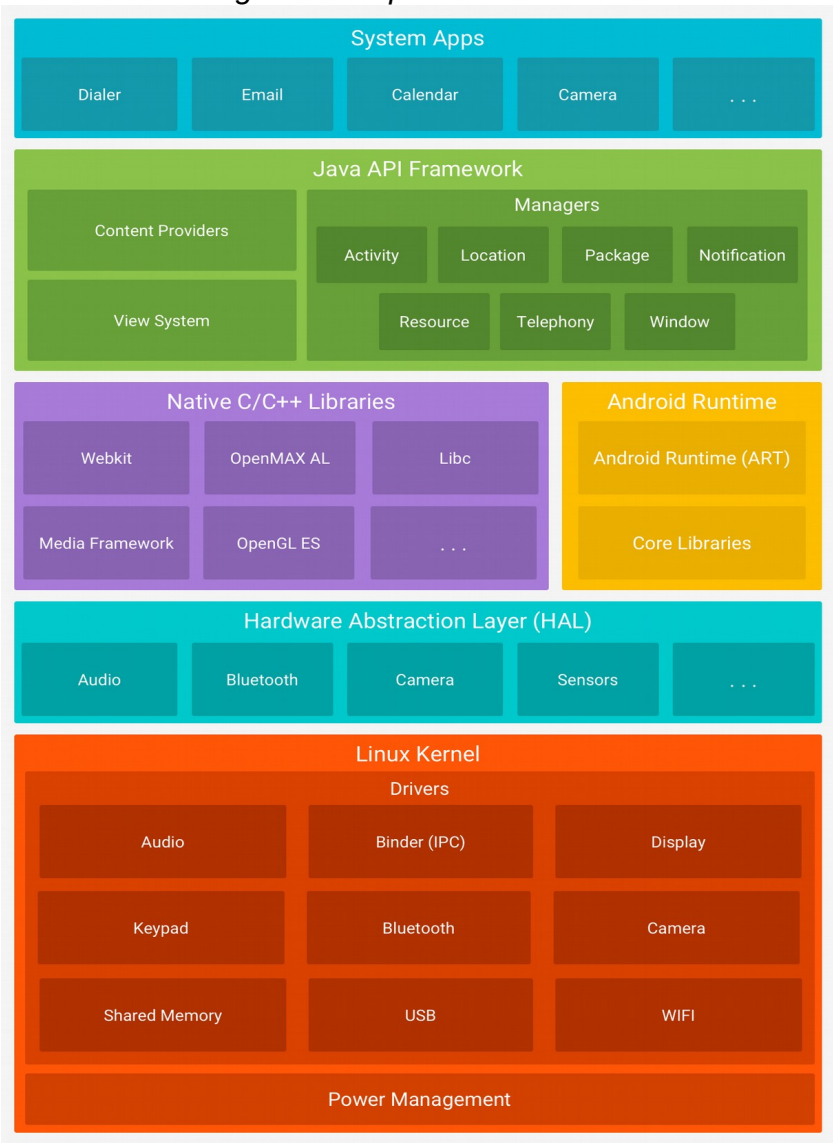
A API usada no trabalho encontra-se na versão 25, portanto, várias melhorias foram incrementadas desde o lançamento do sistema e as funcionalidades são apresentadas com relação à versão anterior. O que são apresentadas aqui são as funcionalidades vigentes desde a API 1 até a API 26

e que são relevantes ao projeto. Demais características podem ser vistas no Anexo I.

2.4. ARQUITETURA DO ANDROID

De acordo com Lecheta (2015), a arquitetura do sistema operacional Android é uma pilha de programas que são agrupados em camadas. Estas camadas podem ser divididas em níveis: zero, um, dois e três, de acordo com a Figura 1.

Figura 1: Arquitetura Android



Fonte: Arquitetura Android (2020)

Na Figura 1 pode-se ver a divisão em níveis da arquitetura Android, sendo a base desenvolvida em núcleo Linux. Logo acima está a camada *Hardware Abstraction Layer,* seguida pelas as camadas de bibliotecas e tempo de execução, que estão as três no mesmo nível. Em seguida está a *Java API Framework*, e por último, no topo da imagem, os aplicativos de sistema. O nível zero é a base da pilha, desenvolvido no sistema operacional Linux, e contém os programas de gerenciamento de memória, configurações de segurança e *drivers* de hardware.

Conforme Gowrishanakar (2015) o Android possui um núcleo Linux específico para Android que fornece gerenciamento de energia, compartilhamento de memória e gerenciamento, isto para conter as menores restrições de memória entre os dispositivos móveis. A comunicação entre processos permite que os processos se comuniquem entre si, logo, os *drivers* do dispositivo podem se conectar com dispositivos de *hardware*, como memória, rádio, câmera entre outros.

Na visão de Lecheta (2015), é no nível 1 que estão as camadas de bibliotecas e tempo de execução, chamado de *Android Runtime*. A camada de biblioteca é formada por instruções que servem para informar o dispositivo sobre como lidar com diferentes tipos de dados usados por componentes do sistema que são expostos aos desenvolvedores através da estrutura do sistema Android.

Já a camada de tempo de execução (*Android Runtime*), ainda segundo Lecheta (2015), inclui um conjunto de bibliotecas Java chamada *Core Java Libraries*. É esta camada que contém a Máquina Virtual Dalvik (DVM) que é usada pelo Android para executar cada aplicação com seu próprio processo, de forma a permitir que múltiplas instâncias executem simultaneamente. Ou seja, caso uma aplicação pare, isso não afeta as demais aplicações em execução. Sendo assim, o gerenciamento de memória é simplificado.

Essa forma de execução das DVMs é importante, segundo Gowrishanakar (2015), porque diferente dos computadores, os dispositivos móveis têm CPU mais lenta, menos memória RAM e tamanho limitado.

De acordo com Gowrishanakar (2015), o Android possui também sua própria biblioteca C que manipula processos sensíveis ao desempenho do núcleo, como por exemplo "renderizar" imagens ou então atualizar as páginas. Essa biblioteca C também é chamada *BIONIC LIBC*, que implementa chamadas padrão do sistema, ou seja, criação de processos e *threads*, cálculos matemáticos e alocação de memória, por exemplo.

De acordo com Lecheta (2015), no nível 2, existe a camada de *framework* de aplicação (***Application Framework***), programas que fazem o gerenciamento das aplicações básicas do aparelho. Os desenvolvedores têm acesso total ao *framework*, como um conjunto de ferramentas básicas que são usadas para desenvolver ferramentas mais complexas. Segundo Gowrishanakar (2015), este nível tem os componentes de software reutilizáveis, que são:

- ***View System:*** Contém itens gráficos que são incluídos na interface do usuário, como botões, por exemplo.
- ***Package Manager:*** São os pacotes que estão atualmente instalados no dispositivo.
- ***Window Manager:*** Gerencia as janelas que compõem o aplicativo.
- ***Resource Manager:*** Gerencia os recursos não compilados, como por exemplo *strings*, gráficos e arquivos de *layout*.
- ***Activity Manager:*** Corresponde a tela de interface de usuário.
- ***Content Providers:*** São os bancos de dados que permitem o aplicativo armazenar e compartilhar as informações estruturadas.
- ***Location Manager:*** Permite que o aplicativo acesse as informações de localização e informações de movimento como sistemas GPS.
- ***Notification Manager:*** Permite os ícones de notificação na barra superior do aparelho quando ocorrem eventos como o recebimento de uma mensagem, por exemplo.

Portanto, neste nível, é feito o uso de componentes reutilizáveis, que são usados para a criação de aplicativos. Também é realizada a gerência dos aplicativos bases que integram o aparelho.

Para Lecheta (2015), o nível 3 é composto pela camada de aplicações e as funções básicas do dispositivo. É a camada de interação entre o usuário e o dispositivo móvel, na qual estão os aplicativos, por exemplo, e-mail, mensagens SMS, calendário, mapas, navegador e contatos.

Logo, o Android é separado em três níveis, que por sua vez possuem camadas. Com a junção dessas camadas, desde a base formada pelo núcleo do Linux até os aplicativos básicos do aparelho, é formada a arquitetura do sistema Android.

2.5. DESENVOLVIMENTO DE APLICAÇÕES ANDROID

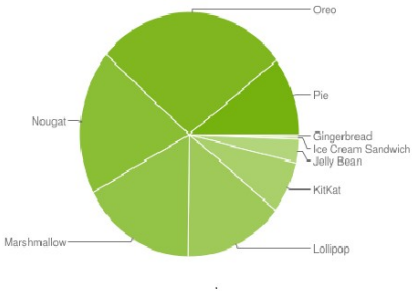
Visto como é estruturado o sistema operacional e como os recursos são dispostos, é apresentado agora a forma de implementar estes recursos para o desenvolvimento de uma solução bom base no sistema operacional Android.

2.5.1. Estruturas de APIs

A API (Interface de Programação de Aplicativos) é um conjunto de funções e rotinas que são disponibilizadas para que outros programas que a utilizem não façam as suas implementações. A cada nova versão do Android, uma nova API é lançada. Como mostra a Figura 2, o Android está na versão 9 e sua API nível 28, mas o sistema permite especificar quais versões serão suportadas através de um arquivo de configurações onde é definido um nível mínimo da API, permitindo às aplicações o uso de versões de bibliotecas mais antigas. Segundo Meyer (2020), o Android fornece em suas APIs um conjunto de pacotes e classes, elementos XML para acessar recursos do sistema, um conjunto de permissões que podem ser solicitadas pelos programas e um pacote de *intents*.

Figura 2: Utilização das versões Android

2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%



Fonte: Android Developer's (2020

Devido então à proporção de dispositivos que usam determinada versão, optou-se por focar na API 26, correspondente à versão 8.0 (Oreo), o que abrange boa parte dos dispositivos atuais e possibilita seu uso e suporte por um tempo aceitável em termos de projeto.

2.5.2. Ambientes de desenvolvimento SDK

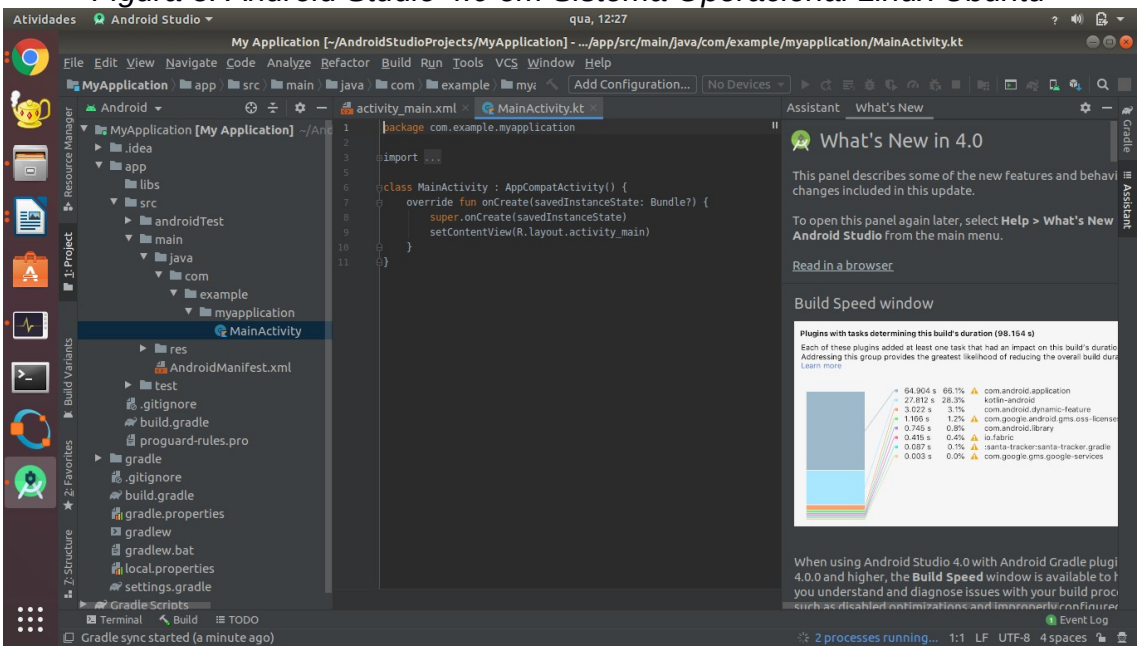
Segundo DUTSON (2013) a SDK (Kit de Desenvolvimento de Software) Android é um conjunto de ferramentas usadas para desenvolver aplicações para a plataforma Android. O SDK Android contém:

- *Debugger*;
- Emulador do sistema Android em várias versões;
- Exemplos de código-fonte;
- Tutoriais;
- Documentação das APIs Android; e
- Bibliotecas de código padrão para cada API.

Com base na documentação do Android (2020), quando o SDK é utilizado, os códigos devem ser escritos na linguagem Java e são executados na máquina virtual Dalvik, que é executada sobre um núcleo Linux. Também está disponível a SDK para códigos nativos C/C++, chamada de NDK (Kit de

Desenvolvimento Nativo), que permite acessar funções de baixo nível da arquitetura Android. Vale ressaltar que existem vários ambientes de desenvolvimento e a escolha natural é pelo SDK do próprio Google, o Android Studio que está disponível para Windows, Linux e iOS e pode ser visto na Figura 3.

Figura 3: Android Studio 4.0 em Sistema Operacional Linux Ubuntu



Fonte: Autores (2020)

Como qualquer software, o Android Studio apresenta vantagens e desvantagens frente aos concorrentes. Como principal vantagem é uma melhor integração e atualização das APIs e como desvantagem pode ser citada a necessidade de desenvolvimento de aplicações para iOS em um outro ambiente, sendo que em outros SDKs é possível fazer o desenvolvimento de aplicações que sirvam para ambos sistemas operacionais.

2.6. CORONA WARN

Segundo WOLF(2020), o aplicativo é um projeto de código aberto, que ajuda a rastrear as cadeias de infecção de SARS-CoV-2, que pode causar COVID-19, na Alemanha. O aplicativo (disponível para Android e IOS), é

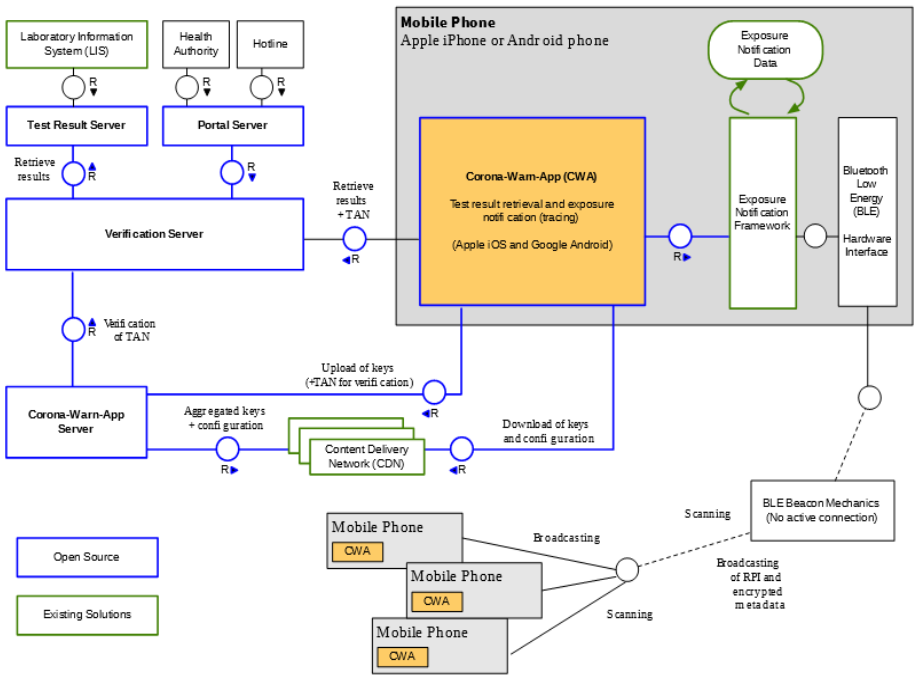
baseado em tecnologias com uma abordagem descentralizada e notifica os usuários se eles foram expostos ao COVID-19.

Segundo a Deutsche Welle (2020), emissora internacional da Alemanha, o aplicativo Corona Warn é um recurso que notifica se usuário esteve próximo a infectados nas últimas duas semanas. Porém, por questão de privacidade e normas do governo alemão, o sistema não registra localização do celular, mas atua a partir de trocas de dados via Bluetooth, que permite aos dispositivos trocarem dados a curta distância. No capítulo 3, é feito o estudo detalhado do aplicativo e de todos os recursos e funções disponíveis e como é preservada a confidencialidade dos usuários.

2.6.1. Funcionamento do aplicativo

Segundo Kowark (2020), o aplicativo Corona Warn funciona em quatro etapas, sendo elas: coletar e distribuir informações de e para outros aparelhos próximos, através de identificadores; comunicar o resultado do teste de usuários; distribuir a lista de chaves; e verificar a exposição a usuários confirmados com COVID-19. A Figura 4 demonstra como isto ocorre.

Figura 4: Arquitetura de troca de dados



Fonte: Kowark (2020)

De acordo com Sarkar (2020), a aplicação Corona Warn utiliza da estrutura de notificação de exposição criada e disponibilizada pela Apple e Google. Essa API faz com que um dispositivo móvel transmita um identificador de proximidade rotativa chamado *Rolling Proximity Identifier* (RPI) e, ao mesmo tempo, permite que o telefone móvel procure identificadores de outros telefones móveis, usando a tecnologia *Bluetooth Low Energy*. Os identificadores são válidos por 10 a 20 minutos e são derivados de outra chave de Exposição Temporária (TEK) que é substituída à meia-noite todos os dias por meio de criptografia. Após serem coletados de outros dispositivos, estes identificadores são armazenados localmente no dispositivo.

Segundo Kowark (2020), ao realizar o teste e ser positivado para SARS-CoV-2, caso o laboratório suporte o processo eletrônico do Corona Warn, este laboratório irá gerar um folheto informativo com um código QR personalizado. O código é criado no local ou já está disponível como uma pilha de códigos QR pré-impressos. O código QR contém um identificador globalmente exclusivo (GUID). De posse de seu de seu código QR de notificação de resultado positivo para SARS-CoV-2, os usuários podem enviar voluntariamente suas chaves temporárias dos últimos 14 dias para o servidor da aplicação. Caso o usuário compartilhe a informação, lendo o código gerado anteriormente, uma chamada de serviço da web (REST) é colocada no servidor de verificação que adiciona as chaves do usuário à lista de confirmados com SARS-CoV-2, a qual é regularmente transmitida para todos os dispositivos com o aplicativo Corona Warn.

Conforme Wolf (2020), a aplicação, periodicamente, realiza *download* da lista de todas chaves disponíveis de usuários com teste positivo para o dispositivo. De posse da lista de chaves, API de notificação de exposição verifica se algum dos identificadores armazenada localmente corresponde a algum dos identificadores de Proximidade Rotativa da lista obtida. Em caso de exposição, é então calculado o risco de exposição do usuário. Tudo isto obedecendo os critérios de confidencialidade das informações requeridas pelo governo alemão e expostas por Vaudenay (2020).

2.6.2. Recursos necessários para instalação e uso

Segundo a documentação do Corona Warn (2020), é necessário um dispositivo com Android 6.0 ou superior que possua suporte ao *Bluetooth Low Energy* (*Bluetooth* 4.0 em diante). A API de notificação de exposição é integrada ao *Google Play Services*, portanto apenas essa aplicação precisa ser atualizada.

2.7. CONSIDERAÇÕES PARCIAIS

O sistema operacional Android, é desenvolvido com base em um núcleo Linux, sendo sua arquitetura dividida em camadas de zero a três. Os aplicativos desenvolvidos para o sistema operacional Android, em sua maioria, são compilados pelas ferramentas do Android SDK (exemplo, Android Studio). As APIs, um conjunto de funções e rotinas fornecidas pelo sistema Android, podem ser usadas para facilitar o desenvolvimento das aplicações que necessitam acessar as funções mais primitivas do sistema.

A aplicação Corona Warn, com seus aspectos focada para o sistema operacional Android com base na API de notificação de exposição foi criada e disponibilizada pela Apple e Google, com o objetivo de auxiliar no combate à proliferação do contágio do COVID-19. Interessante ressaltar que os recursos empregados no desenvolvimento desta aplicação também podem ser utilizadas em aplicações futuras caso surja uma necessidade semelhante como uma nova pandemia. Tudo isto devido aos critérios de desenvolvimento em código aberto e que foram disponibilizados pela empresas envolvidas.

3. TECNOLOGIAS RELACIONADAS

A partir do final do ano de 2019 e início de 2020, o mundo foi surpreendido com a pandemia do novo COVID-19. O número de mortes, até o presente momento desse estudo, segundo a Organização Mundial da Saúde (WHO, 2020), passa dos 980 mil óbitos. Tendo em vista esse cenário

alarmante, várias empresas de tecnologia se mobilizaram para desenvolver recursos que possam auxiliar a atenuar as taxas de contágio das pessoas. As empresas Google e Apple lançaram em conjunto a API de notificação de exposição, possibilitando assim que diversas empresas pudessem desenvolver aplicações que troquem informações entre dispositivos, possibilitando saber se certa pessoa em algum momento nos últimos dias teve contato com alguém infectado e notificar essa pessoa sobre seu risco de contágio.

3.1. API DE NOTIFICAÇÃO DE EXPOSIÇÃO

De acordo com a documentação fornecida pelo Android Developer's (2020), a API de notificações de exposição é um esforço conjunto entre a Apple e o Google para fornecer uma funcionalidade para a construção de aplicativos Google Android e Apple iOS para notificar os usuários sobre a possível exposição a casos COVID-19 confirmados. A API está na versão 1.5 e tem por objetivo auxiliar os governos e a comunidade global no combate e redução do contágio da COVID-19. Os aplicativos Android que fazem uso desta API, notificam os usuários sobre uma possível exposição a casos COVID-19 confirmados.

De acordo com WOLF(2020), o sistema de notificações de exposição, quando ativo, possibilita que o dispositivo gere códigos aleatórios, que mudam entre 10 e 20 minutos, e são enviados por Bluetooth para dispositivos próximos, ao mesmo tempo que recebe os códigos dos dispositivos próximos. Periodicamente os códigos são comparados com uma lista de códigos de casos positivos de COVID-19, salva no dispositivo. Caso haja correspondência a aplicação notificará. O sistema de notificações de exposição faz uso do Bluetooth e internet do dispositivo. As funcionalidades do Bluetooth, por sua vez, ocorrem nos serviços do Google Play.

Segundo o consórcio Bluetooth (BLUETOOTH, 2020), através desta interface são ativados os *beacons* de transmissão (para saber mais sobre *beacons*, ver a seção 3.3), busca por outros *beacons*, verifica através da

comparação de chaves se o usuário teve contato com pessoas infectadas, calcula o nível de risco de exposição do usuário.

Os serviços Google Play também são responsáveis pelo gerenciamento das chaves diárias, dentre as tarefas estão: gerar as chaves de exposição temporárias aleatórias diárias e identificadores de proximidade rotativa e fornecer as chaves de certo intervalo de tempo para usuários diagnosticados com COVID-19, segundo Google (2020). Os serviços Google Play apresentam os termos de consentimento aos usuários, seja logo no início ao ativar o sistema ou antes de compartilhar suas chaves para *upload*, caso o usuário tenha sido diagnosticado com COVID-19.

Os métodos e especificações da API estão previamente documentados. Os métodos da API de notificação de exposição (*Exposure Notification* API) estão dispostos na Quadro 1.

Quadro 1: Métodos e descrições da API de Notificação de Exposição

Método	**Descrição**
start()	Diz ao Google Play Services para iniciar o processo de transmissão e verificação. Na primeira vez que esse método é chamado após a instalação do aplicativo ou sempre que stop()é chamado, ele solicita que o Google Play Services exiba uma caixa de diálogo, onde o usuário é solicitado a dar permissão para transmitir e digitalizar.
isEnabled()	Indica se as notificações de exposição estão habilitadas no dispositivo. Observe que isso não verifica se as configurações de Bluetooth ou localização estão ativadas.
stop()	Desativa o processo de transmissão e verificação. Você pode ligar para isso diretamente. Também é chamado quando um usuário desinstala o aplicativo. Quando é chamado como parte do processo de desinstalação, o banco de dados e as chaves são excluídos do dispositivo.
deviceSupportsLocationlessScanning()	Indica se o dispositivo oferece suporte para digitalização sem localização.
getVersion()	Retorna um LongInt que representa a versão do módulo de Notificação de Exposição instalado no dispositivo do usuário.
setDiagnosisKeysDataMapping()	Esta função pega um objeto

	DiagnosisKeysDataMapping e define o mapeamento dos dados das chaves de diagnóstico, se já não tiver sido alterado recentemente. Se for chamada duas vezes em 7 dias, a segunda chamada não terá efeito e gerará uma exceção com o código de status. FAILED_RATE_LIMITED
getDiagnosisKeysDataMapping()	Esta função retorna o atual DiagnosisKeysDataMapping.
getCalibrationConfidence()	Esta função retorna um Int representando o valor de confiança da calibração para. Um valor mais alto significa maior confiança.
getTemporaryExposureKeyHistory()	Recupera o histórico de chaves do armazenamento de dados no dispositivo para fazer *upload* para o servidor acessível pela Internet. Chamar esse método faz com que o Google Play Services exiba uma caixa de diálogo que solicita o consentimento do usuário para reunir e fazer *upload* de suas chaves de exposição. As chaves retornadas dependem da versão da API. Essas chaves incluem os últimos 14 dias, mas não a chave do dia atual. (Isso é diferente para contas permitidas, que também retornam a chave do dia atual.) O consentimento concedido pelo usuário dura 24 horas. A caixa de diálogo solicitando consentimento aparece apenas uma vez para cada período de 24 horas, independentemente de quantas.
provideDiagnosisKeys()	O provideDiagnosisKeys()método insere um lote de arquivos ou um ou mais lotes de arquivos (v1.5 e superior) contendo informações importantes no banco de dados do dispositivo. Essas chaves devem corresponder aos casos confirmados recuperados de seu servidor acessível pela Internet. Cada lote (mesmo start, countrye batch_count) deve incluir o conjunto completo de arquivos no lote, caso contrário, ele descarta todos eles. As informações sobre o formato do arquivo estão no formato de arquivo de exportação da Exposure Key e verificação.
getExposureSummary()	O getExposureSummary()método recupera o ExposureSummary objeto que corresponde ao token provideDiagnosisKeys() fornecido ao método. O ExposureSummary objeto fornece uma visão geral de alto nível da exposição que um

	usuário experimentou.
getExposureInformation()	O getExposureInformation()método fornece uma versão mais detalhada das informações fornecidas por getExposureSummary(). Se você passar um token de provideDiagnosisKeys()por uma chave de exposição, getExposureInformation()fornece uma lista de ExposureInformation objetos, a partir da qual você pode avaliar o nível de risco da exposição com o usuário. O Google Play Services exibe uma notificação ao usuário sempre que esse método é invocado.
getExposureWindows()	O getExposureWindows()método recupera a lista de janelas de exposição correspondentes às chaves fornecidas provideDiagnosisKeys()com token=TOKEN_A. Como as longas exposições a uma chave são divididas em janelas de até 30 minutos de varredura, uma determinada chave pode levar a várias janelas de exposição se os avistamentos do farol durarem mais de 30 minutos. O vínculo entre eles (em outras palavras, o fato de que todos correspondem à mesma chave) é perdido porque essas janelas são embaralhadas antes de serem retornadas e as chaves subjacentes não são expostas.
getDailySummaries()	O getDailySummaries()método recupera os resumos diários dos dados de exposição. Esta função retorna uma lista de DailySummaryobjetos dos últimos 14 dias (ou menos, se especificado no DailySummariesConfig).

Fonte: Android Developer's (2020)

O Quadro 1 ilustra quais são os métodos disponíveis para utilização na API. Também é informada a descrição desses métodos, como são chamados e suas variáveis de retorno.

Esta informação das funções é essencial para compreender o funcionamento do software em termos de programação no SDK. São eles que farão uso dos recursos do Sistema Operacional e de suas bibliotecas.

3.2. ARQUITETURA DESCENTRALIZADA

De acordo com Roesch (2020), diferentemente do modelo cliente-servidor, no qual os dados são constantemente controlados pelo servidor ao qual a aplicação está conectada, no modelo de arquitetura descentralizada todas as operações e dados ocorrem e ficam localmente nos dispositivos com o aplicativo Corona Warn, sem sofrer controle do servidor.

O Corona Warn é baseado em uma arquitetura descentralizada. Segundo a documentação fornecida pela Android (2020), a abordagem descentralizada do aplicativo é fortemente inspirada no protocolo DP-3T (*Descentralized Privacy-Preserving Proximity Tracing*). Os dados coletados dos dispositivos próximos são armazenados localmente em cada dispositivo, evitando o acesso e o controle dos dados por qualquer outra pessoa.

3.2.1. Protocolo DP-3T

Segundo Vaudenay (2020), o Rastreamento de Proximidade com Preservação de Privacidade Pan-Europeia (PEPP-PT) ofereceu várias soluções, que eram chamadas de centralizadas e descentralizadas. A partir deste protocolo, o Rastreamento de Proximidade com Preservação de Privacidade Descentralizado (DP-3T) foi criado como uma solução descentralizada.

ROESCH (2020) e TRANCOSO (2020) explicam que o projeto DP-3T é um protocolo aberto para o rastreamento de proximidade do COVID-19, usando a funcionalidade do *Bluetooth* em dispositivos móveis, o que assegura que dados pessoais estejam totalmente no aparelho do usuário, havendo, assim, mais privacidade. Assim, o DP-3T tem como objetivo simplificar e acelerar o processo de identificação de pessoas que estiveram em contato com pessoas infectadas, visando minimizar os riscos com a privacidade e segurança e garantir o maior nível de proteção de informações possível.

O protocolo é um trabalho feito por um consórcio internacional, com um time de mais de 25 especialistas, liderado pela professora e pesquisadora Carmela Troncoso (TRANCOSO, 2020).

3.2.2. PROTOCOLO TCN

O protocolo de Número de Contato Temporário (TCN) é um protocolo descentralizado, de rastreamento de primeiro contato com privacidade, que foi desenvolvido pela TCN Coalition (TCN-COALITION, 2020). O protocolo vem sendo construído para ser extensível, com o objetivo de fornecer interoperabilidade entre aplicativos de notificação de exposição.

Segundo LEWIS(2020), os aplicativos, que seguem o protocolo TCN, usam a capacidade dos sistemas Android e iOS para compartilhar um Número de Contato Temporário (TCN) de 128 bits com aplicativos próximos, usando Bluetooth Low Energy (BLE). Para que o Número de Contato Temporário (TCN) seja encontrado de um Android para um IOS, o Android (*sender*) transmite TCN por *Bluetooth* e o IOS (*listener*) recebe essa transmissão diretamente. Ocorre da mesma maneira entre dois sistemas Android.

Segundo a TCN (TCN-COALITION, 2020), quando a transmissão ocorre de um IOS para um Android, o Android (*listener*) sinaliza a disponibilidade como um periférico *Bluetooth e, e*ntão, o *sender* (IOS), atuando como uma central *Bluetooth*, conecta-se a esse periférico e grava seu TCN em um campo exposto pelo periférico e depois se desconecta.

3.3. BLUETOOTH LOW ENERGY

De acordo com WASSOM (2020), a tecnologia de pilha de rede Bluetooth permite a comunicação e troca de dados entre dispositivos portadores da tecnologia *Bluetooth*. Ao usar as APIs *Bluetooth*, um aplicativo Android pode executar as seguintes atividades: Procurar outros dispositivos *Bluetooth*; Consultar o adaptador *Bluetooth* local para verificar a existência de dispositivos *Bluetooth* pareados; Conectar-se a outros dispositivos por meio da descoberta de serviços; Transferir dados de e para outros dispositivos e gerenciar várias conexões.

Segundo a própria Android (ANDROID, 2020), na versão Google Android 4.3 (API de nível 18), o Google lançou APIs compatíveis com o

Bluetooth de baixa energia (Bluetooth Low Energy-BLE, no inglês). O *Bluetooth* de baixa energia foi projetado para fornecer um consumo de energia significativamente menor em comparação com a tecnologia *Bluetooth* tradicional. Alguns exemplos de aplicação são: Transferência de pequenas quantidades de dados entre dispositivos próximos e Interação com sensores de proximidade como os Sensores do Google para oferecer aos usuários uma experiência personalizada com base na localização deles.

Visto as características do Bluetooth de baixa energia e o objetivo da API de Notificação de Exposição, de acordo com Android (2020) a Google e a Apple empregaram a mecânica do BLE em sua API de Notificação de exposição. O Bluetooth de baixa energia, no uso da API, possibilita que os dispositivos troquem constantemente um identificador temporário chamado Rolling Proximity Identifier (RPI), de acordo com Google (2020). O dispositivo transmite os identificadores, ao mesmo tempo que buscam por identificadores próximos e recebem estes.

3.3.1. Rolling Proximity Identifier (RPI)

Segundo a documentação fornecida pela equipe do Corona Warn no GitHub (CORONA WARN, 2020), os identificadores de proximidade rotativa (RPI) são números de identificação enviados entre os dispositivos por Bluetooth. Para garantir a privacidade e evitar o rastreamento dos padrões de movimento do usuário do aplicativo, esses identificadores transmitidos são apenas temporários e mudam constantemente. Cada vez que o endereço MAC do Bluetooth muda, é derivado um novo identificador de proximidade rotativa. Os identificadores são derivados de uma Chave de Exposição Temporária (TEK). Junto com os identificadores também vem metadados cifrados associados. O metadado contém a potência de transmissão do sinal Bluetooth para melhor aproximação da distância dos dispositivos. A RPI e estes metadados são associados a um intervalo do dia gerando assim uma identificação para um dia e hora para as TEKs do dispositivo.

3.3.2. Chave de Exposição Temporária (TEK)

No cenário do Corona Warn (CORONA WARN, 2020), as chaves são constantemente comparadas com uma listagem de chaves positivadas, ou seja, comparadas com chaves de dispositivos que o usuário foi diagnosticado com COVID-19. Como ocorre isso? Um usuário ao ser diagnosticado com o resultado positivo para vírus, este disponibiliza suas chaves para o servidor da Google e lá as chaves ficam disponíveis para serem baixadas pelos dispositivos, a fim de alimentar a lista de chaves de infectados. Os dispositivos constantemente realizam *download* dessas chaves. Tendo obtido essas chaves, o dispositivo usará essas chaves de exposição temporária para gerar todos os *Roling proximity identifier* que podem ter sido transmitidos ao longo de um determinado dia. Após gerados, esses RPIs são comparados em relação à lista de RPIs recebidos ao longo dos dias anteriores. No Corona Warn, cada chave de exposição temporária é substituída à meia-noite (UTC) todos os dias por meio de criptografia. As chaves são publicadas enviando o documento JSON apropriado no corpo de uma solicitação HTTP POST para o servidor.

3.4. OUTRAS BIBLIOTECAS UTILIZADAS

De acordo com o que WOLF (2020) descreve na parte da documentação, outras importantes bibliotecas são utilizadas na construção do aplicativo para o sistema operacional Android. Dentre elas estão a biblioteca de leitura de códigos de barra e a biblioteca Joda Time.

A biblioteca para leitura de códigos de barra para utilizada é a Android ZXing. Segundo Kistner (2020), a dependência é adicionada utilizando Gradle (sistema de automatização de Build). Um exemplo de código de aplicação pode ser visto na Figura 5, no arquivo build.gradle. Para outros projetos, basta-se utilizar este mesmo trecho de código.

Figura 5: Adicionando ZXing

```
repositories {
    jcenter()
}

dependencies {
    implementation 'com.journeyapps:zxing-android-embedded:4.1.0'
    implementation 'androidx.appcompat:appcompat:1.0.2'
}

android {
    buildToolsVersion '28.0.3' // Older versions may give compile errors
}
```

Fonte: Kistner (2020)

Ainda segundo WOLF (2020), também é empregada a biblioteca Joda Time, utilizada como classe padrão de manipulação de data e hora. Segundo a própria documentação do Corona Warn, este é usado para cálculos de data, calendário e manipulação de fuso horário. Na Figura 6 é dado um exemplo do uso no código apresentado.

Figura 6: Exemplo de uso do Joda Time

```
public  boolean isAfterPayDay ( DateTime datetime) {
   if (datetime . getMonthOfYear () ==  2 ) {    // Fevereiro é o mês 2 !!
    return datetime . getDayOfMonth () >  26 ;
  }
  return datetime . getDayOfMonth () >  28 ;
}

public  Days daysToNewYear ( LocalDate fromDate) {
   LocalDate newYear = fromDate . plusYears ( 1 ) . withDayOfYear ( 1 );
  dias de retorno  . diasentre (deData, novoAno);
}

public  boolean isRentalOverdue ( DateTime datetimeRented) {
   Período rentalPeriod =  new  Período () . withDays ( 2 ) . withHours ( 12 );
  return datetimeRented . mais (período de aluguel) . isBeforeNow ();
}

public  String getBirthMonthText ( LocalDate dateOfBirth) {
   return dateOfBirth . monthOfYear () . getAsText ( Locale . ENGLISH );
}
```

Fonte: Colebourne (2020)

Segundo Colebourne (2020) o criador da biblioteca, ela é um ótima alternativa para as classes Java de data e hora. Uma biblioteca semelhante só foi criada a partir da API 26, o que justifica a escolha desta biblioteca para o projeto.

3.5. CONSIDERAÇÕES PARCIAIS

O aplicativo Corona Warn, para o sistema operacional Google Android, foi criado com o propósito de auxiliar governos, empresas e a população de modo geral no combate à pandemia de COVID-19. O principal método no contato da proliferação é o distanciamento social e com isso se faz importante mapear se uma pessoa teve contato com infectados, para que esta tome os devidos procedimentos necessários.

A aplicação foi desenvolvida com base na API de notificações de exposição, que por sua vez foi construída com base na mecânica da tecnologia Bluetooth de baixa energia. Esta tecnologia Bluetooth de baixa energia, consome uma quantidade de energia significativamente menor que a tecnologia Bluetooth tradicional, o que possibilita que os dispositivos troquem identificadores temporários. A API de notificações de exposição cria chaves de identificação únicas e temporárias, que por sua vez geram identificadores que são trocados entre dispositivos e armazenados localmente. Caso um usuário venha a ser diagnosticado com COVID-19, este pode submeter suas chaves para o servidor do Google e essas chaves então podem ser obtidas por outros dispositivos. Com base nessas chaves, a aplicação consegue identificar se a pessoa teve contato próximo com alguma pessoa positivada com COVID-19 e tendo esse dado, notifica a pessoa sobre o risco de contágio ao que ela foi exposta.

4. ESTUDO DE CASO – CORONA WARN

Compreendido o sistema Android e suas aplicações, neste capítulo é detalhada a aplicação Corona Warn: como a API de exposição faz uso dos recursos para seu funcionamento; como é implementada a troca de dados de exames laboratoriais; o cálculo de nível de risco de exposição e suas implicações; e, por fim, a infraestrutura necessária e a mecânica de troca e processamento de dados.

4.1. ARQUITETURA DO SISTEMA

No Capítulo 3, foi citada a utilização da API de notificação de exposição para o desenvolvimento do Corona Warn. Citou-se também a utilização dos serviços do Google Play por parte da API, pois é nele que se encontra a funcionalidade do Bluetooth, sendo os serviços do Google Play responsáveis pelas duas principais tarefas agora descritas:

1. Gerenciamento de chaves aleatórias diárias: Geração das chaves de exposição diárias e identificadores de proximidade rotativa; disponibilização das chaves caso usuários diagnosticados com COVID-19 e incluir um número de intervalo que indica a data chave; recebimento de chaves e seus metadados associados.

2. Gerenciamento de transmissão e coleta do Bluetooth: Envio dos identificadores de proximidade rotativa; Busca por identificadores de proximidade rotativa; armazenamento dos identificadores no dispositivo, identificação se o usuário esteve em contato próximo de algum dispositivo com caso confirmado, procedimento esse feto através da comparação entre os identificadores recebidos nos últimos 14 dias e a lista de chaves diagnosticadas com COVID-19; cálculo e geração de um nível de risco de exposição para o aplicativo.

No Corona Warn a implementação da API é feita invocando as classes *ExposureConfiguration, TemporaryExposureKey* e ExposureSummary. Um exemplo da implementação destas classes é feito conforme a Figura 7.

Figura 7: Chamada de classes no Android Studio

```
import com.google.android.gms.nearby.Nearby
import com.google.android.gms.nearby.exposurenotification.ExposureConfiguration
import com.google.android.gms.nearby.exposurenotification.ExposureConfiguration.ExposureConfigurationBuilder
import com.google.android.gms.nearby.exposurenotification.ExposureSummary
import com.google.android.gms.nearby.exposurenotification.TemporaryExposureKey
```

Fonte: Autores (2020)

Cada uma destas classes funciona da seguinte forma:

- *ExposureConfiguration*: Classe que configura o comportamento dos metodos getExposureSummary()- método que recupera as informações

da classe ExposureSummary- e também configura o do método ExposureInformation()- método que fornece uma versão mais detalhada das informações fornecidas por getExposureSummary().

- *TemporaryExposureKey*: Classe contém hora em que a chave foi gerada, em intervalos de 10 minutos; o número de intervalos de 10 minutos para os quais uma chave é válida; tipo de diagnóstico da chave.

- *ExposureSummary*: Classe que contém de maneira resumida as informações da exposição. Dentre essas informações está o número de dias desde a última exposição com uma chave diagnosticada, o nível de risco de transmissão mais alto de todos os incidentes de exposição e o número de chaves de diagnóstico correspondentes.

4.2. RECUPERAÇÃO DE DADOS DO LABORATÓRIO

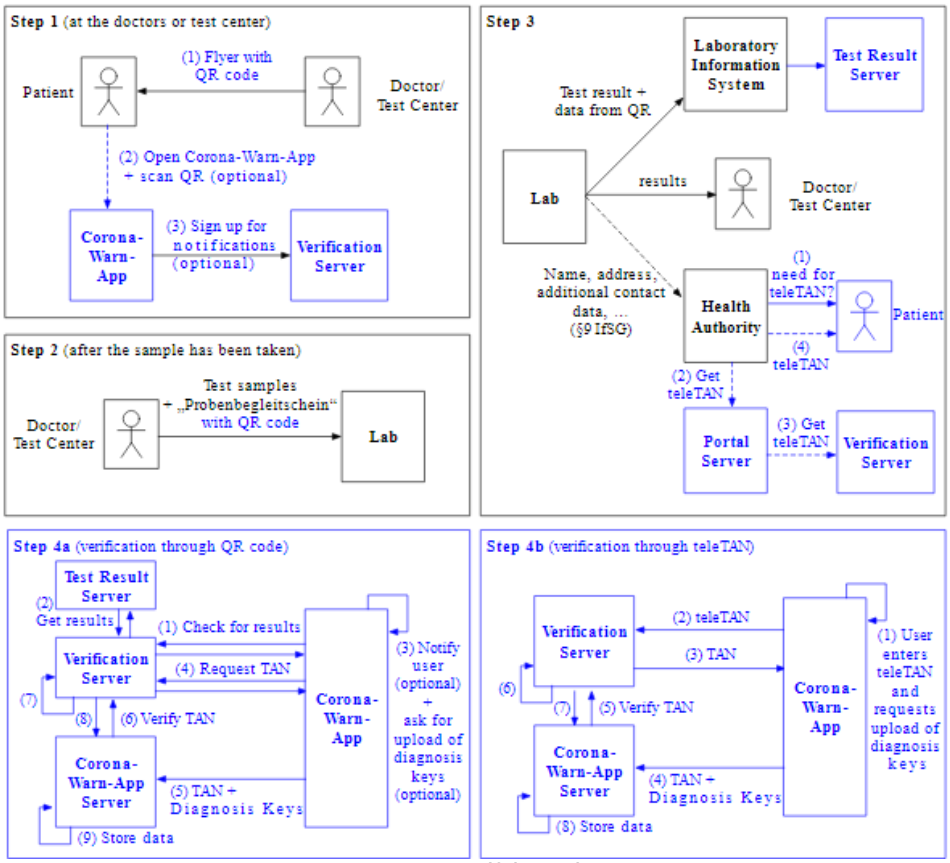
Após a confirmação do teste positivo para COVID-19, o usuário tem a opção de informar ou não para outros usuários que foi diagnosticado. Informar outros usuários significa enviar suas chaves para o servidor do Corona Warn, para que este disponibilize para download. Caso o usuário opte pelo compartilhamento, essa é uma informação secreta, ou seja, não é apresentada para outros usuários quem foi a pessoa que teve o contato.

Segundo Wolf (2020), o processo pode ser verificado na Figura 8 e ocorre sequencialmente da seguinte maneira: No próprio consultório, caso testado positivo, o usuário queira compartilhar seu resultado, o consultório cadastrado na solução, gera um QR Code; o usuário então faz a leitura desse QR Code com seu aplicativo e a API coloca uma chamada de serviço web (REST) para o servidor de verificação do Corona Warn. Essa chamada vincula o dispositivo aos dados do QR Code através de um Token gerado no servidor. O QR Code também é enviado para o software executado localmente no laboratório. A aplicação executada localmente no laboratório então transmite o resultado do teste para o Sistema de Informação do Laboratório junto com o GUID do código QR. O Sistema de Informações do Laboratório pública junto

com o resultado do teste no Servidor de resultado do Teste, que por sua vez o torna disponível para o servidor de verificação.

Essa verificação se dá para maior segurança antes do envio das chaves. Essa verificação pode ser feita de duas maneiras: Através do QR Code e através de um token chamado TAN, como pode ser visto na Figura 8 etapa 4a e 4b. De maneira geral, o servidor de verificação busca se a informação está no servidor do Corona Warn, ou seja, se o resultado coincide.

Figura 8: Fluxo de dados



Fonte: Wolf (2020)

Após a verificação, as chaves temporárias, chamadas de chaves de diagnóstico, quando associadas a um resultado de teste, já podem ser enviadas para o servidor do Corona Warn. De acordo Wolf (2020), existe uma programação para o envio das chaves temporárias. A aplicação necessita enviar ao servidor as 14 chaves temporárias dos 14 dias anteriores mais a chave do dia atual. O problema é que a chave do dia atual ainda pode gerar identificadores de proximidade, então seria um risco à segurança enviá-la antes do fim do dia. Desse modo, o *upload* ao servidor ocorre, enviando

primeiro as 14 chaves dos dias anteriores e após esse envio, é enviada a do dia atual e esta é imediatamente substituída por outra chave. Nos dias subsequentes nenhuma chave é enviada. Na aplicação, as chaves citadas são recuperadas através do método *getTemporaryExposureKeyHistory*() e o *upload* é realizado no seguinte trecho de código da Figura 9.

Figura 9: Implementação das chaves

```
executeState(RETRIEVE_TAN_AND_SUBMIT_KEYS) {
    PlaybookImpl(WebRequestBuilder.getInstance()).submission(
        registrationToken,
        temporaryExposureKeyList
    )
}
```

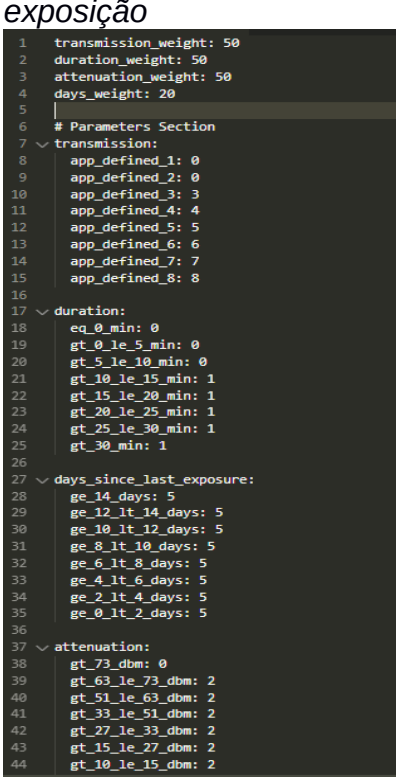
Fonte: Autores (2020)

Vale salientar que o servidor do Corona Warn é algo que o governo do país em que o serviço está disponível precisa ser implementado e seus cálculos devem levar em consideração a quantidade de dados necessária para cada aplicação (cada cidadão), o tamanho da população e uma taxa de contaminação esperada.

4.3. CÁLCULO DE NÍVEL DE RISCO DE EXPOSIÇÃO

Segundo Kowark (2020), caso um dispositivo tenha contato próximo com outro dispositivo na qual o usuário foi constatado com COVID-19, a aplicação utiliza quatro variáveis para calcular o nível de risco de exposição e com isso, atribuir uma pontuação. Cada uma das quatro categorias de risco recebe um valor de entrada do evento que é então mapeado de acordo com a categoria e direcionado para um intervalo de valor de entrada predefinido. Após alocados, estes quatro intervalos recebem uma pontuação entre 0 e 8, onde 0 representa um risco muito baixo e 8 representa um risco muito alto. O parâmetro para atribuição da pontuação é definido na própria implementação da aplicação, no trecho de código que pode ser observado na Figura 10.

Figura 10: Dados de exposição



```
transmission_weight: 50
duration_weight: 50
attenuation_weight: 50
days_weight: 20

# Parameters Section
transmission:
  app_defined_1: 0
  app_defined_2: 0
  app_defined_3: 3
  app_defined_4: 4
  app_defined_5: 5
  app_defined_6: 6
  app_defined_7: 7
  app_defined_8: 8

duration:
  eq_0_min: 0
  gt_0_le_5_min: 0
  gt_5_le_10_min: 0
  gt_10_le_15_min: 1
  gt_15_le_20_min: 1
  gt_20_le_25_min: 1
  gt_25_le_30_min: 1
  gt_30_min: 1

days_since_last_exposure:
  ge_14_days: 5
  ge_12_lt_14_days: 5
  ge_10_lt_12_days: 5
  ge_8_lt_10_days: 5
  ge_6_lt_8_days: 5
  ge_4_lt_6_days: 5
  ge_2_lt_4_days: 5
  ge_0_lt_2_days: 5

attenuation:
  gt_73_dbm: 0
  gt_63_le_73_dbm: 2
  gt_51_le_63_dbm: 2
  gt_33_le_51_dbm: 2
  gt_27_le_33_dbm: 2
  gt_15_le_27_dbm: 2
  gt_10_le_15_dbm: 2
```

Fonte: Autores (2020)

O produto das quatro pontuações de risco é a pontuação de risco total da exposição individual, sendo este um intervalo de 0-4096 pontos (12 bits de armazenamento). As quatro variáveis consideradas para o cálculo são:

- Dias desde a exposição. Com base nos identificadores de proximidade, a aplicação pode saber a data da exposição e quantos dias fazem da data atual até a data do contato com o dispositivo que posteriormente foi notificado com resultado positivo ao COVID-19. Para a realização do cálculo é imputado um valor entre 0 e 14, que se referem ao número de dias.
- Tempo de exposição. Com base também nos identificadores de proximidade juntamente com os metadados trocados, é possível identificar o tempo ao qual o dispositivo ficou exposto a outro dispositivo positivado. Esse campo recebe uma duração, em minutos, entre 0 e 30.
- Atenuação do sinal. Essa categoria diz respeito a força do sinal do Bluetooth de baixa energia, não serão abordados maiores

detalhes, porém essa força é medida através da potência de transmissão menos a potência recebida.

Attenuation=TX_power-(RSSI_measured+RSSI_correction).

(Essa atenuação pode ser entendida como a distância entre os dispositivos)

- Nível de risco de transmissão. Esse fator está associado à chave de diagnóstico do dispositivo positivado. Esta chave que por sua vez foi anteriormente enviada para o servidor da aplicação, juntamente com os metadados associados. Esse nível contém um valor de 1 a 8 que diz respeito a probabilidade de transmissão da doença em certo período. Esse número é calculado e definido pela autoridade de saúde- no caso do Corona Warn, a entidade capacitada alemã. Estes valores específicos estão na aplicação no seguinte trecho de código na Figura 11.

Figura 11: Implementação do nível de risco de transmissão

```
object ProtoFormatConverterExtensions {

    private const val ROLLING_PERIOD = 144
    private const val DEFAULT_TRANSMISSION_RISK_LEVEL = 1
    private const val TRANSMISSION_RISK_DAY_0 = 5
    private const val TRANSMISSION_RISK_DAY_1 = 6
    private const val TRANSMISSION_RISK_DAY_2 = 8
    private const val TRANSMISSION_RISK_DAY_3 = 8
    private const val TRANSMISSION_RISK_DAY_4 = 8
    private const val TRANSMISSION_RISK_DAY_5 = 5
    private const val TRANSMISSION_RISK_DAY_6 = 3
    private const val TRANSMISSION_RISK_DAY_7 = 1
    private val DEFAULT_TRANSMISSION_RISK_VECTOR = intArrayOf(
        TRANSMISSION_RISK_DAY_0,
        TRANSMISSION_RISK_DAY_1,
        TRANSMISSION_RISK_DAY_2,
        TRANSMISSION_RISK_DAY_3,
        TRANSMISSION_RISK_DAY_4,
        TRANSMISSION_RISK_DAY_5,
        TRANSMISSION_RISK_DAY_6,
        TRANSMISSION_RISK_DAY_7
    )
    private const val MAXIMUM_KEYS = 14

    fun List<TemporaryExposureKey>.limitKeyCount() =
        this.sortedWith(compareByDescending { it.rollingStartIntervalNumber }).take(MAXIMUM_KEYS)

    fun List<TemporaryExposureKey>.transformKeyHistoryToExternalFormat() =
        this.sortedWith(compareByDescending { it.rollingStartIntervalNumber })
            .mapIndexed { index, it ->
                // The latest key we receive is from yesterday (i.e. 1 day ago),
                // thus we need use index+1
                val riskValue =
                    if (index + 1 <= DEFAULT_TRANSMISSION_RISK_VECTOR.lastIndex)
                        DEFAULT_TRANSMISSION_RISK_VECTOR[index + 1]
                    else
                        DEFAULT_TRANSMISSION_RISK_LEVEL
                KeyExportFormat.TemporaryExposureKey.newBuilder()
                    .setKeyData(ByteString.readFrom(it.keyData.inputStream()))
                    .setRollingStartIntervalNumber(it.rollingStartIntervalNumber)
                    .setRollingPeriod(ROLLING_PERIOD)
                    .setTransmissionRiskLevel(riskValue)
                    .build()
            }

    fun AppleLegacyKeyExchange.Key.convertToGoogleKey(): KeyExportFormat.TemporaryExposureKey =
        KeyExportFormat.TemporaryExposureKey.newBuilder()
            .setKeyData(this.keyData)
            .setRollingPeriod(this.rollingPeriod)
            .setRollingStartIntervalNumber(this.rollingStartNumber)
            .setTransmissionRiskLevel(this.transmissionRiskLevel)
            .build()
```

Fonte: Autores (2020)

Com base nos dados obtidos através da interface Bluetooth, é calculada a pontuação do risco. A Figura 12 apresenta como a pontuação de risco total é calculada.

Figura 12: Calculo pontuação de risco total

Fonte: Wolf (2020)

O valor resultante é então a multiplicação do valor de risco diário (de 0 para 0 dias e de 8 para 14 dias de exposição ao risco) pelo tempo de exposição com determinada pessoa (variando de 0 a um valor de 8 caso a pessoa fique próxima por mais de 30 minutos) pela distância à esta pessoa (um valor de 0 para uma distância maior que 7,3 metros e 8 para uma distância menor ou igual a 1 metro) e também pela chave da pessoa com quem se teve contato (também variando de 0 a 8 caso a pessoa esteja positivada e também levando-se em consideração o estágio da doença em dias). Desta multiplicação é resultado o valor, sendo o máximo equivalente a 8 x 8 x 8 x 8 que é igula a 4096 conforme dito no início deste tópico.

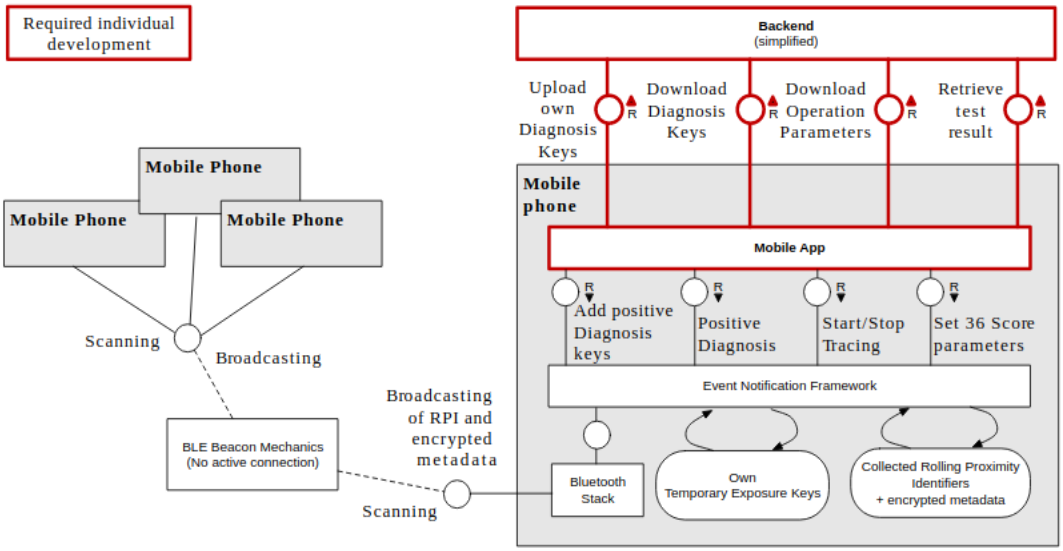
4.4. TRANSFERÊNCIA E PROCESSAMENTO DE DADOS

De acordo com Wolf (2020) e conforme já explicado no capítulo anterior com relação aos protocolos DP-3T e TCN, os dados se encontram de forma descentralizada e tenta-se ao máximo manter a confidencialidade dos dados dos usuários do aplicativo. Uma série de eventos são realizados durante o ciclo

de vida da aplicação, o que requer uma série de dados distintos sendo trocados.

Como o foco deste trabalho é no sistema operacional do *smartphone*, no aplicativo e nos recursos que ele utiliza do sistema, o mais importante aqui é se atentar que há um armazenamento nos dispositivos móveis de chaves RPI tanto para cálculo do risco de exposição quanto para notificação de pessoas próximas positivadas, sempre mantendo a confidencialidade e com mecanismos de confiabilidade dos dados. A Figura 13 apresenta a arquitetura de troca e armazenamento de dados no Corona Warn.

Figura 13: Troca e armazenamento de dados



Fonte: Wolf (2020)

A infraestrutura necessária conta também com um servidor para onde notificações de testes positivos para COVID-19 são enviados e posteriormente as chaves RPIs de testes positivos são transmitidas aos smartphones. A Figura 13 ajuda a compreender melhor todo esse mecanismo de troca e armazenamento de dados.

Como pode-se ver, esta aplicação não é nada trivial em termos de infraestrutura e troca de dados e por isso foi a escolhida para este trabalho já que se distancia um pouco do que é visto em um ambiente acadêmico e se

aproxima das soluções que são implementadas comercialmente e que é visto ao se finalizar os estudos de graduação.

4.5. CONSIDERAÇÕES PARCIAIS

Neste capítulo é possível observar o uso da API de notificação de exposição no código do aplicativo Corona Warn. A API de lde notificação de exposição faz parte da arquitetura de plataforma. No aplicativo, as classes e seus respectivos métodos são responsáveis pela criação, troca e envio das chaves temporárias e dos identificadores de proximidade.

5. CONSIDERAÇÕES FINAIS

Com o trabalho atingimos o objetivo de compreender os detalhes da arquitetura e componentes básicos do sistema operacional Android. O estudo se aprofundou na API de notificação de exposição, onde foram detalhadas as classes e a implementação no código do aplicativo Corona Warn, atingindo assim o objetivo de compreender a API de notificação de exposição. Também, foi possível compreender o funcionamento da aplicação estudada, o Corona Warn.

O aplicativo Corona Warn foi criado no ano de 2020 com o intuito de rastrear as cadeias de infecção de SARS-CoV-2 e contribuir no combate a disseminação da doença. Foi desenvolvido para o sistema operacional Android, que é atualmente o sistema operacional mais utilizado do mundo. De acordo com Lecheta (2015) a plataforma Android oferece aos desenvolvedores de aplicativos mobile uma gama de recursos como componentes, bibliotecas e interface gráfica e conta com ferramentas para o desenvolvimento dos aplicativos.

O aplicativo analisado neste estudo de caso está voltado para a área da saúde, em específico com o propósito de auxiliar governos, empresas e a população de modo geral no combate à pandemia de COVID-19. Esta

aplicação foi desenvolvida com base na API de notificação de exposição, desenvolvida pela Apple e o Google. A API de notificações de exposição, por sua vez, foi construída com base na mecânica da tecnologia Bluetooth de baixa energia que permite que os dispositivos celulares troquem identificadores de proximidade temporários. Os identificadores de proximidade temporários são gerados a partir de chaves temporárias geradas diariamente pela API de notificação de exposição. Caso um usuário seja contaminado com COVID-19, esse usuário tem a opção de compartilhar suas chaves temporárias para que outros dispositivos, através do método *getTemporaryExposureKeyHistory*, baixem essa chave do servidor, recriem todos os identificadores de proximidade derivados dessa chave contaminada e realizem a comparação de identificadores, sendo assim possível identificar se esse dispositivo teve contato com algum infectado dentro de 14 dias. A API é implementada através da chamada das classes *ExposureConfiguration*, *TemporaryExposureKey* e *ExposureSummary*.

As principais dificuldades encontradas para o desenvolvimento do presente trabalho, foi justamente na fase do estudo de caso. No desenrolar das pesquisas, percebeu-se que, pelo fato de ser uma aplicação completa, não valeria a pena se aprofundar em certos temas propostos para o capítulo, pois assim estaria fugindo do objetivo inicial do estudo. O que gerou uma dificuldade em filtrar e identificar no que focar. A exemplo, a temática da arquitetura e transferência de dados entre servidores da aplicação, que nem mesmo se comunicam diretamente com o dispositivo e o sistema operacional Android. Outra dificuldade encontrada deu-se na proposta de apresentar a implementação da API na prática no código da aplicação. Nesse ponto, apesar de toda documentação e código estar disponível publicamente, o código se encontra espalhado tornando a localização destes um tanto complexa. Ademais, nenhum dos integrantes da equipe tem uma grande familiaridade com as linguagens de programação utilizada.

Em relação ao cronograma proposto inicialmente, houve alterações no que diz respeito ao tempo total de desenvolvimento do trabalho. O tempo real gasto no desenvolvimento do estudo acompanhou as mudanças do cronograma de aulas do semestre 2020/2 da Universidade do Estado de Santa

Catarina. Com mudanças, compreende-se por duas extensões do calendário acadêmico e por consequência do prazo das entregas finais do proposto trabalho. Essas mudanças acabaram sendo bem vindas pela equipe, já que a pandemia do COVID-19, criou um cenário onde rotinas em vários âmbitos foram alteradas, afetando o tempo disponível dos autores, empregado no desenvolvimento do presente estudo.

Como sugestão para trabalhos futuros, sugere-se a condução de trabalhos semelhantes ao deste, focados no Coronavírus SUS (Android, iOS), que é também utiliza a API de notificação de exposição. Também se sugere a condução de estudos para verificar e que tragam dados do impacto positivo ou negativo, das aplicações que utilizam essa API, no combate à pandemia do COVID-19.

6. REFERÊNCIAS

ANDROID DEVELOPERS. Disponível em: < https://developer.android.com/about/versions/oreo/android-8.0>. Acesso em 07 ago. 2020.

ANDROID DEVELOPER. Visão geral do Bluetooth de baixa energia. Disponível em: <https://developer.android.com/guide/topics/connectivity/bluetooth-le>. Acesso em: 22 ago. 2020.

ARQUITETURA ANDROID. Disponível em: <https://developer.android.com/guide/platform/index.html>. Acesso em 07 ago. 2020.

BLUETOOTH. Intro to Bluetooth Low Energy. Disponível em: <https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-low-energy/ >. Acesso em: 22 ago. 2020.

BUNDESREGIERUNG. Coronavirus warning app. Disponível em: <https://www.bundesregierung.de/breg-de/themen/corona-warn-app/corona-warn-app-englisch/how-does-the-corona-warn-app-work-and-what-does-it-do—1758870>. Acesso em: 3 ago. 2020.

BUTCHER, Isabel. Até 2023, 5,3 bilhões de pessoas estarão conectadas à Internet, diz Cisco. Disponível em:< https://www.mobiletime.com.br/noticias/18/02/2020/ate-2023-53-bilhoes-de-pessoas-estarao-conectadas-a-internet-diz-cisco/#:~:text=A%20base%20de%20dispositivos%20m%C3%B3veis,bilh%C3%B5es%2C%20incremento%20de%2049%25>. Acesso em: 11 ago. 2020.

COLEBOURNE, Stephen. Joda-Time. Disponível em: <https://github.com/JodaOrg/joda-time>. Acesso em: 11 ago. 2020.

CORONA WARN. Projeto de código aberto Corona-Warn-App. Disponível em: <https://www.coronawarn.app/en/#howto>. Acesso em: 01 set. 2020.

COSTA, Norben P.O; DUARTE FILHO, Nemésio F. Análise e Avaliação Funcional de Sistemas Operacionais Móveis: Vantagens e Desvantagens. Revista de Sistemas e Computação, Salvador, v. 3, n. 1, p. 66-67, jan/jun. 2013.

DEUTSCHE WELLE. Alemanha lança aplicativo de rastreamento para covid-19. Disponível em: <https://p.dw.com/p/3dqsw >. Acesso em: 01 set. 2020.

DUTSON, Phil; SCHWARZ, Ronan; STEELE, James; TO, Nelson. The Android Developer's Cookbook: Building Applications with the Android SDK. Developer's Library, United States of America, 2013.

GOOGLE. Exposure-notifications-android. Disponível em: <https://github.com/google/exposure-notifications-android>. Acesso em: 11 ago. 2020.

GOOGLE DEVELOPERS. Exposure Notifications implementation guide. Disponível em: <https://developers.google.com/android/exposure-notifications/exposure-notifications-api>. Acesso em: 10 ago. 2020.

GUO-HONG, Shao. Application Development Research Based on Android Platform. 2014 7th International Conference on Intelligent Computation Technology and Automation, Changsha, China, v. 1, n. 1, p. 1, jan./2018. Disponível em: <https://ieeexplore.ieee.org/document/7003608>. Acesso em: 22 ago. 2020.

KISTNER, Ralf. Zxing Android Barcode Scanner application. Disponível em: <https://github.com/journeyapps/zxing-android-embedded>. Acesso em: 08 ago. 2020.

KOWARK, Thomas. Cwa-app-android. Disponível em: <https://github.com/corona-warn-app/cwa-app-android/blob/master/docs/architecture-overview.md>. Acesso em: 10 ago. 2020.

LECHETA, Ricardo R. Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK. Ed. Novatec. 5ª ed. São Paulo, 2015.

LEWIS, Dana. TCNCoalition. Disponível em: <https://github.com/TCNCoalition/TCN>. Acesso em: 13 set. 2020.

MEYER, Maximiliano. Disponivel em: < https://www.oficinadanet.com.br/post/13939-a-historia-do-android>. Acesso em 07 ago. 2020.

ROESCH, Simon. DP3T-SDK for Android. Disponível em: < https://github.com/DP-3T/dp3t-sdk-android#introduction>. Acesso em: 12 set. 2020.

SARKAR, Anirban; GOYAL, Ayush; HICKS, David. Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems. 2019 Third International conference on I-SMAC, Palladam, India, India, v. 1, n. 1, p. 1, dez./2019. Disponível em:< https://ieeexplore.ieee.org/document/9032440>. Acesso em: 22 ago. 2020.

TCN-COALITION. About TCN. Disponível em: <https://tcn-coalition.org/>. Acesso em: 22 ago. 2020.

TOCARNIA, Mariana. Disponível em: <https://agenciabrasil.ebc.com.br/economia/noticia/2020-04/celular-e-o-principal-meio-de-acesso-internet-no-pais>. Acesso em: 17 ago. 2020.

TRANCOSO, Carmela. DP-3T / documents. Disponível em: <https://github.com/DP-3T/documents/find/master>. Acesso em: 12 set. 2020.

VAUDENAY, Serge. Centralized or Decentralized? The Contact Tracing Dilemma. Disponível em: <https://eprint.iacr.org/eprint-bin/getfile.pl?entry=2020/531&version=20200507:064545&file=531.pdf>. Acesso em: 12 set. 2020.

WASSOM, Brian D. Bluetooth Low Energy. Disponível em: <https://www.sciencedirect.com/topics/computer-science/bluetooth-low-energy>. Acesso em: 22 ago. 2020.

WOLF, Sebastian. Cwa-documentation. Disponível em: <https://github.com/corona-warn-app/cwa-documentation>. Acesso em: 21 ago. 2020.

WORD HEALTH ORGANIZATION. WHO Coronavirus Disease (COVID-19) Dashboard. Disponível em:<https://covid19.who.int/>. Acesso em: 10 ago. 2020.

ZURIARRAIN, José Mendiola. Android já é o sistema operacional mais usado do mundo. Disponível em: <http://brasil.elpais.com/brasil/2017/04/04/tecnologia/1491296467_396232.html >. Acesso em 20 jun de 2020.