UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Pedro Augusto Freddi Rodrigues

CheckIn@UFSC: Aplicativo iOS para a detecção de exposição à COVID-19 nas dependências da UFSC

Florianópolis
2021

Pedro Augusto Freddi Rodrigues

CheckIn@UFSC: Aplicativo iOS para a detecção de exposição à COVID-19 nas dependências da UFSC

Trabalho de Conclusão de Curso submetido ao Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Sistemas de Informação.
Orientador: Prof. Jônata Tyska Carvalho, Dr.
Coorientador: Prof. Mateus Grellert, Dr.

Florianópolis
2021

AGRADECIMENTOS

Gostaria de começar agradecendo a minha mãe Shirley Andrezza Freddi Barbosa, pois sem o apoio dela talvez esse trabalho nunca teria sido executado. Não poderia deixar de registrar meus agradecimentos à minha amada esposa Lígia Sell, pois seu apoio foi essencial para que eu pudesse chegar até aqui. Agradeço também professores, familiares e amigos, que estiveram junto comigo nessa trajetória da graduação.

Sem todos vocês isso aqui não seria possível! A vocês minha eterna gratidão!

“Nós precisamos que a sociedade também seja ética. O cidadão não pode reclamar de um servidor que não é ético, se ele fura a fila do cinema, se atravessa a grama na praça pública em frente a uma placa que diz que é proibido pisar ali. Precisa haver um comprometimento da sociedade com a ética, porque o Estado que aí está não caiu do céu nem saiu do inferno, nós construímos. Não podemos cobrar do outro o que não fazemos.”

(LÚCIA Cármen, 2016)

RESUMO

O presente trabalho tem como objetivo apresentar o desenvolvimento de um aplicativo para o sistema operacional iOS em parceria com o projeto de pesquisa do CheckIn@UFSC que permita realizar a identificação de contato com outras pessoas que possam ter sido infectadas pelo COVID-19 utilizando o protocolo de rastremento anônimo e *open source*: *Covid Community Alert*. As principais funcionalidades do aplicativo serão: monitorar dispositivos próximos que possuam o aplicativo por meio de conexão bluetooth, permitir que o usuário realize o autodiagnóstico no aplicativo, notificar o usuário em caso de contato com outro usuário suspeito de infecção ou mesmo infectado. Além disso, esse projeto prevê também o teste de comunicação da aplicação entre diferentes dispositivos, buscando aferir se os aparelhos conseguem detectar outros que estejam próximos e utilizam o mesmo aplicativo.

Palavras-chave: Rastreamento de contatos. iOS. COVID-19. Bluetooth.

ABSTRACT

This paper presents the development of an iOS app in partnership with the CheckIn@UFSC research project. The app allows the identification of contact with other people who may have been infected by COVID-19 using the anonymous tracking protocol and open source Covid Community Alert. The main functionalities of the application will be: monitor nearby devices that have the application using Bluetooth connection, allow the user to submit your self-diagnosis, and notify you in case of contact with another user suspected of infection or even infected.

Keywords: Contact tracing. iOS. COVID-19. Bluetooth.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

BLE	*Bluetooth Low Energy* - *Bluetooth* de baixa energia
IDE	*Integrated development environment* - Ambiente de desenvolvimento integrado
JSON	JavaScript Object Notation - Notação de Objetos JavaScript
OMS	Organização Mundial da Saúde
URL	*Uniform Resource Locator* - Localizador Uniforme de Recursos
XML	Extensible Markup Language - Linguagem de Marcação Extensível

SUMÁRIO

1 INTRODUÇÃO

A humanidade como parte da natureza está passível de consequências das próprias ações no meio como também da ação dos demais integrantes desse mesmo universo. A pandemia de COVID-19 é um claro exemplo de como essa interação pode ser danosa aos seres humanos quando não se possui conhecimento sobre o comportamento dos coadjuvantes presentes no meio ambiente.

De acordo com dados cronológicos(25) entre o anúncio do surto viral de uma síndrome respiratória grave em Wuhan na China até a declaração de pandemia dada pela Organização Mundial da Saúde (OMS) se passaram 90 dias e nesse período já eram contabilizados pela entidade mais de 118 mil casos em 114 países e 4,2 mil mortes anunciadas, fora um aumento de 13 vezes nos casos notificados fora da China(27).

Com a escalada de casos, governos pelo mundo se viram na necessidade tomar controle da situação para evitar que a catástrofe atingisse ainda mais pessoas e para isso era necessário converter os dados que estavam à disposição em informações que auxiliassem nas tomadas de decisão. Uma das soluções que surgiram à época foi o *dashboard* criado pelo Centro de Sistemas em Engenharia e Ciência da Universidade John Hopkins(12), uma das primeiras soluções de monitoramento de casos notificados por entidades de saúde ao nível mundial. Portanto, iniciativas relacionadas à saúde digital(14) foram aliadas para a mitigação da propagação da doença, sendo uma delas o uso de aplicativos para a identificação de indivíduos que tiveram contato com um paciente infectado, foco do presente trabalho.

O processo de rastrear quais foram as pessoas que estiveram em contato com um paciente positivo para COVID-19 é chamado de rastreamento de contatos. Essa atividade é realizada é após a descoberta da infecção, por meio uma entrevista que visa obter do paciente quais foram as pessoas com que ele teve contado direto no período de incubação da doença e do aparecimento dos primeiros sintomas. Com essa informação as entidades de saúde podem entrar em contato com essas pessoas e orientá-las a realizar testes de infecção, ou mesmo permanecer em quarentena.

Porém, como ambos os processos de realização de entrevistas e contatar as pessoas mencionadas pelo paciente dependem de capital humano especializado, em um contexto de explosão de casos essas atividades são negativamente impactadas por depender de capital humano em um momento de infecção em massa.

Com essa situação, soluções tecnológicas que pudessem realizar essa atividade de descoberta e contato com possíveis infectados se tornaram uma alternativa para a mitigação do contágio, países como Coreia do Sul, Singapura e Alemanha utilizaram ferramentas de rastreamento digital de contatos(34), assim como *Apple* e *Google* também lançaram em conjunto um protocolo nativo(28) para essa finalidade e também

diversas iniciativas *open source* surgiram com esse mesmo objetivo.

Servindo como base para a implementação a ser apresentada adiante nesse trabalho, o protocolo *Covid Community Alert*(11) foi uma das diversas soluções de código aberto que surgiram no período e que buscava oferecer uma ferramenta de garantisse o anonimato e uma detecção de qualidade. O projeto teve suporte de desenvolvedores e pesquisadores pelo mundo, inclusive com o apoio de integrantes da comunidade acadêmica da UFSC.

O protocolo tem como principal objetivo garantir que o usuário tenha em mãos uma ferramenta que realiza o monitoramento de contatos com outras pessoas de maneira anônima. Para isso usa o *Bluetooth Low Energy* - *Bluetooth* de baixa energia (BLE) para realizar a varredura das proximidades do usuário e oferece compatibilidade entre Android e iOS, dois sistemas operacionais que embora possuam restrições de conectividade para transferência de arquivos(26) conseguem se comunicar para esse fim específico.

O *Covid Community Alert* permite que o usuário deixe o aplicativo do protocolo rodando em segundo plano, quando um contato acontece, ou seja, quando o protocolo detecta outro dispositivo rodando o mesmo protocolo essa leitura é enviada para um banco de dados seguro. Caso em 14 dias algum dos usuários notificar suspeita ou confirmação de infecção, todos os que foram expostos, portanto, estiveram nas proximidades do usuário infectado, recebem uma notificação indicando quais são as ações necessárias a se tomar.

A equipe que desenvolvia o protocolo, já contava com o protocolo finalizado e trabalhava nos aplicativos que dariam suporte ele, porém o desenvolvimento foi interrompido por falta de recursos e voluntários para manter a iniciativa.

A partir disso, os integrantes da UFSC que faziam parte da equipe do projeto repensaram a solução para uso exclusivo nas imediações do Campus Reitor João David Ferreira Lima visando oferecer a administração e comunidade universitária uma ferramenta de prevenção e controle de possíveis contaminações e assim surgiu o projeto CheckIn@UFSC.

Desde então, foram desenvolvidos aplicativo Android baseado no código-fonte da equipe do protocolo e uma interface web para a visualização dos dados gerados pelos usuários. Dessa forma, o presente trabalho pretende apresentar o desenvolvimento de aplicativo nativo para o sistema operacional iOS, completando assim o ecossistema necessário para cobrir uma maior população de usuários.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Realizar o desenvolvimento e testes do aplicativo do CheckIn@UFSC para o sistema operacional iOS.

1.1.2 Objetivos Específicos

a) Realização da fundamentação teórica e estado da arte do rastreamento de contatos digital;

b) Investigação da viabilidade técnica do uso do framework Flutter para o desenvolvimento de um aplicativo que realize a detecção de interação em primeiro e segundo plano;

c) Desenvolvimento do aplicativo, com base nos resultados encontrados na etapa anterior;

d) Realização de testes com diferentes dispositivos;

e) Disponibilização do aplicativo para a comunidade acadêmica;

1.2 JUSTIFICATIVA

O aplicativo tem como por objetivo oferecer à comunidade acadêmica como ferramenta de suporte à prevenção, pois ao informar os usuários que uma exposição ocorreu, isso pode levar a uma realização de testes antecipada, como também a realização de uma quarentena voluntária detectada uma infecção.

Além disso, o aplicativo, como todo o sistema do CheckIn@UFSC servirá à UFSC como ferramenta de observação de dados, permitindo assim, que medidas de contenção a novos surtos sejam tomadas com base nas tendências que se apresentarem.

1.2.1 Metodologia

A seguir é descrita a metodologia usada para a execução das atividades relacionadas aos objetivos específicos citados anteriormente:

a) Levantamento de trabalhos e publicações de cunho científico relacionadas ao tema do projeto

b) Levantamento de informações sobre o protocolo *Covid Community Alert* que deem suporte ao desenvolvimento do aplicativo iOS

c) Levantamento de requisitos utilizando o aplicativo Android desenvolvido pela equipe do CheckIn@UFSC e execução do desenvolvimento do aplicativo iOS.

d) Execução de testes de comunicação em ambiente controlado.

e) Submissão do aplicativo para revisão e publicação na respectiva loja.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 *BLUETOOTH LOW ENERGY*

O BLE é uma variação do *Bluetooth* clássico e visa ser uma alternativa que utiliza menos energia para a sua operação, além de suportar diferentes faixas de frequência e topologias de conexão, além de ponto a ponto suportada pela versão clássica. E também suporta a criação de redes de larga escala de dispositivos(30).

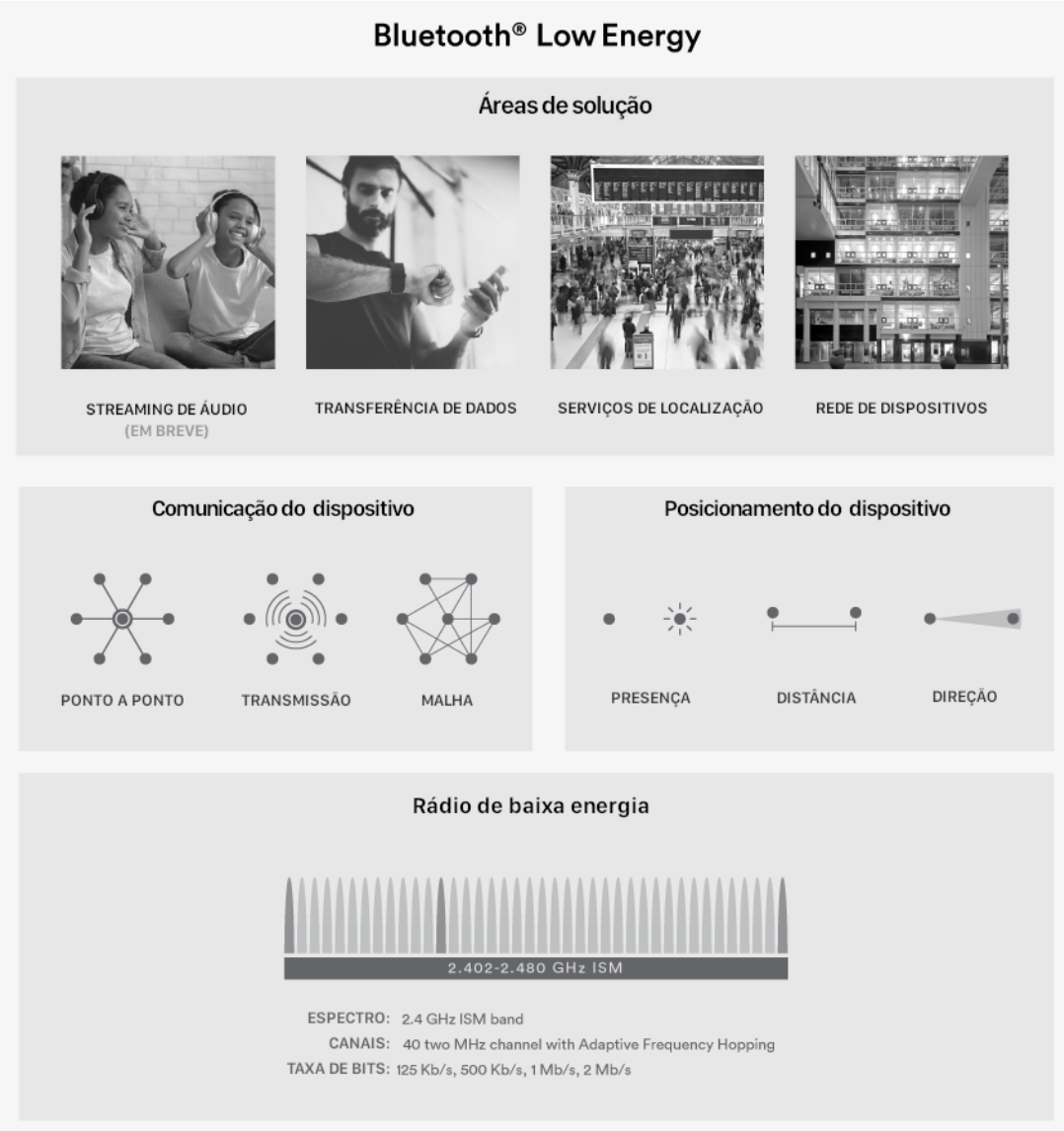
Figura 1 – Infográfico sobre o *Bluetooth* clássico.

Fonte: Autor (2022) adaptado de *Bluetooth Special Interest*(2022).

Conforme o *Bluetooth Special Interest Group* a versão clássica tem como principal aplicação o *streaming* sem fio de áudio, como também permitir a conexão entre

dispositivos sem fio como fones, caixas de som e sistemas de entretenimento automotivo (30).

Figura 2 – Infográfico sobre o *Bluetooth Low Energy*



Fonte: Autor (2022) adaptado de *Bluetooth Special Interest*(2022).

E embora herde características em comum com a versão clássica, o BLE não é compatível com a versão tradicional do Bluetooth(19).

2.2 *BEACONS*

Beacons são dispositivos que implementam o BLE sendo usados para transmitir um identificador único para dispositivos em seus arredores(7). Essa tecnologia começou a ganhar popularidade em meados de 2013 quando a Apple fez o lançamento do iBeacon, versão voltada apenas para o iPhone e para permitir que os *smartphones* da

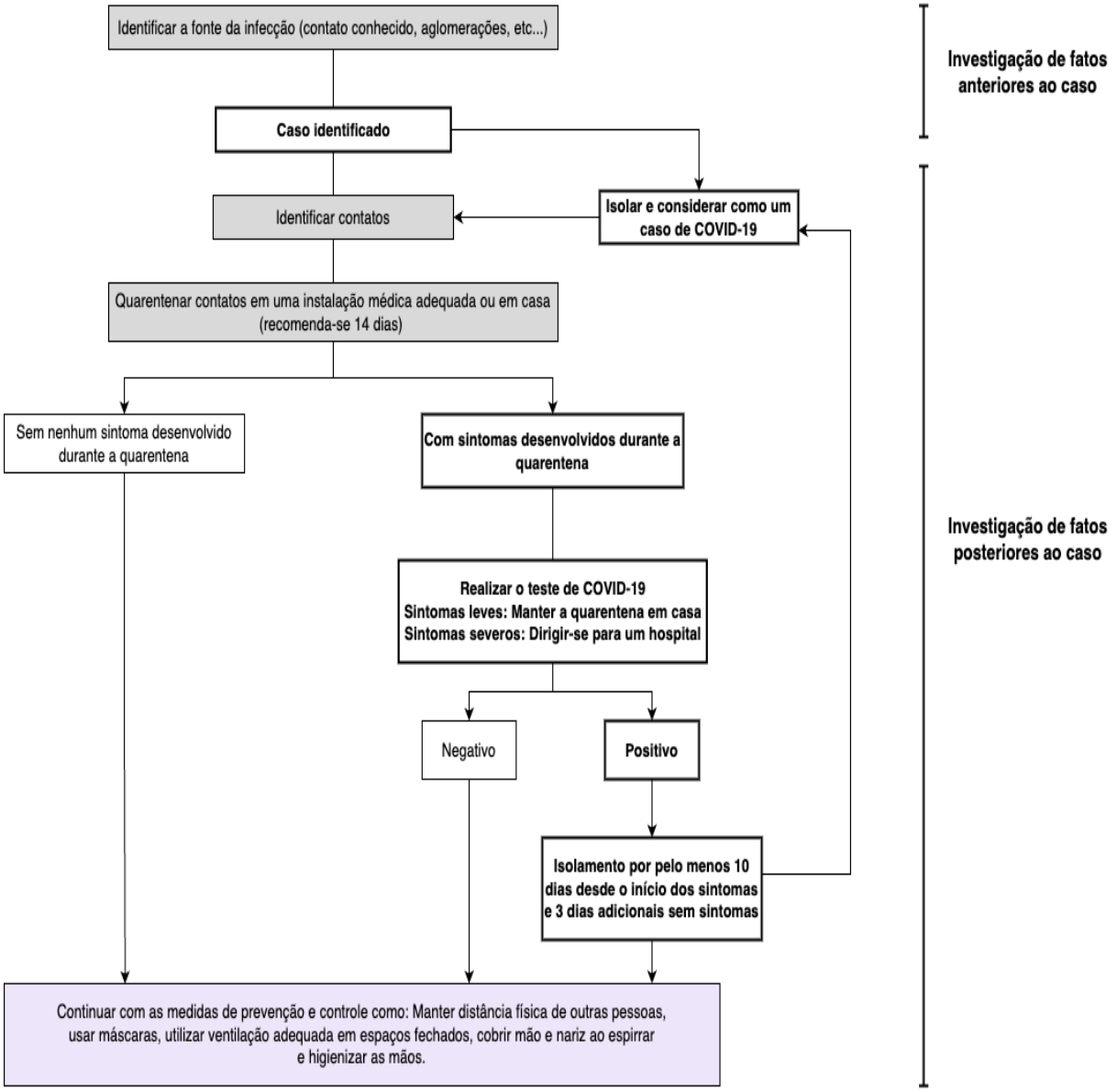
marca pudessem perceber quando o usuário entrasse em uma área coberta por um dispositivo beacon compatível(18). Por exemplo, em uma loja de departamentos, caso um cliente se aproximasse de uma área coberta por um iBeacon, ele receberia uma notificação relacionada sobre a seção onde está, ou mesmo uma promoção.

Após 2013, surgiram outras versões de *Beacons*(32) como: AltBeacon, Eddystone e GeoBeacon. Todas elas *open source* e possibilitando que dispositivos Android também oferecessem a mesma experiência imersiva de localização dos iPhones.

2.3 RASTREAMENTO DE CONTATOS

O rastreamento de contatos no contexto desse trabalho se refere ao processo de levantamento de prováveis infectados que tiveram contato com um paciente, que foi testado positivo para COVID-19. No diagrama a seguir, publicado pela OMS, são indicadas as etapas desse processo:

Figura 3 – Cadeia de eventos para rastreamento, monitoramento e atendimento de contatos de casos prováveis e confirmados de COVID-19



Fonte: Autor (2022) adaptado de OMS(10)(2021).

É possível constatar que para que esses passos sejam seguidos é necessária uma equipe especializada, fato endossado pela organização no documento *Contact tracing in the context of COVID-19*(10) e que traz orientações voltadas para governos e entidades de saúde sobre como esse processo deve acontecer.

Porém, em um cenário de alta transmissibilidade e baixa testagem, essa estratégia se torna frágil por depender exclusivamente de capital humano para a coleta e tratamento das informações dos pacientes positivos para a doença.

Buscando solucionar esse problema, que surge o rastreamento de contatos

digital, que ao invés de depender de uma equipe de pessoas para entrevistar a pessoa infectada pelo vírus, usa dados obtidos pelos *smartphones* da população, que registram automaticamente contatos com outros usuários e os notifica caso uma exposição tenha acontecido.

2.4 *COVID COMMUNITY ALERT*

O *Covid Community Alert* é um protocolo de rastreamento de contatos criado por uma iniciativa *open source* cujo objetivo é reduzir a proliferação do vírus da COVID-19, alertando pessoas em risco e fornecendo instruções para reduzir o contágio.

Suas especificações técnicas(29) preveem:

1. Infraestrutura: Responsável por manter os serviços do projeto online;
2. *Backend*: Responsável por processar as requisições e fornecer as leituras dos dados salvos;
3. Protocolo de *roaming*: Responsável por identificar os contatos que cada usuário teve, usa o BLE para realizar a estimação da exposição do usuário;
4. Aplicativos iOS e Android do usuário (CovidApp): Responsável por monitorar os contatos com outros usuários;
5. Aplicativos iOS e Android do médico (CovidDoc): Responsável por confirmar um usuário como infectado ou suspeito;
6. Painel interno de Epidemiologistas: Permite que os Epidemiologistas definam como e quais serão as notificações de exposição e contágio a serem enviadas para os usuários.

Para realizar o monitoramento de contatos foi usado o BLE fazendo com que os dispositivos se comportassem como *beacons* por conta das restrições de conexão entre dispositivos iOS e Android utilizando o Bluetooth clássico. Na prática, os dispositivos usam o bluetooth intermitentemente, sendo que em alguns intervalos fazem o papel de *beacons* divulgando para os dispositivos próximos o seu identificador único e em outros fazem o papel de detecção dos beacons próximos.

A documentação(29) detalha o funcionamento da aplicação da seguinte maneira:

1. Cada dispositivo possui um identificador único que pode ser lido a distância por meio do BLE.
2. O protocolo realiza varreduras continuas das proximidades sem a necessidade de interações entre o aplicativo e os usuários e coleta os identificadores dos

celulares próximos ao usuário e os armazena em um banco de dados seguro centralizado em nuvem.

3. Autoridades médicas atualizam o banco de dados com os ids dos dispositivos que confirmaram uma infecção.

4. E se um usuário esteve próximo de um usuário infectado durante os últimos 14 dias, ele será notificado e receberá instruções sobre quais ações ele deve tomar a depender da exposição estimada com o vírus.

2.5 CHECKIN@UFSC

O CheckIn@UFSC é um projeto que se derivou da solução criada pelo *Covid Community Alert* com o objetivo produzir uma ferramenta voltada apenas para os campi da UFSC, sendo num primeiro momento pensada para atuar apenas no Campus Reitor João David Ferreira Lima em Florianópolis.

Diante disso, todo o código existente do projeto foi adaptado para a realidade da Universidade, o que alterou sua estrutura, removendo o aplicativo CovidDoc que era voltado para uso médico e que confirmava um caso suspeito ou positivo e integrando essa função ao aplicativo do usuário comum - o CovidApp.

Além disso, diferentemente do projeto inicial e por se tratar de uma solução voltada exclusivamente para a comunidade da UFSC, foi integrada ao aplicativo uma etapa de autenticação com o sistema da UFSC(6) para restringir o acesso às funções de rastreamento e autodiagnóstico apenas para os integrantes da comunidade acadêmica, como também foi realizada uma restrição de atuação do aplicativo, para que ele colha informações enquanto o usuário estiver no perímetro do campus.

2.6 IOS

O iOS é o sistema operacional para dispositivos móveis exclusivos da empresa Apple Inc., lançado em 2007(20) nos Estados Unidos com o Iphone, o primeiro aparelho lançado no mundo sem teclado de botões e com uma tela sensível ao toque (*touchscreen*).

Atualmente o sistema já está em sua 16^a versão(24) e mantém compatibilidade com 23 modelos de iPhones.

Atualmente para desenvolver um aplicativo iOS é necessário possuir um computador que possua o sistema operacional MacOS instalado, que nesse caso só é possível de ser encontrado em dispositivos vendidos pela própria marca, além disso, a marca possui uma *Integrated development environment* - Ambiente de desenvolvimento integrado (IDE) própria para o desenvolvimento de aplicações para os sistemas operacionais lançados por ela, chamado Xcode.

2.7 FLUTTER

O Flutter é um kit de desenvolvimento de software(17) criado pela Google para o desenvolvimento híbrido de aplicativos, ou seja, com uma base de código é possível criar aplicações para diferentes plataformas como dispositivos móveis, web, *desktops* e incorporados (3).

Lançada em 2015, essa ferramenta permite que o desenvolvedor crie interfaces de usuário e a lógica da aplicação utilizando a linguagem Dart que foi projetada para permitir desenvolvimento rápido de aplicativos para qualquer plataforma.

Além disso, é uma linguagem que possui segurança de tipo e verificação de tipo estático para garantir que os valores dados a uma variável sempre irão ser compatíveis com o tipo da variável(13).

E conforme a documentação da linguagem é possível compilar aplicativos para plataformas nativas, sendo elas: ARM32, ARM64 e x86_64 e para plataforma web por meio do JavaScript.

3 DESENVOLVIMENTO

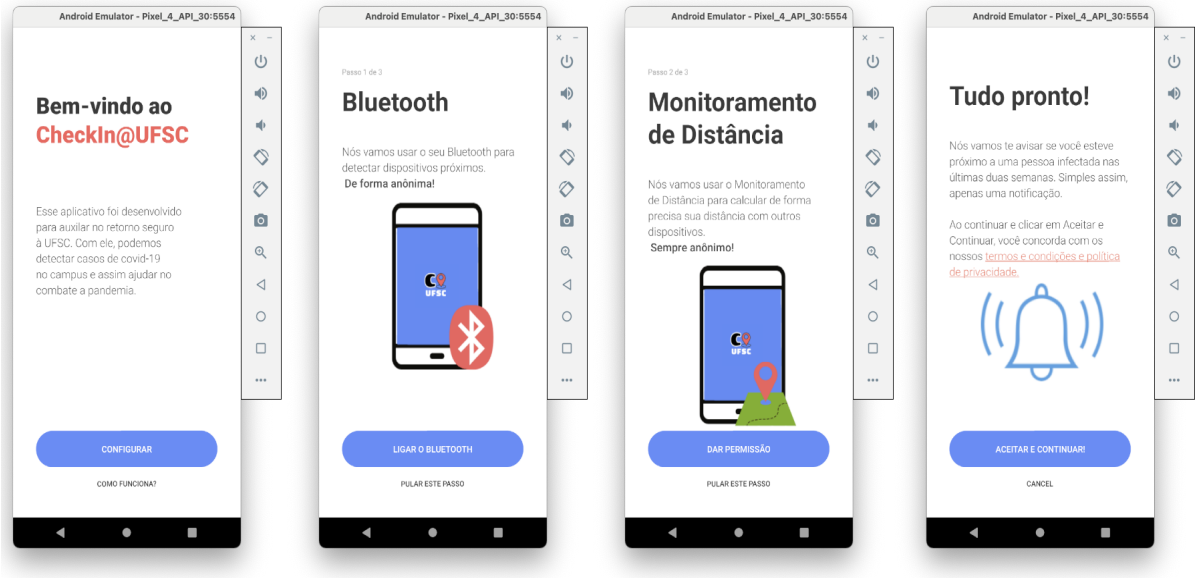
3.1 APLICATIVO ANDROID

Como o projeto já contava com um aplicativo Android desenvolvido, ele foi utilizado como base para o desenvolvimento da aplicação iOS. O aplicativo contava com dois fluxos de interfaces: interfaces de introdução ao aplicativo e as interfaces da área autenticada do aplicativo.

3.1.1 Interfaces de introdução ao aplicativo

As interfaces responsáveis pela introdução ao aplicativo são responsáveis por: validar o usuário por meio de um *captcha*, permitir que o usuário dê as permissões necessárias para o funcionamento do aplicativo como uso do bluetooth e localização e o *login* com o sistema de autenticação centralizada da UFSC.

Figura 4 – Telas de introdução — Android

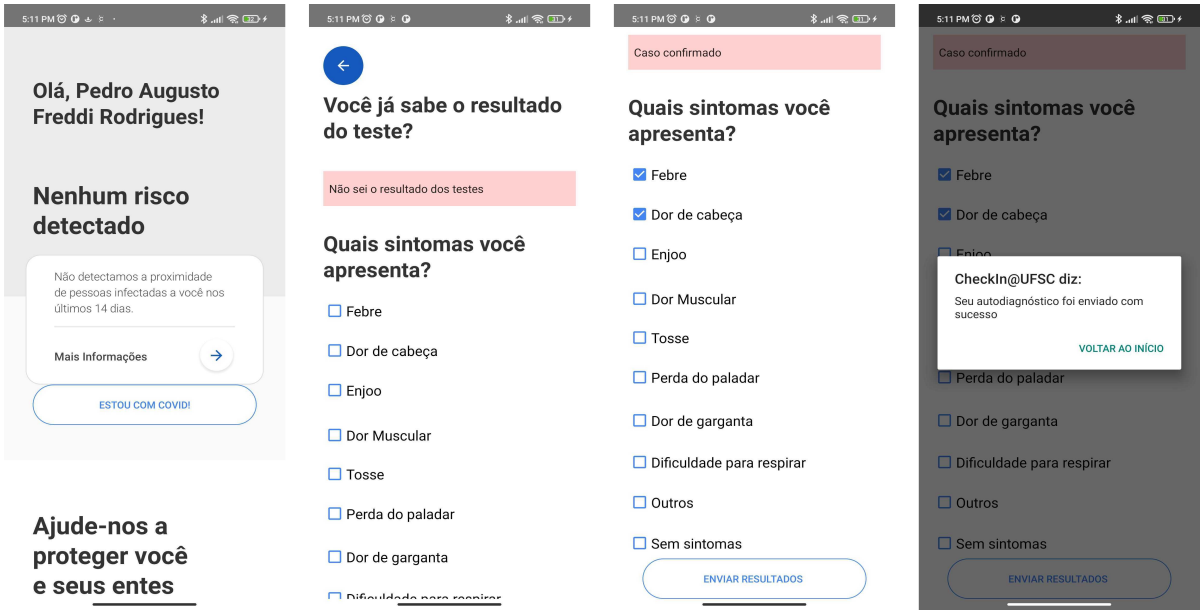


Fonte: Autor (2022)

3.1.2 Interfaces da área autenticada

Já as interfaces relacionadas à área logada, são responsáveis por permitir que o usuário confira as notificações de risco quando detectado e notificar os demais quando possuir sintomas ou testar positivo para a doença.

Figura 5 – Telas da área autenticada - Android



Fonte: Autor (2022)

3.1.3 Requisitos elicitados com base no aplicativo

Com base nas interfaces apresentadas, foram listados os seguintes requisitos do aplicativo:

Quadro 1 – Requisitos elicitados a partir da interação com o aplicativo *Android*.

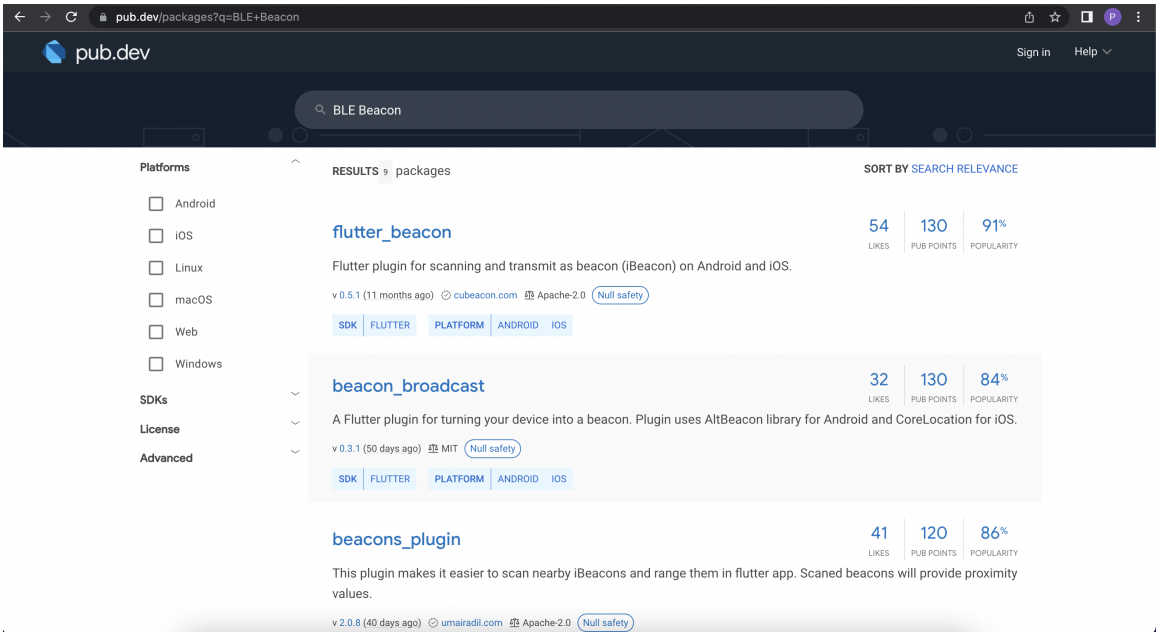
REQ-001	Validar o usuário por meio de *Captcha*
REQ-002	Solicitar a permissão de uso do *bluetooth* ao usuário
REQ-003	Solicitar a permissão de uso da localização ao usuário
REQ-004	Realizar a autenticação com o CAS UFSC
REQ-005	Exibir a situação de exposição do usuário
REQ-006	Permitir que o usuário envie seu autodiagnóstico
REQ-007	Exibir mais informações sobre seu funcionamento
REQ-008	Permitir que o usuário compartilhe o aplicativo para que mais pessoas possam ser protegidas
REQ-009	Oferecer informações adicionais sobre a situação de exposição do usuário
REQ-010	Exibir a tela de abertura com a logo do CheckIn@UFSC
REQ-011	Salvar os dados da sessão do usuário
REQ-012	Exibir notificações de *push* quando o usuário estiver em risco
REQ-013	Ao ser colocado em segundo plano, deve escanear dispositivos e transmitir sua presença para dispositivos nas redondezas
REQ-014	Enviar dados para a infraestrutura do CheckIn@UFSC
REQ-015	Mostrar ao usuário quando há alguma permissão pendente que impeça o funcionamento do aplicativo

Com os requisitos do aplicativo listados, se notou a possibilidade de explorar uma alternativa de desenvolvimento híbrido, ao invés de nativa para o desenvolvimento do aplicativo, visando reduzir as bases de código iOS e Android para apenas uma. Acreditava-se essa opção de desenvolvimento poderia ser vantajosa para o projeto que contava com recursos humanos e financeiros limitados.

3.2 INVESTIGAÇÃO DE VIABILIDADE DO *FRAMEWORK* FLUTTER

Foi realizada uma investigação sobre a viabilidade do uso do framework Flutter para a criação de um aplicativo multiplataforma. Por meio de uma pesquisa com os termos BLE e *beacon* no repositório oficial de pacotes para aplicativos Flutter: pub.dev, foi possível localizar 9 resultados relacionados:

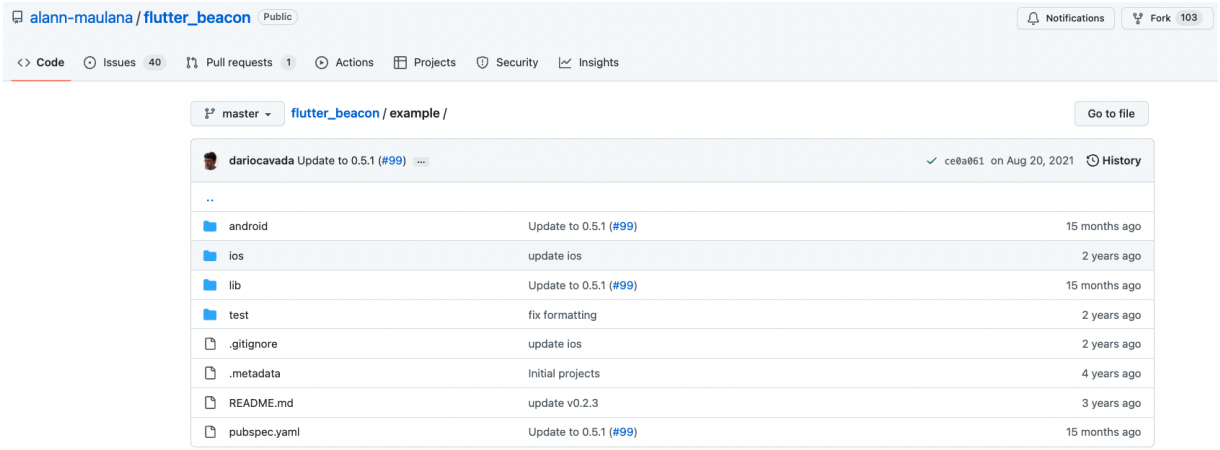
Figura 6 – Resultados retornados para a pesquisa "BLE Beacon".



Fonte: Autor (2022)

A partir dos resultados encontrados as bibliotecas foram testadas seguindo sua popularidade, sendo esse um índice gerado pela própria plataforma pub.dev. Portanto, a primeira biblioteca a ser testada foi a flutter_beacon, que possuía popularidade de 91%. Para a realização do teste foi utilizado um aplicativo de exemplo disponibilizado pelo repositório da biblioteca.

Figura 7 – Aplicativo de exemplo



Fonte: Autor (2022)

Esse aplicativo possui a biblioteca implementada, como também as funções de transmissão e detecção de *beacons* nas proximidades, sendo assim o teste buscava

identificar se os seguintes pontos eram cobertos pela tecnologia:

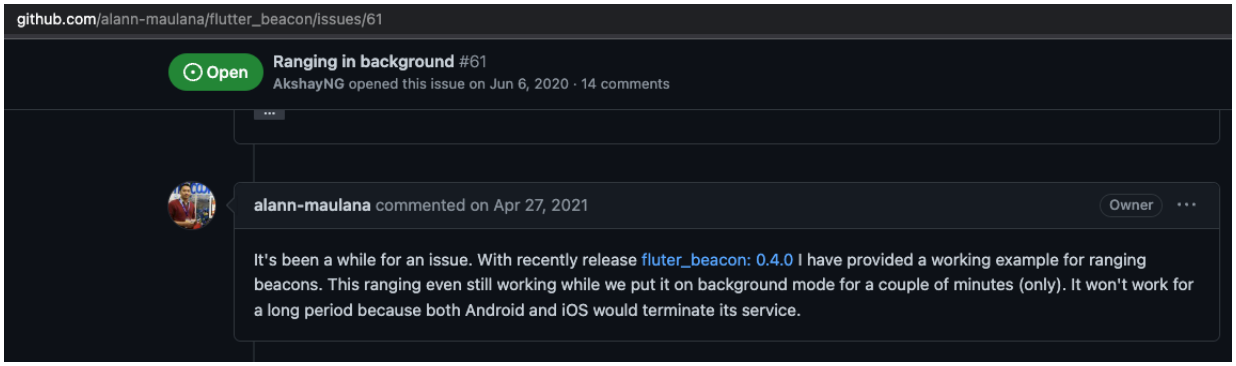
1. Encontrar dispositivos com o mesmo sistema operacional;
2. Encontrar dispositivos com sistemas operacionais diferentes;
3. Manter a detecção e transmissão em segundo plano em dispositivos iOS;
4. Manter a detecção e transmissão em segundo plano em dispositivos Android.

Uma vez que todos os pontos fossem contemplados, se iniciaria o desenvolvimento do aplicativo em Flutter por ter se provado a viabilidade de uso da tecnologia. Para a realização dos testes foram utilizados 4 dispositivos, sendo 2 deles Android e 2 iOS nas seguintes configurações:

Nome	Sistema Operacional	Versão	Memória Interna	Memória RAM
iPhone SE (2016)	iOS	13	16GB	4GB
iPhone 7	iOS	13	32GB	4GB
Poco X3 NFC	Android	11	128GB	8GB
Xiaomi Mi Note 8	Android	10	64GB	4GB

Ao testar a biblioteca foi possível identificar que os pontos 1 e 2 foram contemplados, porém, não foi possível manter o aplicativo funcionando em segundo plano em ambos sistemas operacionais. Ao investigar as causas desse comportamento foi localizada na discussão de número 61 do repositório, intitulada *"Ranging in background"* na qual o autor da biblioteca, Alann Maulana, declara que não era possível manter o aplicativo em segundo plano por restrições dos sistemas operacionais.

Figura 8 – Comentário do autor da biblioteca flutter_beacon sobre o funcionamento em segundo plano.



Fonte: Autor (2022)

Quanto ao teste realizado com outras bibliotecas, o resultado obtido foi o mesmo encontrado anteriormente. Por conta disso, assumiu-se que esse comportamento

ocorre por se tratar de um aplicativo multiplataforma, visto que esse tipo de tecnologia é recente.

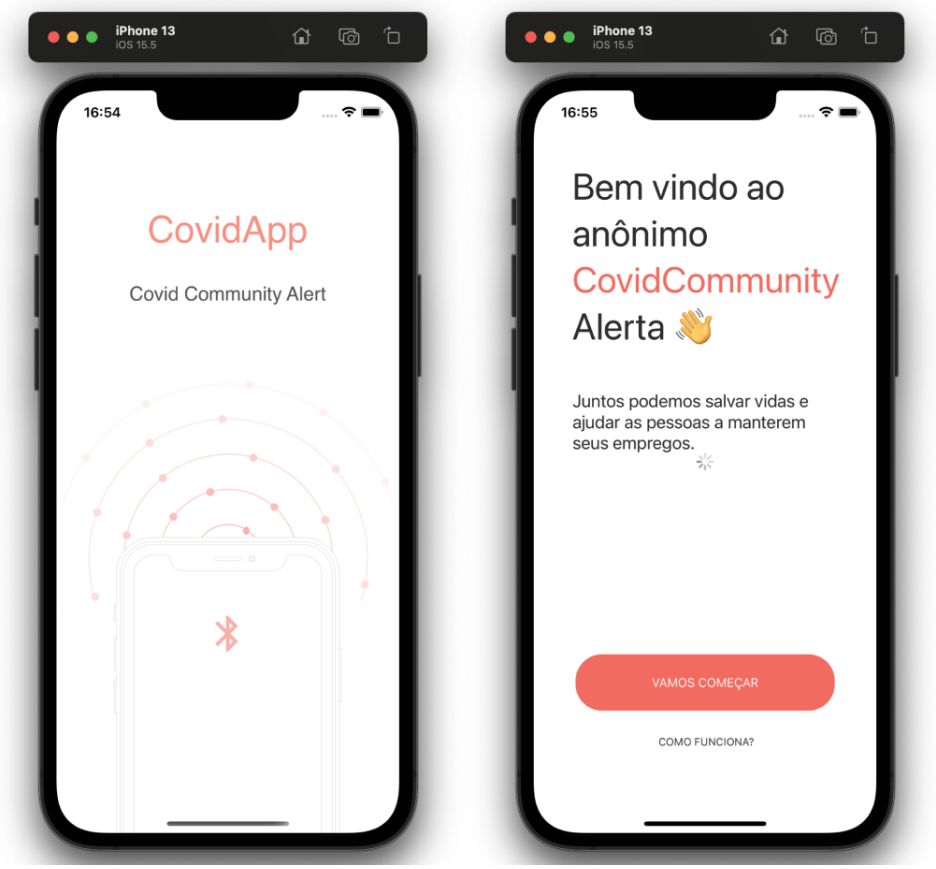
Dessa forma foi descartada a opção de desenvolvimento com Flutter e retomado o desenvolvimento nativo utilizando a linguagem Swift, visto que a equipe do projeto do *Covid Community Alert* já tinha um aplicativo iOS desenvolvido e que conseguia manter o rastreamento ativo em segundo plano.

3.3 DESENVOLVIMENTO NATIVO DO APLICATIVO IOS

Como a equipe do projeto *Covid Community Alert* já possuía um aplicativo iOS em andamento, optou-se por utilizar o código existente, assim como já havia sido feito com o aplicativo Android, que utiliza a base de código criado pela mesma equipe.

Ao compilar o código foram encontrados os mesmos dois fluxos de interfaces presentes no aplicativo Android, mas com algumas diferenças visuais como cores e imagens conforme mostram as imagens abaixo:

Figura 9 – CovidApp iOS — Telas de abertura e Boas-vindas



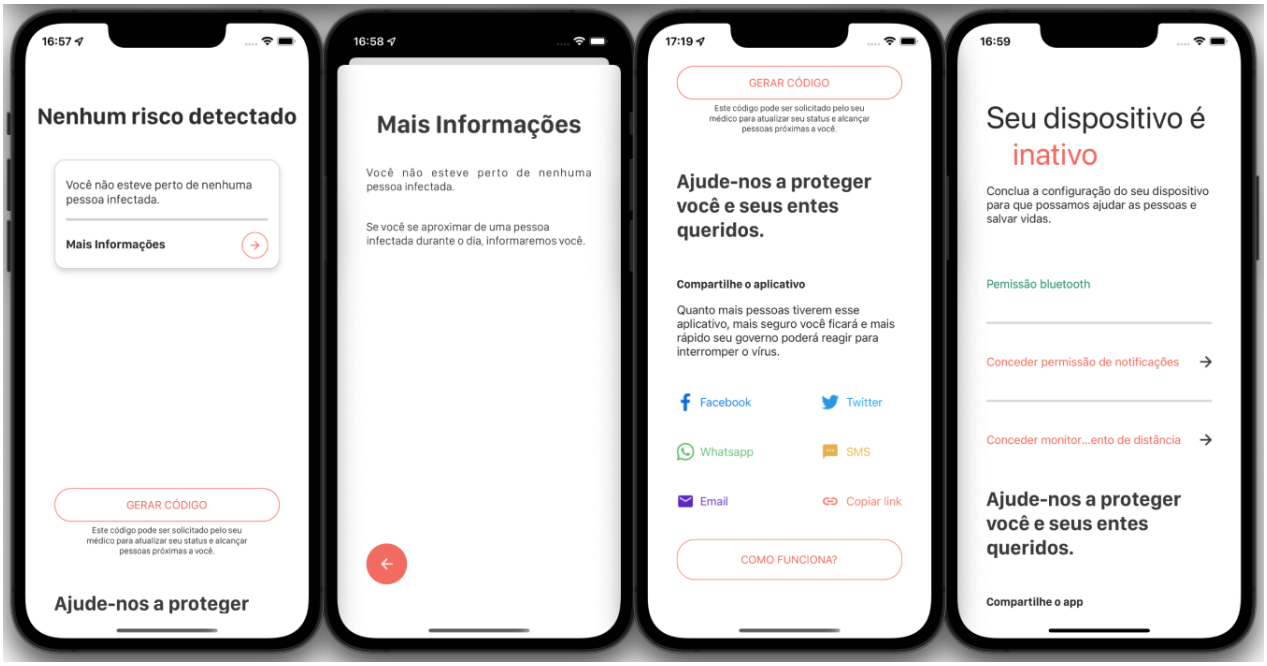
Fonte: Autor (2022)

Figura 10 – CovidApp iOS — Telas de permissões

Fonte: Autor (2022)

Após o fluxo de Boas-vindas, o aplicativo conta com interfaces que comunicam aos usuários se os mesmos estão em risco, permitem que o usuário saiba se há permissões pendentes e também possibilitam o compartilhamento do aplicativo por meio das redes sociais.

Figura 11 – Fluxo principal — Interfaces de usuário



Fonte: Autor (2022)

Como esse aplicativo havia sido criado e integrado para se comunicar com a infraestrutura do projeto *Covid Community Alert* alguns dos requisitos listados anteriormente já haviam sido implementados, mas novos tiveram que ser adicionados para que ele pudesse ser operacional no contexto do projeto CheckIn@UFSC.

Quadro 2 – Requisitos elicitados após com o app iOS da *Covid Community Alert*

REQ-001	Permitir a autenticação com o CAS UFSC
REQ-002	Mostrar a situação de exposição do usuário
REQ-003	Permitir que o usuário envie seu autodiagnóstico
REQ-004	Exibir a tela de abertura com a logo do CheckIn@UFSC
REQ-005	Exibir notificações de *push* quando o usuário estiver em risco
REQ-006	Ao ser colocado em segundo plano, deve escanear dispositivos e transmitir sua presença para dispositivos nas redondezas
REQ-007	Enviar dados para a infraestrutura do CheckIn@UFSC
REQ-008	Exibir nome e ícone do CheckIn@UFSC
REQ-009	Exibir interfaces localizadas para os idiomas português e inglês
REQ-010	O código-fonte deve estar documentado para permitir que novos desenvolvedores entendam a função que cada classe desempenha

E com base nos requisitos apresentados na tabela 2 foram criadas tarefas de desenvolvimento para guiar a implementação dos mesmos. No quadro abaixo são listados as tarefas que foram encontradas:

Quadro 3 – Tarefas encontradas com base nos requisitos listados.

CHK-001 — Atualizar o README do projeto	Inserir informações e imagens atualizadas sobre projeto, os passos necessários para colocar o aplicativo em funcionamento e outras informações de desenvolvimento que sejam relevantes
CHK-002 — Atualizar as imagens do aplicativo para as correspondentes ao CheckIn@UFSC	Substituir os arquivos de imagem referentes à identidade visual do Covid Community Alert para os do CheckIn@UFSC
CHK-003 — Criar documentação da classe IBeaconManager	Criar documentação descrevendo o que cada componente dessa classe realiza.
CHK-004 — Criar documentação da classe StorageManager	Criar documentação descrevendo o que cada componente dessa classe realiza.
CHK-005 — Criar documentação da classe BackgroundManager	Criar documentação descrevendo o que cada componente dessa classe realiza.
CHK-006 — Criar documentação da classe BluetoothManager	Criar documentação descrevendo o que cada componente dessa classe realiza.
CHK-007 — Criar a interface de autodiagnóstico	Implementar a interface de autodiagnóstico seguindo a implementação realizada no aplicativo *Android*
CHK-008 — Integração do autodiagnóstico	Implementar a integração do autodiagnóstico com o *backend* do CheckIn@UFSC
CHK-009 — Criar a interface de login com a UFSC	Implementar a interface de autenticação com os sistemas da Universidade
CHK-009 — Integrar a interface de login com a UFSC	Integrar a interface de autenticação com os sistemas da Universidade
CHK-011 — Ajustar a localização do aplicativo	Realizar ajustes na localização para que ela apresente o idioma correto com base na região do dispositivo

3.3.1 Documentação do projeto

3.3.1.1 Configurações gerais do projeto

O repositório do projeto possuía um arquivo vazio de README, portanto, foram adicionadas a eles informações úteis para manter documentadas algumas informações básicas a respeito do aplicativo, além de instruções de como instalar as dependências usadas.

Figura 12 – Arquivo de README do projeto.

README.md

CheckIn@UFSC iOS

O CheckIn@UFSC foi criado com o objetivo de ser uma ferramenta para auxiliar na diminuição do contágio do COVID-19. Ele é um aplicativo que realiza o rastreamento anônimo de interações com outros usuários, permite que cada usuário faça seu autodiagnóstico, indicando se o indíviduo possui sintomas ou foi testado como positivo para a doença e notifica os demais usuários que tiveram em contato com uma pessoa infectada, fornecendo instruções de como proceder.

Funções

- [x] Autenticação com o serviço de autenticação centralizada da UFSC (CAS)
- [x] Autodiagnóstico de sintomas de COVID-19
- [x] Exibição do status de exposição do usuário
- [x] Rastremento anônimo de interações com outros usuários em segundo plano
- [x] Notificações de push

Requisitos

- iOS 12.1+
- Xcode 10.1+

Dependências

Esse projeto utiliza **SPM (Swift Package Manager)** e **cocoapods** para o gerenciamento de dependências do projeto.

Para instalar as dependências relacionadas ao **cocoapods** é necessário acessar via terminal o diretório `CheckInUFSC` e executar o seguinte comando `pod install`

Já as dependências gerenciadas pelo **SPM** são instaladas automáticamente quando o projeto é aberto pela primeira vez.

Fonte: Autor (2022)

3.3.1.2 Classes

Durante a investigação da base de código foi possível identificar dois tipos de classe que são vitais para o funcionamento do aplicativo: classes *managers* e *view controllers*. Ambas foram documentadas usando UML sendo adicionadas ao Anexo B.

As classes *view controllers* são as responsáveis por exibir e gerenciar as interfaces do aplicativo.

Já as classes *Managers*, são as principais responsáveis por armazenar a lógica de negócio de operações como armazenamento local de leituras, intermitência entre

as funções de rastreamento e transmissão de presença e funcionamento em segundo plano.

No entanto, essas classes não contavam com nenhuma documentação a respeito do papel que elas e seus respectivos métodos desempenham.

Para tornar mais claras as funções de cada uma dessas classes e seus métodos foi utilizada a notação de documentação provida pelo *Xcode* (IDE de desenvolvimento iOS criada pela Apple) como mostra a figura abaixo:

Figura 13 – Exemplo de comentário usado para documentar as classes do aplicativo.

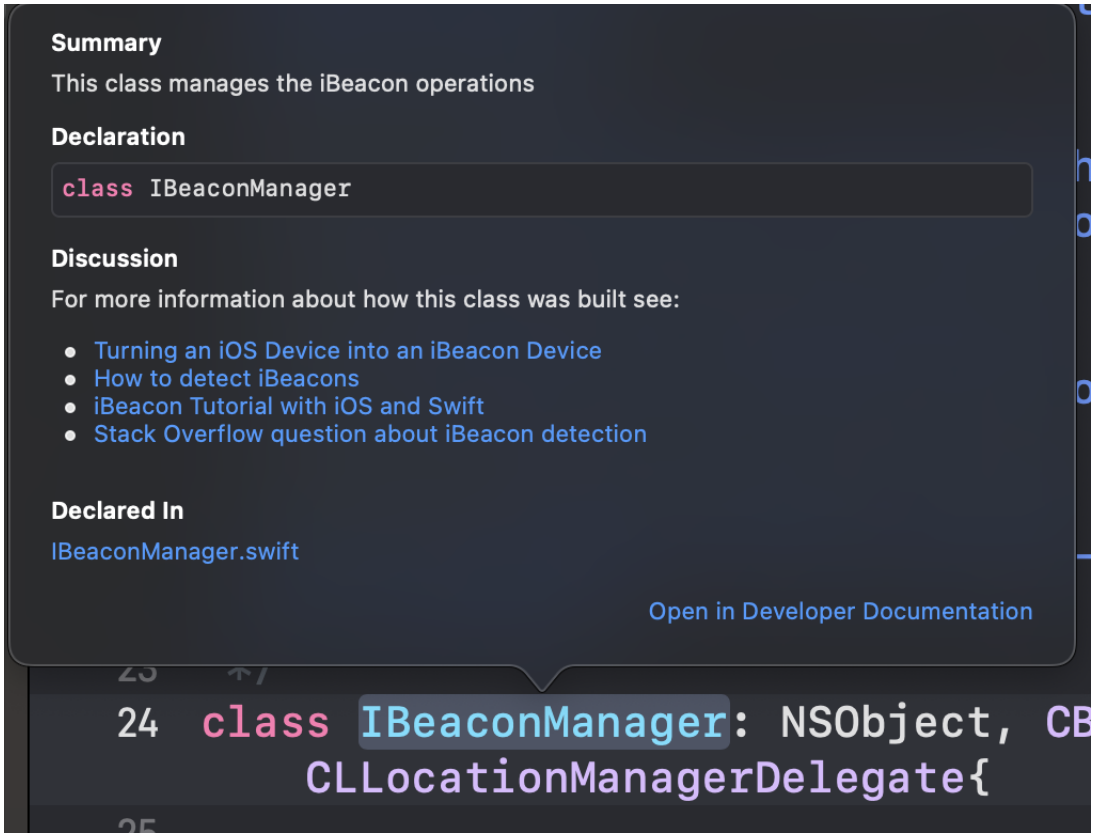
```
1 /**
2     This class manages the iBeacon operations
3
4     For more information about how this class was built see:
5     - [Turning an iOS Device into an iBeacon Device](https://
      developer.apple.com/documentation/corelocation/
      turning_an_ios_device_into_an_ibeacon_device)
6     - [How to detect iBeacons](https://www.hackingwithswift.com/
      example-code/location/how-to-detect-ibeacons)
7     - [iBeacon Tutorial with iOS and Swift](https://www.
      raywenderlich.com/632-ibeacon-tutorial-with-ios-and-swift)
8     - [Stack Overflow question about iBeacon detection](https://
      stackoverflow.com/questions/39977251/a-simple-code-to-detect-any
      -beacon-in-swift/46448986)
9  */
```

Fonte: Autor (2022)

A figura acima mostra a documentação da classe *IBeaconManager* responsável por gerenciar as operações de leitura e transmissão de beacons próximos e detalha quais foram as fontes utilizadas para sua criação.

Além disso, exemplo mostra que é possível utilizar a linguagem de marcação *Markdown* em associação ao texto do comentário. Com essa estrutura a IDE permite que a documentação seja visualizada como uma caixa de diálogo como mostra a figura abaixo:

Figura 14 – Caixa de diálogo.



Fonte: Autor (2022)

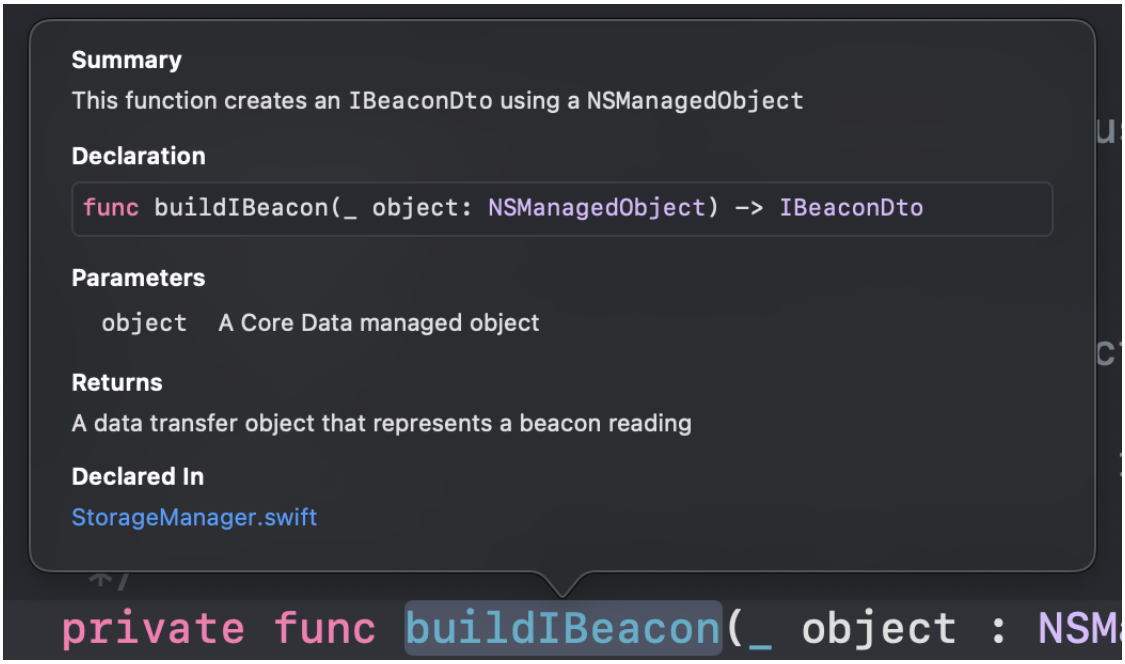
Os comentários que documentam métodos seguem uma estrutura similar, mas também permitem a adição de descrições para parâmetros e valor retornado como mostram as figuras abaixo:

Figura 15 – Exemplo de comentário usado para documentar as funções do aplicativo

```
1 /**
2       This function creates an ‘IBeaconDto‘ using a ‘NSManagedObject
      ‘
3
4       - parameters:
5           - object: A Core Data managed object
6
7       - Returns: A data transfer object that represents a beacon
      reading
8 */
```

Fonte: Autor (2022)

Figura 16 – Caixa de diálogo.



Fonte: Autor (2022)

Ao seguir essa notação é possível gerar a documentação do projeto pela própria IDE, dessa forma as classes e métodos que seguem essa notação são incluídos automaticamente.

3.3.2 Atualização da estrutura do aplicativo

Durante a investigação da base de código foi possível constatar que algumas refatorações poderiam ser realizadas no código, sendo elas: a da camada de comuni-

cação, a das interfaces e localização do aplicativo e a da navegação entre telas.

3.3.2.1 Refatoração da camada de comunicação

O aplicativo se comunica com a infraestrutura do projeto por meio do protocolo http. Porém, a implementação encontrada no projeto era feita de duas maneiras distintas: em uma parte do projeto era usado o sistema de carregamento de URL(33) e em outra era usada uma biblioteca de terceiros.

O princípio DRY ("Don't Repeat Yourself" — "Não se repita" em tradução livre), apresentado no livro "O programador pragmático" (21) que estabelece que duplicações em código prejudicam a manutenção do código por manterem *bugs* às vezes idênticos em diferentes partes da aplicação. Portanto, a partir dessa premissa de desenvolvimento de *software* a implementação foi considerada problemática para a saúde do projeto, sendo decidida sua refatoração.

Com isso, a primeira decisão tomada foi a uniformizar a camada de comunicação para ela utilizar apenas a biblioteca que estava sendo usada, pois ela possuía uma *API* que facilitava a refatoração.

Em seguida foi criada uma estrutura utilizando o conceito de abstração para padronizar as requisições, por meio de um protocolo chamado *APIRequestable*. Na linguagem *Swift* protocolos podem definir métodos e propriedades para classes e estruturas e enumerações que os implementam, funcionando de maneira similar a Interfaces, comumente utilizadas nas linguagens Java e C#.

Figura 17 – Protocolo *APIRequestable*

```
1  protocol APIRequestable {
2      associatedtype APIResponse: Decodable
3
4      var dateDecodingStrategy: JSONDecoder.DateDecodingStrategy {
       get }
5      var headers: HTTPHeaders? { get }
6      var method: HTTPMethod { get }
7      var parameters: Parameters? { get }
8      var url: String { get set }
9      var interceptor: RequestInterceptor? { get set }
10
11     func convertResponse(from data: Decodable) -> APIResponse?
12     func request(completion: @escaping (APIResult<APIResponse>) ->
       Void) -> Request?
13 }
```

Fonte: Autor (2022)

A figura 17 mostra que o protocolo define propriedades básicas das requisições,

sendo elas:

- ***Headers:*** Ou cabeçalhos HTTP(8) definem metadados que podem dar mais informações ao servidor sobre o cliente ou sobre o recurso(23) a ser requisitado pelo cliente.
- ***Method:*** Define o tipo de método http usado pela requisição (GET, POST, PATCH, HEAD, OPTIONS, DELETE, CONNECT, PUT, TRACE)(16)
- ***Parameters:*** Assim como os cabeçalhos, os parâmetros permitem definir mais detalhes acerca da requisição a ser feita, porém, podem ser inseridos diretamente na URL ou no corpo da requisição(31).
- **URL:** *Uniform Resource Locator* - Localizador Uniforme de Recursos (URL) define o endereço web que será utilizado para realizar a requisição HTTP.
- ***Interceptor:*** *RequestInterceptor*(1) é um protocolo da biblioteca *Alamofire* e possibilita que uma requisição possar ser interceptada antes que seja enviada, ou repetida caso seja retornado um erro pelo servidor.
- **APIResponse:** É um *associatedtype* ou tipo associado, e permite que um mesmo protocolo seja generalizado quando aplicado em diferentes casos.
  - No entanto, nessa implementação o tipo associado foi usado para renomear o protocolo *Decodable* de forma que esse tipo se tornasse sintaticamente significativo ao contexto em que ele é aplicado.
  - Esse protocolo decodificar dados provenientes de fontes externas, nesse caso dados enviados pelo servidor em JavaScript Object Notation - Notação de Objetos JavaScript (JSON) e que serão decodificados para os tipos definidos pela linguagem *Swift*.
- ***DateDecodingStrategy*:** Essa propriedade indica qual o formato das datas retornadas pelo servidor.
  - Por exemplo, o servidor pode enviar um carimbo de data em milissegundos ou uma *string* formatada conforme a ISO-8601. E independente do tipo que esse dado assume no servidor, sua conversão será para o tipo *Date* do *Swift*.
  - É a única propriedade computada do protocolo, ou seja, ela é definida por quem implementa, não sendo possíveis alterações externas.

Já em relação às funções definidas pelo protocolo:

- ***ConvertResponse:*** Será a função responsável por converter os recursos enviados pelo servidor.

- ***Request:*** Será a função responsável realizar a requisição de recursos ao servidor.

Em seguida, para definir a implementação das funções e também da propriedade *dateDecodingStrategy* foi criada uma extensão(15) do protocolo centralizando a implementação no arquivo **APIRequestable.swift**.

Isso permitiu que fossem criadas classes para armazenar requisições afins, como, por exemplo: *AuthManager* que armazena requisições relacionadas a autenticação do usuário, ou mesmo a classe *InteractionManager* responsável pelas chamadas de API relacionadas ao escaneamento de usuários próximos.

Figura 18 – Classe *SelfDiagnosisManager e um exemplo de sua chamada de função*

```
1  class SelfDiagnosisManager {
2      struct SendDiagnosis: APIRequestable {
3          typealias APIResponse = APIResponseEmpty
4          var method: HTTPMethod = .put
5          var parameters: Parameters?
6          var url: String = Constants.Endpoints.selfDiagnosis
7          var headers: HTTPHeaders?
8          var interceptor: Alamofire.RequestInterceptor?
9
10         init(_ params: SymptomsParameters) {
11             parameters = params.asParameters
12             headers = [
13                 "Authorization": "Bearer " + (StorageManager.shared
       .getToken() ?? "")
14             ]
15         }
16     }
17
18     // Exemplo de chamada de função que realiza a requisição
19     SelfDiagnosisManager.SendDiagnosis(SymptomsParameters(...)).
       request { result in
20         switch result {
21         case .failure(let error):
22             // Em caso de falha...
23         case .success(_):
24             // Em caso de sucesso...
25         }
26     }
27 }
```

Fonte: Autor (2022)

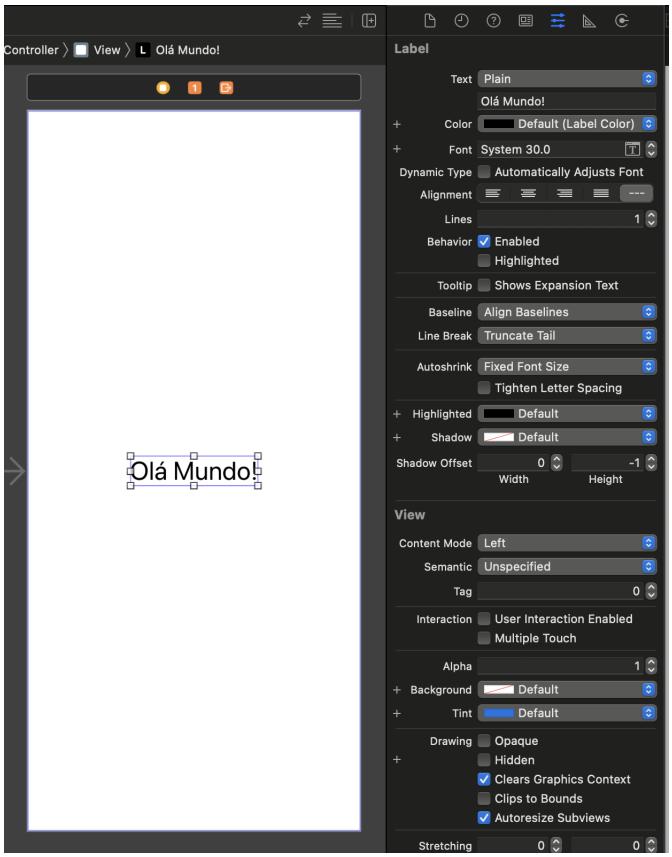
A figura acima exemplifica o uso da classe *SelfDiagnosisManager* que contém uma *struct* implementando o protocolo *APIRequestable.*

3.3.2.2 Refatoração das interfaces de usuário

No desenvolvimento de aplicativos iOS há duas possibilidades de implementação para se criar interfaces de usuário:

- ***ViewCode:*** Implementação de forma programática, ou seja, por comandos que instanciam os componentes da biblioteca de interfaces *UIKit*.
- ***Storyboard:*** Implementação que utiliza a interface gráfica da IDE para a elaboração das interfaces.
  - Na prática, a IDE cria um arquivo de Extensible Markup Language - Linguagem de Marcação Extensível (XML) especificando os componentes de interfaces que compõem a tela e também suas características.

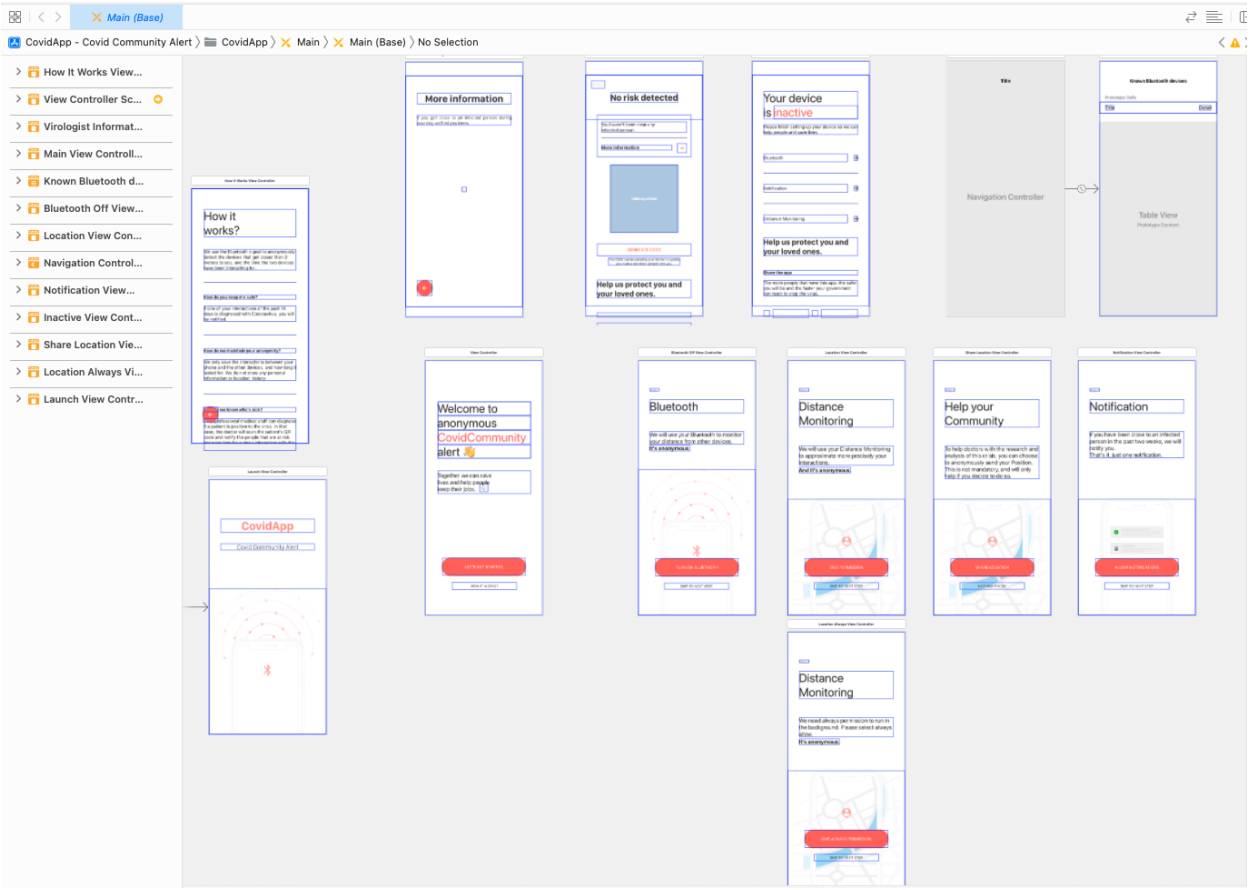
Figura 19 – Exemplo de implementação utilizando *Storyboards*.



Fonte: Autor (2022)

Quanto a implementação do aplicativo era utilizado apenas um *storyboard* para implementar todas as interfaces de usuário como mostra a figura **??**.

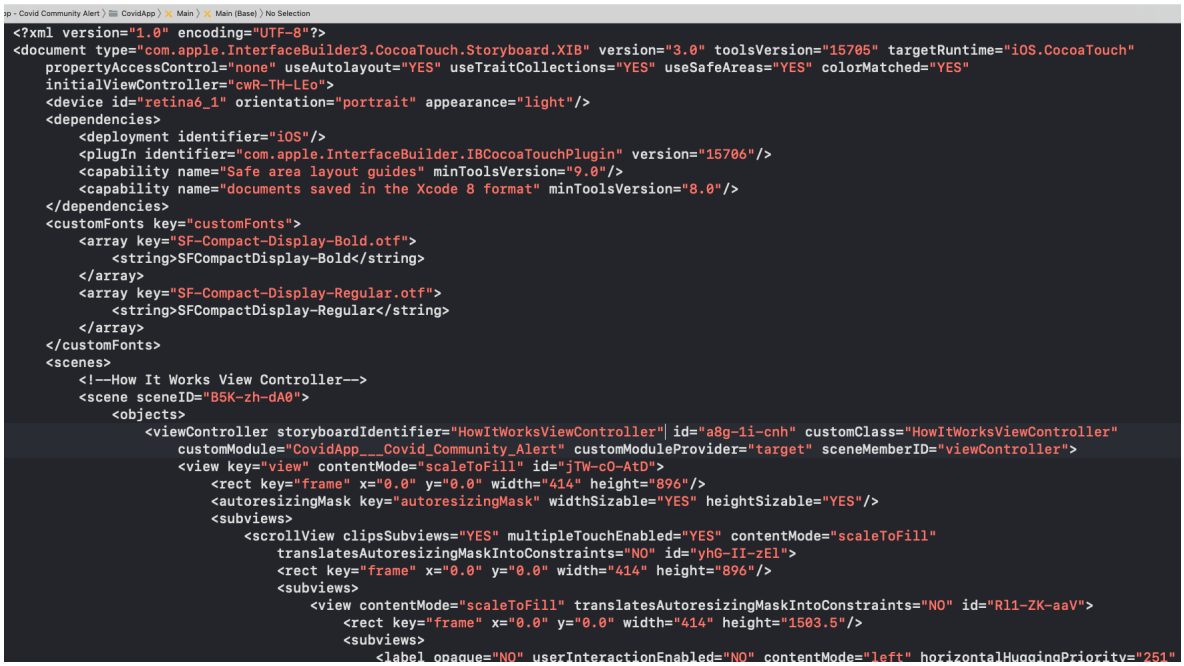
Figura 20 – *Storyboard* central do projeto



Fonte: Autor (2022)

Embora os *storyboards* tenham facilitado a maneira como se constroem interfaces, também trouxeram problemas, pois ao centralizar todas as telas em um só arquivo, isso gerava um arquivo em XML com várias informações que muitas vezes faziam mais sentido para IDE do que para o desenvolvedor. E ao trabalhar em equipe se tornou comum o aparecimento de conflitos de versionamento complexos de serem resolvidos.

Figura 21 – *Storyboard* central do projeto em XML

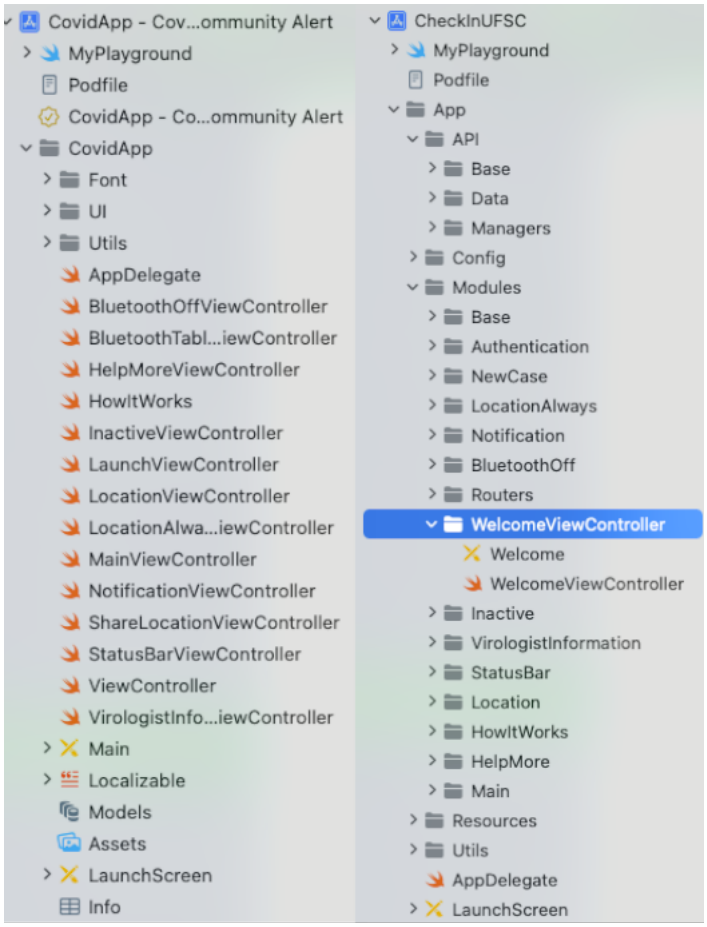


```
<?xml version="1.0" encoding="UTF-8"?>
<document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.XIB" version="3.0" toolsVersion="15705" targetRuntime="iOS.CocoaTouch"
    propertyAccessControl="none" useAutolayout="YES" useTraitCollections="YES" useSafeAreas="YES" colorMatched="YES"
    initialViewController="cwR-TH-LEo">
    <device id="retina6_1" orientation="portrait" appearance="light"/>
    <dependencies>
        <deployment identifier="iOS"/>
        <plugIn identifier="com.apple.InterfaceBuilder.IBCocoaTouchPlugin" version="15706"/>
        <capability name="Safe area layout guides" minToolsVersion="9.0"/>
        <capability name="documents saved in the Xcode 8 format" minToolsVersion="8.0"/>
    </dependencies>
    <customFonts key="customFonts">
        <array key="SF-Compact-Display-Bold.otf">
            <string>SFCompactDisplay-Bold</string>
        </array>
        <array key="SF-Compact-Display-Regular.otf">
            <string>SFCompactDisplay-Regular</string>
        </array>
    </customFonts>
    <scenes>
        <!--How It Works View Controller-->
        <scene sceneID="B5K-zh-dA0">
            <objects>
                <viewController storyboardIdentifier="HowItWorksViewController" id="a8g-1i-cnh" customClass="HowItWorksViewController"
                    customModule="CovidApp___Covid_Community_Alert" customModuleProvider="target" sceneMemberID="viewController">
                    <view key="view" contentMode="scaleToFill" id="jTW-cO-AtD">
                        <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
                        <autoresizingMask key="autoresizingMask" widthSizable="YES" heightSizable="YES"/>
                        <subviews>
                            <scrollView clipsSubviews="YES" multipleTouchEnabled="YES" contentMode="scaleToFill"
                                translatesAutoresizingMaskIntoConstraints="NO" id="yhG-II-zEl">
                                <rect key="frame" x="0.0" y="0.0" width="414" height="896"/>
                                <subviews>
                                    <view contentMode="scaleToFill" translatesAutoresizingMaskIntoConstraints="NO" id="Rl1-ZK-aaV">
                                        <rect key="frame" x="0.0" y="0.0" width="414" height="1503.5"/>
                                        <subviews>
                                            <label opaque="NO" userInteractionEnabled="NO" contentMode="left" horizontalHuggingPriority="251"
```

Fonte: Autor (2022)

Com isso, optou-se por realizar o desacoplamento dos *storyboards* do projeto, criando assim um *storyboard* para cada tela do projeto. Dessa forma foi possível atualizar a estrutura de diretórios do projeto, que antes contava com todos os arquivos responsáveis pelo controle de tela (*ViewControllers* juntos, como mostra a figura abaixo:

Figura 22 – Esquema de diretórios do projeto antes e depois das alterações



Fonte: Autor (2022)

Com o novo esquema, os arquivos do projeto ficaram mais organizados, possibilitando que no futuro, caso novos desenvolvedores precisem fazer alterações nessa base de código, a organização de seus diretórios deixará claro qual a função que os arquivos armazenados por ela desempenham no projeto.

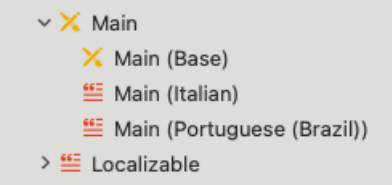
3.3.2.3 Localização

A localização do aplicativo foi outro ponto de melhoria encontrado durante a investigação do código, pois era possível encontrar a tradução dos textos do aplicativo em dois lugares diferentes: no arquivo central de telas e em um arquivo dedicado.

Assim como *storyboards* comportam que os desenvolvedores armazenem múltiplas telas em um mesmo arquivo, eles também permitem que as interfaces contidas neles sejam localizadas.

Embora, o arquivo *Main.storyboard* tenha sido divido na etapa anterior, para as strings de localização do aplicativo optou-se por centralizá-las em um arquivo desas-

Figura 23 – Arquivos de localização do aplicativo



Fonte: Autor (2022)

sociado de interface, ou seja, utilizou-se o arquivo *Localizable* que já estava presente no projeto anteriormente para criar essa localização.

Outra mudança relacionada foi a remoção da localização para o idioma italiano e a inserção do idioma inglês. Essa alteração ocorreu pela ausência de falantes da língua italiana na equipe do projeto.

3.3.2.4 Navegação entre telas

A implementação da lógica de navegação entre telas do projeto foi encontrada espalhada pelos controladores das telas do aplicativo, como mostra a figura abaixo, destacando a navegação sendo realizada diretamente no escopo da função da classe controladora.

Novamente, visualizou-se a oportunidade de aplicar o princípio DRY na implementação e com isso foi criada uma classe para centralizar a responsabilidade pela navegação chamada de *RootRouter*.

Com isso a lógica ficou concentrada em apenas uma só classe, permitindo previsibilidade quando uma manutenção relacionada a navegação do aplicativo for necessária.

3.3.3 Implementação de novas interfaces de usuário

Com base nas interfaces de usuário implementadas no aplicativo Android, haviam duas que não estavam presentes na implementação iOS, portanto as seções abaixo apresenta as interfaces que foram implementadas.

3.3.3.1 Autenticação

Como foi apresentado anteriormente, o aplicativo do CheckIn@UFSC tem como um de seus requisitos a autenticação do usuário com o sistema de autenticação centralizada da UFSC e para isso foi adicionada a seguinte interface de usuário no fluxo de autenticação do aplicativo:

Figura 24 – Função da classe *ViewController* responsável pela navegação

```
class ViewController: StatusBarViewController {
    }

    private func continueNavigation(){
        self.dismiss(animated: true, completion: nil)
        if StorageManager.shared.isFirstAccess(){
            print("FIRST ACCESS")
            let storyboard = UIStoryboard(name: "Main", bundle: nil)
            let controller = storyboard.instantiateViewController(withIdentifier:
                "BluetoothOffViewController")
            UIApplication.shared.windows.first?.rootViewController = controller
            UIApplication.shared.windows.first?.makeKeyAndVisible()
        }else{
            print("LATER ACCESS")
            if Utils.isActive(){
                let storyboard = UIStoryboard(name: "Main", bundle: nil)
                let controller = storyboard.instantiateViewController(withIdentifier:
                    "MainViewController")
                UIApplication.shared.windows.first?.rootViewController = controller
                UIApplication.shared.windows.first?.makeKeyAndVisible()
            }else{
                let storyboard = UIStoryboard(name: "Main", bundle: nil)
                let controller = storyboard.instantiateViewController(withIdentifier:
                    "InactiveViewController")
                UIApplication.shared.windows.first?.rootViewController = controller
                UIApplication.shared.windows.first?.makeKeyAndVisible()
            }
        }
    }
```

Fonte: Autor (2022)

Figura 25 – Classe *RootRouter*

```
final class RootRouter: PresentView {
    // MARK: Static

    static func initModule() -> RootRouter {
        return RootRouter()
    }

    // MARK: Actions

    func presentInitialScreen() {•••}

    // MARK: Private

    func presentWelcomeScreen() {
        let viewController = WelcomeViewController.initModule()
        presentView(viewController)
    }

    func presentHomeScreen() {•••}

    func presentInactiveHomeScreen() {•••}
```

Fonte: Autor (2022)

Figura 26 – Navegação após a introdução da classe de roteamento

```
private func continueNavigation(){
    self.dismiss(animated: true, completion: nil)
    if StorageManager.shared.isFirstAccess(){
        AppDelegate.shared.rootRouter.presentBluetoothOffScreen()
    }else{
        print("LATER ACCESS")
        if Utils.isActive(){
            AppDelegate.shared.rootRouter.presentHomeScreen()
        }else{
            AppDelegate.shared.rootRouter.presentInactiveHomeScreen()
        }
    }
}
```

Fonte: Autor (2022)

Figura 27 – Tela de autenticação

Fonte: Autor (2022)

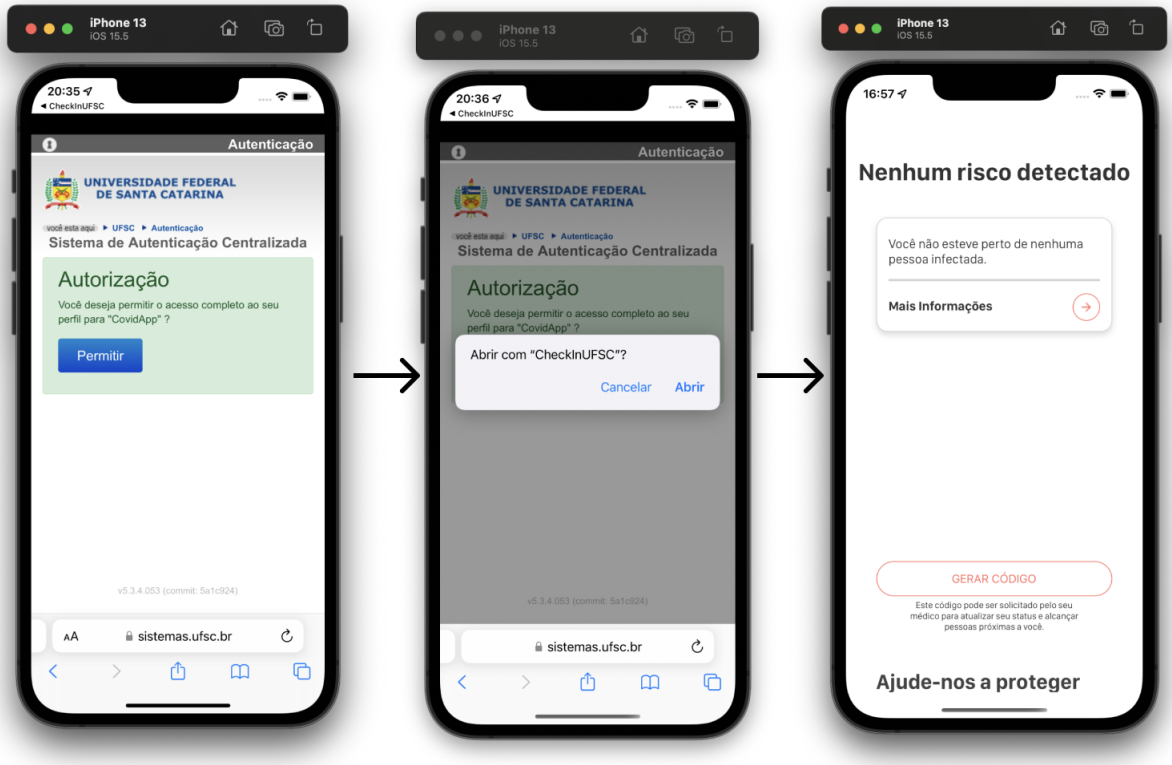
Ao pressionar com o botão “Acessar com IDUFSC”, é aberto o navegador do dispositivo e o usuário é direcionado para a página web do CAS.

Figura 28 – Autenticação com o CAS UFSC

Fonte: Autor (2022)

Ao realizar o login com sucesso, o usuário é redirecionado para o fluxo principal do aplicativo:

Figura 29 – Redirecionamento do usuário para o fluxo principal do aplicativo



Fonte: Autor (2022)

Caso o login não seja feito com sucesso ou se o usuário fechar o aplicativo na etapa de autenticação, o aplicativo irá exibir o estado de aplicativo inativo.

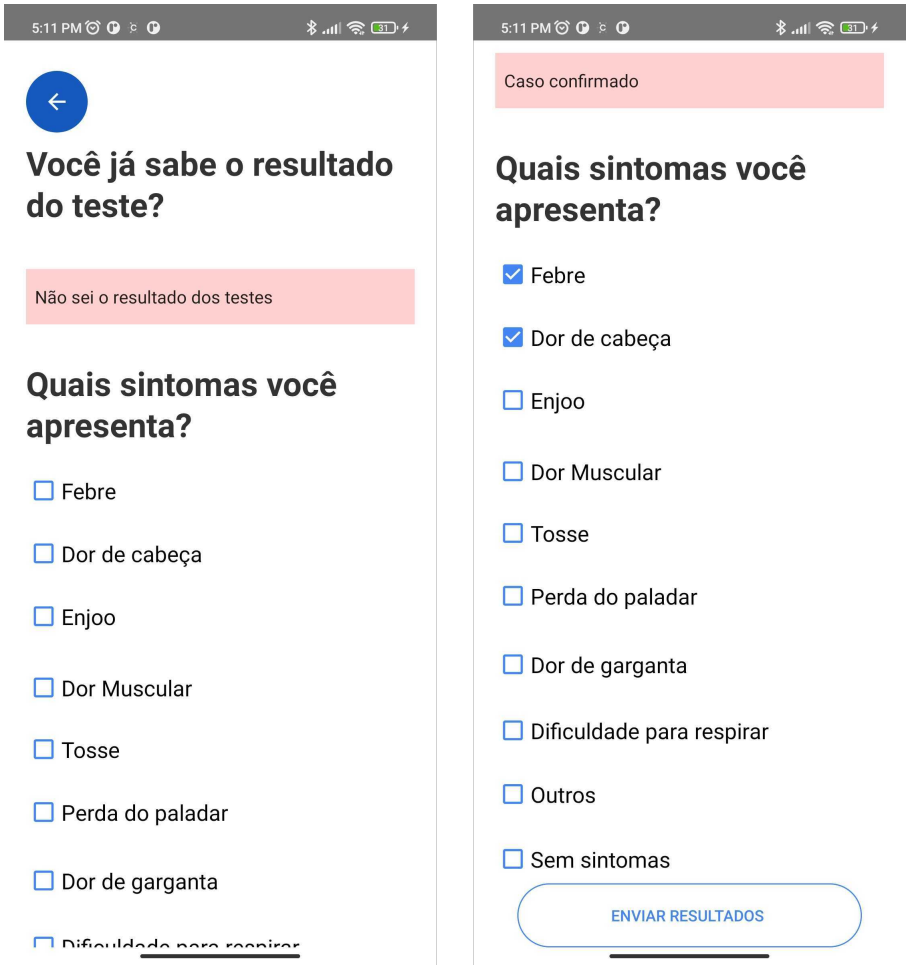
Figura 30 – Tela de funções inativas

Fonte: Autor (2022)

3.3.3.2 Autodiagnóstico

A interface de autodiagnóstico foi adicionada pela equipe CheckIn@UFSC no aplicativo Android. A função que permite que os usuários notifiquem anonimamente quando possuem sintomas ou se estão positivos para a doença.

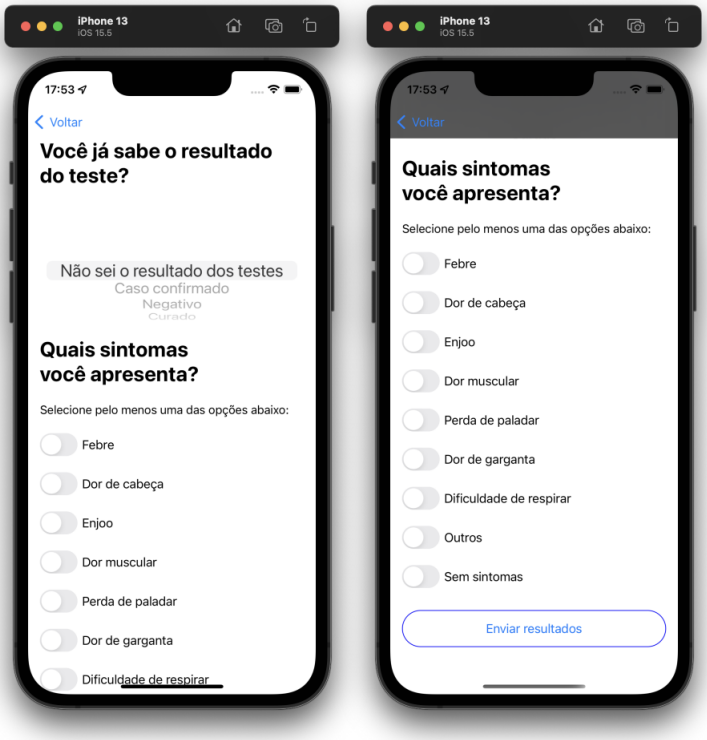
Figura 31 – Tela Autodiagnóstico — Android



Fonte: Autor (2022)

Portanto, com base nessa interface apresentada na figura acima foi criada a interface iOS correspondente. Como há diferenças entre os componentes de interface de usuário entre um sistema operacional, é possível notar algumas diferenças visuais, mas ambas as interfaces realizam a mesma função em ambos os aplicativos.

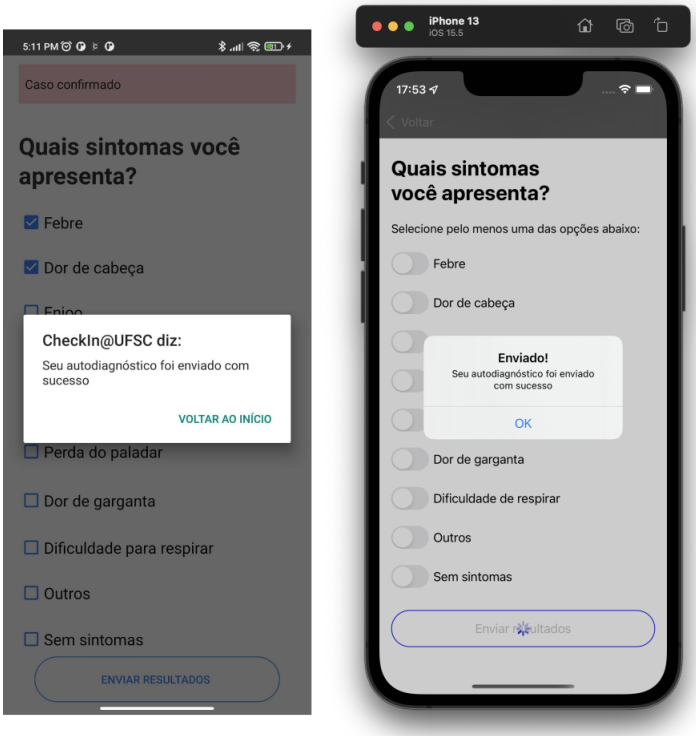
Figura 32 – Tela Autodiagnóstico - iOS



Fonte: Autor (2022)

E ao enviar o autodiagnóstico com sucesso é exibido um aviso ao usuário indicando que a operação foi realizada. A figura abaixo ilustra as interfaces do Android e do iOS respectivamente:

Figura 33 – Aviso de envio de autodiagnóstico bem-sucedido



Fonte: Autor (2022)

4 TESTES

Para a realização dos testes do aplicativo iOS foi usado como base os testes realizados pela equipe de desenvolvimento do aplicativo Android. No anexo A, anexado no final desse trabalho, é possível ter acesso ao documento com todos os casos de testes e resultados obtidos por essa equipe.

Além disso, os testes descritos nesse documento traduzem um estado do aplicativo antes da adição das telas de autenticação com a UFSC e autodiagnóstico, além de fazerem menção a funções que não existem mais no aplicativo como a descrita no "Caso de teste 5: Usuário gera seu código de paciente".

Dessa forma foram filtrados os casos de teste que não faziam parte do atual conjunto de funções oferecidas por ambos aplicativos, resultando na seguinte tabela com os casos de teste:

Quadro 4 – Casos de teste

Cód.	**Título**
CT-1	Usuário realiza *onboarding*
CT-2	Usuário interrompe *onboarding*
CT-3	Usuário pula as etapas do *onboarding*
CT-4	Usuário obtém mais informações sobre sua situação
CT-5	Usuário consegue compartilhar o aplicativo
CT-6	Usuário obtém mais informações sobre como funciona o aplicativo
CT-7	Usuário reativa as permissões caso necessário
CT-8	As interações são registradas corretamente no banco de dados
CT-9	Usuário consegue enviar o autodiagnóstico com sucesso
CT-10	Os autodiagnósticos enviados são registrados corretamente no banco de dados

Nas seções a seguir prosseguiu-se com definição dos seus pré-requisitos, descrição e testagem de cada um dos casos listados acima.

4.0.1 Caso de Teste 1: Usuário realiza *onboarding*

4.0.1.1 Pré-requisitos

- Possuir o aplicativo instalado, sem tê-lo aberto desde a instalação.
- O dispositivo utiliza a língua portuguesa como idioma principal.
- Ter conexão com a *Internet* para realizar a autenticação.

4.0.1.2 Descrição

Esse teste planeja verificar se o uso do aplicativo pela primeira vez é bem-sucedido. Para isso espera-se que o usuário consiga passar por todas as telas que integram o fluxo de introdução. No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

Quadro 5 – Passos para execução do CT-1

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela de Boas-vindas.
2	Clicar no botão “Configurar”.	É exibida uma verificação de *captcha*.
3	Selecionar as imagens que correspondem com a verificação e clicar em “Verificar”.	É exibida a tela de permissão de uso do *Bluetooth*.
4	Clicar no botão “Conceder Permissão” e permitir o uso do *Bluetooth* por meio do diálogo do sistema.	É exibida a tela de permissão de uso da localização do usuário.
5	Clicar no botão “Conceder Permissão” e permitir o uso da localização do usuário por meio do diálogo do sistema, selecionando a opção “Permitir Durante o Uso do Aplicativo”.	É exibida a tela de permissão de notificações.
6	Clicar no botão “Permitir notificações” e permitir que o aplicativo envie notificações ao usuário por meio do diálogo do sistema, selecionando a opção “Permitir”.	É exibida a tela de autenticação.
7	Clicar no botão “Acessar com IDUFSC”.	É aberto o navegador web na página do CAS UFSC.
8	Realizar a autenticação com sucesso e permitir o acesso do perfil pelo aplicativo.	É exibido diálogo do sistema com o texto: “Abrir com ’CheckInUFSC’”
9	Clicar no botão “Abrir” do diálogo exibido.	O usuário é redirecionado ao aplicativo que exibe a tela principal.

4.0.1.3 Resultado

Foi possível realizar o *onboarding* ao executar todos os passos elencados por esse caso de teste.

4.0.2 Caso de Teste 2: Usuário interrompe *onboarding*

4.0.2.1 Pré-requisitos

- Possuir o aplicativo instalado, sem tê-lo aberto desde a instalação.
- O dispositivo utiliza a língua portuguesa como idioma principal.

4.0.2.2 Descrição

Esse teste planeja verificar se é possível conceder as permissões necessárias para o funcionamento do aplicativo após interromper o processo de introdução. No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

Quadro 6 – Passos para execução do CT-2

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela de Boas-vindas.
2	Clicar no botão “Configurar”.	É exibida uma verificação de *captcha*.
3	Selecionar as imagens que correspondem com a verificação e clicar em “Verificar”.	É exibida a tela de permissão de uso do *Bluetooth*.
4	Encerrar o aplicativo e abrir ele novamente.	É exibida a tela de “Dispositivo Inativo” com as permissões que não foram concedidas escritas em vermelho.
5	Clicar em “Conceder permissão bluetooth”.	É exibida a tela de permissão de uso do *Bluetooth*.
6	Clicar no botão “Ligar o Bluetooth” e permitir o uso do Bluetooth por meio do diálogo do sistema.	É exibida a tela de permissão de uso da localização do usuário.
7	Clicar no botão “Conceder Permissão” e permitir o uso da localização do usuário por meio do diálogo do sistema, selecionando a opção “Permitir Durante o Uso do Aplicativo”.	É exibida a tela de permissão de notificações.
8	Clicar no botão “Permitir notificações” e permitir que o aplicativo envie notificações ao usuário por meio do diálogo do sistema, selecionando a opção “Permitir”.	É exibida a tela de autenticação.
9	Clicar no botão “Acessar com IDUFSC”.	É aberto o navegador web na página do CAS UFSC.
10	Realizar a autenticação com sucesso e permitir o acesso do perfil pelo aplicativo.	É exibido diálogo do sistema com o texto: “Abrir com ’CheckInUFSC’”
11	Clicar no botão “Abrir” do diálogo exibido.	O usuário é redirecionado ao aplicativo que exibe a tela principal com o nome dele.

4.0.2.3 Resultado

Foi possível realizar a interrupção *onboarding* ao executar todos os passos elencados por esse caso de teste.

4.0.3 Caso de Teste 3: Usuário pula as etapas do *onboarding*

4.0.3.1 Pré-requisitos

- Possuir o aplicativo instalado, sem tê-lo aberto desde a instalação.
- O dispositivo utiliza a língua portuguesa como idioma principal.
- Ter conexão com a Internet para realizar a autenticação.

4.0.3.2 Descrição

Esse teste planeja verificar se é possível pular as telas que solicitam as permissões necessárias para o funcionamento do aplicativo e concedê-las após a autenticação do usuário. No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

Quadro 7 – Passos para execução do CT-3

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela de Boas-vindas.
2	Clicar no botão “Configurar”.	É exibida uma verificação de *captcha*.
3	Selecionar as imagens que correspondem com a verificação e clicar em “Verificar”.	É exibida a tela de permissão de uso do *Bluetooth*.
4	Encerrar o aplicativo e abrir ele novamente.	É exibida a tela de “Dispositivo Inativo” com as permissões que não foram concedidas escritas em vermelho.
5	Clicar em “Pular”.	É exibida a tela de permissão de uso da localização do usuário.
6	Clicar no botão “Pular”	É exibida a tela de permissão de notificações.
7	Clicar no botão “Pular”.	É exibida a tela de autenticação.
8	Clicar no botão “Acessar com IDUFSC”.	É aberto o navegador web na página do CAS UFSC.
9	Realizar a autenticação com sucesso e permitir o acesso do perfil pelo aplicativo.	É exibido diálogo do sistema com o texto: “Abrir com ’CheckInUFSC’”
10	Clicar no botão “Abrir” do diálogo exibido.	O usuário é redirecionado ao aplicativo que exibe a tela de “Dispositivo inativo” com as permissões que não foram concedidas escritas em vermelho.
11	Clicar em “Conceder permissão bluetooth”.	É exibida a tela de permissão de uso do *Bluetooth*.
12	Clicar no botão “Ligar o Bluetooth” e permitir o uso do Bluetooth por meio do diálogo do sistema.	É exibida a tela de permissão de uso da localização do usuário.
13	Clicar no botão “Conceder Permissão” e permitir o uso da localização do usuário por meio do diálogo do sistema, selecionando a opção “Permitir Durante o Uso do Aplicativo”.	É exibida a tela de permissão de notificações.
14	Clicar no botão “Permitir notificações” e permitir que o aplicativo envie notificações ao usuário por meio do diálogo do sistema, selecionando a opção “Permitir”.	É exibida a tela principal do aplicativo com o nome do usuário.

4.0.3.3 Resultado

Foi possível realizar a interrupção *onboarding* ao executar todos os passos elencados por esse caso de teste.

4.0.4 Caso de Teste 4: Usuário obtém mais informações sobre sua situação

4.0.4.1 Pré-requisitos

- Possuir o aplicativo instalado;
- Ter finalizado o *onboarding*;

4.0.4.2 Descrição

Esse teste planeja verificar se o usuário consegue obter mais informações sobre sua situação. No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

Quadro 8 – Passos para execução do CT-4

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela principal com o nome do usuário.
2	Clicar no botão ao lado de “Mais informações”.	É exibida a tela de mais informações detalhando a situação do usuário.

4.0.4.3 Resultado

Foi possível verificar a situação ao executar todos os passos elencados por esse caso de teste.

4.0.5 Caso de Teste 5: Usuário consegue compartilhar o aplicativo

4.0.5.1 Pré-requisitos

- Possuir o aplicativo instalado;
- Ter finalizado o *onboarding*.

4.0.5.2 Descrição

Esse teste visa verificar se é possível utilizar a função de compartilhar o CheckIn@UFSC por meio do aplicativo de SMS. No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

Quadro 9 – Passos para execução do CT-5

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela principal com o nome do usuário.
2	Rolar a página até o texto “Compartilhe o aplicativo” ficar visível e clicar no botão “SMS”.	O aplicativo de SMS é aberto com o link do website do CheckIn@UFSC

4.0.5.3 Resultado

Foi possível compartilhar o aplicativo por meio do das mensagens SMS ao executar todos os passos elencados por esse caso de teste.

4.0.6 Caso de Teste 6: Usuário obtém mais informações sobre como funciona o aplicativo

4.0.6.1 Pré-requisitos

- Possuir o aplicativo instalado.
- O dispositivo utiliza a língua portuguesa como idioma principal.
- Ter conexão com a Internet para realizar a autenticação. (Opcional, caso o usuário opte por acessar as informações adicionais do aplicativo sem iniciar o *onboarding*

4.0.6.2 Descrição

Esse teste visa verificar se é possível acessar a tela com mais informações sobre o aplicativo a partir da tela de Boas-vindas ou a partir da tela inicial. Nos quadros abaixo são especificados quais são os passos esperados para se concluir esse teste:

Quadro 10 – Passos para execução do CT-6 (Sem autenticação)

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela de Boas-vindas.
2	Clicar no botão “Como funciona?”.	É exibida a tela de informações adicionais com o título “Como funciona?”

Quadro 11 – Passos para execução do CT-6 (Com autenticação)

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela principal com o nome do usuário.
2	Rolar a página até o botão “Como funciona?” ficar visível e clicar nele.	É exibida a tela de informações adicionais com o título “Como funciona?”

4.0.6.3 Resultado

Foi possível consultar informações adicionais sobre o aplicativo ao executar todos os passos elencados por esse caso de teste.

4.0.7 Caso de Teste 7: Usuário reativa as permissões caso necessário

4.0.7.1 Pré-requisitos

- Possuir o aplicativo instalado.
- O dispositivo utiliza a língua portuguesa como idioma principal.

4.0.7.2 Descrição

Esse teste visa verificar se o aplicativo solicita novamente as permissões necessárias para o funcionamento caso alguma delas seja desativada ou revogada após o *onboarding*. No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

4.0.7.3 Resultado

Ao desativar a permissão de *bluetooth* o aplicativo exibiu a tela de “Dispositivo Inativo” conforme os passos elencados por esse caso de teste.

4.0.8 Caso de Teste 8: As interações são registradas corretamente no banco de dados

4.0.8.1 Pré-requisitos

- Possuir o aplicativo instalado;
- Ter finalizado o *onboarding*;
- O dispositivo utiliza a língua portuguesa como idioma principal;
- Ter conexão com a Internet para que as interações sejam enviadas;
- Saber quais são os *ids* de usuário que serão utilizados para o teste;
- Ter as credenciais de acesso ao *PHPMyAdmin* referente ao banco de dados da infraestrutura do projeto.

4.0.8.2 Descrição

Esse teste planeja verificar se o aplicativo detecta outros dispositivos próximos e como ele realiza o registro dessa interação.

Além disso, outro objetivo é aferir se essa detecção gera dados confiáveis relacionados a duração e distância entre os dispositivos testados.

Para a realização desse caso de testes são especificados os seguintes passos e seus resultados esperados:

Quadro 12 – Passos para execução do CT-8

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela principal com o nome do usuário.
2	Colocar o aplicativo em segundo plano e bloquear o dispositivo.	O aplicativo continua ativo em segundo plano, sendo possível observar a instância dele aberta no Seletor de Aplicativos(4).
3	Deixar os dispositivos próximos durante a duração do teste e conforme a distância escolhida	O aplicativo continua aberto e detecta dispositivos próximos a ele.
4	Após o período estabelecido, afastar os dispositivos a pelo menos 5 metros ou de forma que não consigam realizar a leitura de um dispositivo próximo.	Os aplicativos enviam os dados para o servidor.
5	Aguardar o envio dos dados e processamento dos dados pelo servidor	É esperado que o servidor registre o contato no banco de dados entre 5 a 10 minutos após o teste.
6	Verificar o banco de dados buscando os *ids* dos dispositivos testados	Os registros de interação devem concordar com o teste realizado

Para a realização dos passos acima, foram utilizados 2 dispositivos *iOS* e *Android*, totalizando 4 diferentes aparelhos rodando o aplicativo em segundo plano, no quadro abaixo são detalhados cada um deles:

Quadro 13 – Dispositivos usados para execução do CT-8

Nome	Sistema Operacional	Versão	Memória Interna	Memória RAM
iPhone 11	iOS	15.5	64GB	4GB
iPhone 13	iOS	16	32GB	4GB
Poco X3 NFC	Android	11	128GB	8GB
Moto G5S	Android	8	32GB	3GB

Para a realização dos testes foram criados cenários de testes com base no número de dispositivos, na distância entre eles e considerando os testes que haviam sido feitos anteriormente (descritos no Anexo A):

Quadro 14 – Cenários usados para execução do CT-8

Cenário	Variante	Distância	Tempo	Nro. de dispositivos
1	1.1	d <0,40 m	1 min	2
	1.2	d <0,40 m	1 min	3
	1.3	d <0,40 m	1 min	4
2	2.1	d <0,40 m	3 min	2
	2.2	d <0,40 m	3 min	3
	2.3	d <0,40 m	3 min	4
3	3.1	d <0,40 m	10 min	2
	3.2	d <0,40 m	10 min	3
	3.3	d <0,40 m	10 min	4
4	4.1	0,40 m $\leq d \leq 2m$	1 min	2
	4.2	0,40 m $\leq d \leq 2m$	1 min	3
	4.3	0,40 m $\leq d \leq 2m$	1 min	4
5	5.1	0,40 m $\leq d \leq 2m$	3 min	2
	5.2	0,40 m $\leq d \leq 2m$	3 min	3
	5.3	0,40 m $\leq d \leq 2m$	3 min	4
6	6.1	0,40 m $\leq d \leq 2m$	10 min	2
	6.2	0,40 m $\leq d \leq 2m$	10 min	3
	6.3	0,40 m $\leq d \leq 2m$	10 min	4
7	7.1	d $\geq 4m$	1 min	2
	7.2	d $\geq 4m$	1 min	3
	7.3	d $\geq 4m$	1 min	4
8	8.1	d $\geq 4m$	3 min	2
	8.2	d $\geq 4m$	3 min	3
	8.3	d $\geq 4m$	3 min	4
9	9.1	d $\geq 4m$	10 min	2
	9.2	d $\geq 4m$	10 min	3
	9.3	d $\geq 4m$	10 min	4

Os dados referentes às interações são armazenados no banco de dados, na tabela *interactionlog*, da seguinte maneira:

Quadro 15 – Colunas da tabela *interactionlog*

Nome	**Tipo**	**Descrição**
id	*bigint(20)*	Identificador da interação armazenada.
my_id	*bigint(20)*	Identificador do dispositivo que enviou a interação.
other_id	*bigint(20)*	Identificador do dispositivo detectado na interação.
timestamp_start	*int(11)*	Carimbo de tempo do momento em que a interação começou.
timestamp_end	*int(11)*	Carimbo de tempo do momento em que a interação terminou.
duration	*int(11)*	Duração total da interação em segundos.
rssi	*float*	Intensidade do sinal durante a interação
distance	*float*	Distância onde os dispositivos estavam no momento da interação. Quando enviado por um dispositivo *iOS* pode assumir os seguintes valores: • 0 - Desconhecido, ou seja, não foi possível identificar a distância • 1 - Imediato, está a alguns centímetros; • 2 - Próximo, está entre 1 e 10 metros de distância; • 3 - Longe, está a mais de 10 metros de distância. Já a leitura enviada por aparelhos Android é uma estimação em metros da distância entre dispositivos.
platform	*char(1)*	Indica qual o sistema operacional do dispositivo que enviou a interação, sendo “a” para *Android* e “i” para *iOS*

É importante explicar que a diferença entre os valores de distância enviados por dispositivos *iOS* e dispositivos *Android* se dá, pois a implementação de *beacon* da *Apple*, o *iBeacon* não torna público para os desenvolvedores dados sobre a intensidade do sinal(22), como acontece na implementação *AltBeacon* desenvolvida pela *Google*(5).

Sobre as colunas armazenadas no banco de dados, foram omitidas do quadro 15 as colunas de *latitude* e *longitude*, pois ambas não estão sendo utilizadas nas leituras colhidas durantes os testes.

4.0.8.3 Resultado

Abaixo serão apresentados cada um dos cenários testados com os resultados encontrados para cada uma de suas variantes.

4.0.8.3.1 Cenário 1 (d <0,40 m, 1 min)

Para esse cenário foi possível encontrar leituras no banco de dados para todas as variantes testadas.

Ao testar a variante 1.1, foram utilizados os dispositivos iPhone 11 e Poco X3 NFC, ambos foram deixados lado a lado executando o aplicativo em segundo plano a uma distância de aproximadamente 30 centímetros. Foi possível localizar no banco apenas uma leitura referente a esse teste:

Quadro 16 – Resultados do teste da variante 1.1

Dispositivo detector	**Dispositivo Detectado**	**Duração (s)**	**RSSI**	**Distância**
Poco X3 NFC	iPhone 11	34	20	0

Como o quadro 16 mostra, o dispositivo *Android* detectou o dispositivo *iOS* por cerca de 34 segundos, ou seja, durante metade do tempo testado. A intensidade de sinal era boa e a distância estimada pelo dispositivo está próxima da realidade, indicando que ambos estavam próximos.

Ao testar a variante 1.2, foram utilizados os dispositivos iPhone 13, Moto G5S e Poco X3 NFC, todos foram deixados lado a lado executando o aplicativo em segundo plano a uma distância de aproximadamente 30 centímetros. Foi possível localizar no banco duas leituras referentes a esse teste:

Quadro 17 – Resultados do teste da variante 1.2

Dispositivo detector	**Dispositivo Detectado**	**Duração (s)**	**RSSI**	**Distância**
Moto G5S	iPhone 13	20	20	0
Poco X3 NFC	iPhone 13	13	41	0

Como o quadro 17 mostra, ambos os dispositivos *Android* detectaram o dispositivo *iOS* entre 13 e 20 segundos, menos da metade do tempo do teste. A intensidade de sinal era boa para ambos e a distância estimada pelos dispositivos está próxima da realidade, indicando que eles estavam próximos.

Ao testar a variante 1.3, foram utilizados os dispositivos iPhone 13, iPhone 11, Moto G5S e Poco X3 NFC, todos foram deixados lado a lado executando o aplicativo em segundo plano a uma distância de aproximadamente 30 centímetros. Foi possível localizar no banco apenas uma leitura referente a esse teste:

Como o quadro 18 mostra, apenas um dispositivo *Android* detectou o dispositivo *iOS* por 15 segundos, ou seja, por um quarto do tempo do teste. A intensidade de sinal

Quadro 18 – Resultados do teste da variante 1.3

Dispositivo detector	Dispositivo Detectado	Duração (s)	RSSI	Distância
Moto G5S	iPhone 13	15	28	0

era boa e a distância estimada está próxima da realidade, indicando que eles estavam próximos.

4.0.8.3.2 Cenário 2 (d <0,40 m, 3 min)

Ao testar o cenário 2 apenas a variante 2.1 apresentou resultados. Para o teste foram utilizados os dispositivos iPhone 11 e Moto G5S, os dispositivos foram deixados lado a lado executando o aplicativo em segundo plano a uma distância de aproximadamente 30 centímetros. Foi possível localizar no banco apenas uma leitura referente a esse teste:

Quadro 19 – Resultados do teste da variante 2.1

Dispositivo detector	Dispositivo Detectado	Duração (s)	RSSI	Distância
Moto G5S	iPhone 11	22	16	0

Como o quadro 19 mostra, o dispositivo *Android* detectou o dispositivo *iOS* por apenas 22 segundos. A intensidade de sinal era boa e a distância estimada está próxima da realidade, indicando que eles estavam próximos. Quanto às variantes não detectadas não foi possível encontrar a causa desse fato.

4.0.8.3.3 Cenário 3 (d <0,40 m, 10 min)

Ao testar o cenário 3 apenas a variante 2.3 apresentou resultados. Para o teste foram utilizados todos os dispositivos, deixados lado a lado executando o aplicativo em segundo plano a uma distância de aproximadamente 30 centímetros. Foi possível localizar no banco três leituras relacionadas a esse teste:

Quadro 20 – Resultados do teste da variante 2.1

Dispositivo detector	Dispositivo Detectado	Duração (s)	RSSI	Distância
Poco X3 NFC	Moto G5S	31	61	1.1
Poco X3 NFC	iPhone 11	32	49	0.2

Como o quadro 20 mostra, o dispositivo *Android* detectou dois dispositivos, sendo um *Android* e um *iOS*. O tempo de detecção ficou na média de 30 segundo. A intensidade de sinal de ambos era boa e a distância estimada está próxima da realidade, embora a estimativa da leitura do Moto G5S tenha indicado uma distância maior do que a distância real.

4.0.8.3.4 Cenário 4 (0,40 m $\leq d \leq$ 2m, 1min)

As variantes desse cenário não apresentaram nenhuma leitura.

4.0.8.3.5 Cenário 5 (0,40 m $\leq d \leq$ 2m, 3min)

As variantes desse cenário não apresentaram nenhuma leitura.

4.0.8.3.6 Cenário 6 (0,40 m $\leq d \leq$ 2m, 10min)

Para o cenário 6 apenas a variante 6.3 apresentou resultados. Foram utilizados todos os dispositivos, deixando-os lado a lado executando o aplicativo em segundo plano a uma distância de aproximadamente 1 metro. Foi possível localizar quatro leituras no banco referente a esse teste:

Quadro 21 – Resultados do teste da variante 6.3

Dispositivo detector	**Dispositivo Detectado**	**Duração (s)**	**RSSI**	**Distância**
iPhone 11	iPhone 13	10	59	1 m $\leq d \leq$ 10 *m*
iPhone 13	iPhone 11	11	58	1 m $\leq d \leq$ 10 *m*
Moto G5S	iPhone 13	24	50	0,3 m
Moto G5S	iPhone 11	58	53	0,3 m

O quadro 21 mostra, um dispositivo *iOS* pôde identificar outro e apenas o Moto G5S detectou os dispositivos *iOS*. O tempo de detecção entre os *iPhones* ficou na média dos 10 segundos, já a detecção do Moto G5S apresentou 24 e 58 segundos. A intensidade de sinal de ambos se mostrou boa e a distância estimada pelos iPhones condiz com a realidade, mas a distância estimada pelo Moto G5S foi menor do que a distância real.

4.0.8.3.7 Cenário 7 (d $\geq$ 4m, 1min)

As variantes desse cenário não apresentaram nenhuma leitura.

4.0.8.3.8 Cenário 8 (d $\geq$ 4m, 3min)

As variantes desse cenário não apresentaram nenhuma leitura.

4.0.8.3.9 Cenário 9 (d $\geq$ 4m, 10min)

Ao testar o cenário 9 apenas a variante 9.3 apresentou resultados. Para o teste foram utilizados todos os dispositivos, deixados lado a lado executando o aplicativo em

Quadro 22 – Resultados do teste da variante 2.1

Dispositivo detector	**Dispositivo Detectado**	**Duração (s)**	**RSSI**	**Distância**
iPhone 13	iPhone 11	9	75	d > 10m

segundo plano a uma distância de aproximadamente 4 metros. Foi possível localizar no banco apenas uma leitura relacionada a esse teste:

Como o quadro 22 mostra, um dispositivo *iOS* pôde identificar outro. O tempo de detecção foi de apenas 9 segundos. A intensidade de sinal de ambos era suficientemente boa e a distância estimada não condiz com a realidade, pois indica que os dispositivos estavam a mais de 10 metros, quando, na verdade estavam a 4 metros.

4.0.9 Caso de Teste 9: Os autodiagnósticos enviados são registrados corretamente no banco de dados

4.0.9.1 Pré-requisitos

- Possuir o aplicativo instalado;
- Ter finalizado o *onboarding*;
- O dispositivo utiliza a língua portuguesa como idioma principal;
- Ter conexão com a Internet para realizar o envio do autodiagnóstico;
- Saber qual é o id do usuário a ser testado;
- Ter as credenciais de acesso ao *PHPMyAdmin* referente ao banco de dados da infraestrutura do projeto.

4.0.9.2 Descrição

Esse teste planeja verificar se é possível enviar o autodiagnóstico com o aplicativo. E também se ao enviá-lo para o servidor, seu registro é realizado corretamente no banco de dados.

No banco de dados os autodiagnósticos enviados são armazenados na tabela *patient_statuses* que possui as seguintes colunas:

Quadro 23 – Colunas da tabela *patient_statuses*

Nome	**Tipo**	**Descrição**
id	*int(11)*	Identificador do autodiagnóstico armazenado
patient_id	*int(11)*	Identificador do dispositivo que enviou o autodiagnóstico
old_status	*int(11)*	Refere-se ao último resultado de teste enviado pelo usuário, podendo assumir os seguintes valores: • 0 - Não sabe o resultado dos testes • 1 - Caso confirmado • 2 - Negativo • 3 - Curado
actual_status	*int(11)*	Refere-se ao atual resultado de teste enviado pelo usuário, podendo assumir os mesmos valores que o campo *old_status*.
symptoms	*varchar(20)*	Sequência de 10 dígitos que indicam quais sintomas foram enviados pelo usuário. Cada dígito pode assumir os valores 0 quando não há ocorrência do sintoma e 1 quando há ocorrência de sintoma. Cada um dos dígitos se refere aos seguintes sintomas respectivamente: 1. Febre 2. Dor de cabeça 3. Enjoo 4. Dor muscular 5. Tosse 6. Perda do paladar 7. Dor de garganta 8. Dificuldade para respirar 9. Outros 10. Sem sintomas
updated_by	*int(11)*	Indica o *id* do usuário que fez a alteração do registro.
updated_at	*int(11)*	Carimbo de tempo em milissegundos indicando quando o registro foi criado
processed	*tinyint(1*)	Pode assumir os valores 0 quando o autodiagnóstico ainda não foi processado e 1 quando ele foi processado.

No quadro abaixo são especificados quais são os passos esperados para se concluir esse teste:

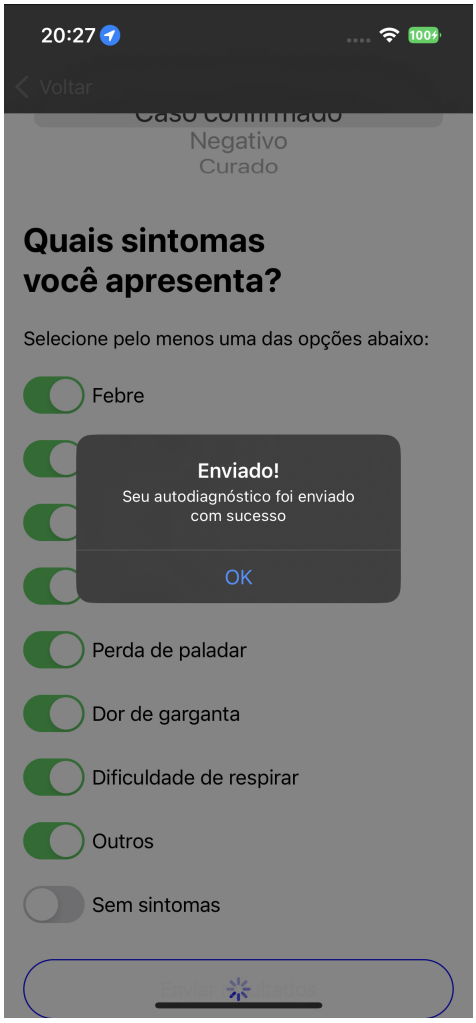
Quadro 24 – Passos para execução do CT-9

Passo	**Ação**	**Resultado esperado**
1	Abrir o aplicativo.	É exibida a tela principal com o nome do usuário.
2	Clicar no botão “Notificar Sintomas”.	É exibida a tela de autodiagnóstico com o título “Você já sabe o resultado do teste?”
3	Selecionar a opção “Caso confirmado” e todas as opções de sintomas, tirando a opção “Sem sintomas”.	Os componentes mostram corretamente todas as opções escolhidas.
4	Rolar a página até o fim e clicar no botão “Enviar resultados”.	É exibido um diálogo com o título “Enviado” e com a mensagem “Seu autodiagnóstico foi enviado com sucesso”
5	Acessar o banco de dados por meio do PHPMyAdmin e acessar a tabela *patient_statuses*	É exibida uma tabela com 8 colunas (*id*, *patient_id*, *old_status*, *actual_status*, *symptoms*, *updated_by*, *updated_at* e *processed*)
6	Localizar a linha que possui o *patient_id* equivalente ao id do usuário que enviou o autodiagnóstico	As colunas *actual_status* e *symptoms* devem conter os respectivos valores: 1 e 1111111100

4.0.9.3 Resultado

Para esse teste foi utilizado o usuário de *id* 342. E ao seguir as instruções indicadas no Quadro 24 foi possível realizar o envio do autodiagnóstico com sucesso, sendo que o diálogo previsto como resultado esperado do passo 4 foi exibido como mostra a figura 34. E ao verificar o banco de dados conforme os passos 5 e 6, detectou-se uma entrada compatível com os dados enviados para o servidor do projeto conforme a figura 35.

Figura 34 – Autodiagnóstico bem-sucedido ao testar o aplicativo



Fonte: Autor (2022)

Figura 35 – Registro do autodiagnóstico no banco de dados

id ▼ 1	patient_id	old_status	actual_status	symptoms	updated_by	updated_at	processed
274	342	0	1	1111111100	342	1670023576	1

Fonte: Autor (2022)

5 CONSIDERAÇÕES FINAIS

Com base no resultado dos testes pode-se concluir que o aplicativo iOS possui interface de usuário funcional, pois nos fluxos de introdução e o principal o usuário conseguiu alcançar os objetivos traçados nos testes de 1 a 7.

Já com relação ao teste 8, foi possível notar que as interações detectadas por dispositivos *iOS* adotaram um padrão breve de aproximadamente 10 segundos, embora nem mesmo as interações *Android* tenham apresentado tempo de detecção equivalente ao tempo testado.

Porém, foram encontradas leituras que possuíam uma estimação próxima da realidade, indicando que provavelmente os aplicativos não conseguiram manter a execução em segundo plano por alguma restrição do sistema operacional, como, por exemplo, políticas de economia de energia.

Além disso, um fato encontrado durante os testes e que não tinha ocorrido nos testes especificados no Anexo A foi a detecção de um dispositivo *iOS* por outro dispositivo *iOS*. Em um teste a parte, feito com o *iPhone* 11 e o *iPhone* 13, deixados próximos com aproximadamente 10 cm de distância durante 20 minutos, rodando o aplicativo em segundo plano, foi possível encontrar uma interação processada com duração de 2:42 m, porém com qualidade de sinal ruim e estimação de distância indicando que os aparelhos estavam a mais de 10 metros de distância.

Embora inconclusiva, a leitura demonstra haver possibilidade de detecção, o que anteriormente acreditava-se não ser possível por conta de políticas de segurança da *Apple*. Existe a hipótese, que isso seja fruto de uma atualização de políticas de uso do *bluetooth* em decorrência do protocolo lançado durante a pandemia(28) para esse fim, como também com o lançamento das *AirTags*, sendo rastreadores *bluetooth* de alta frequência(2) que interagem com qualquer dispositivo da marca, permitindo que seu proprietário tenha sua localização precisa, úteis para localizar itens como chaveiros.

Já com relação ao teste 9, foi alcançado o objetivo de enviar corretamente o autodiagnóstico do usuário, porém não foi testado o recebimento de notificações relacionadas à exposição do usuário ou mesmo um teste positivo, pois para implementar essa funcionalidade era necessária a assinatura de uma conta de desenvolvedor da *Apple*, que por falta de recursos não foi possível adquirir.

Dessa forma, é possível concluir que o aplicativo possui potencial para servir de ferramenta de rastreamento de contatos, sendo necessárias melhorias em seu funcionamento em segundo plano, ou mesmo adoção de alguma estratégia que faça com que o usuário interaja com o aplicativo, fazendo com que ele execute em primeiro plano.

5.1 TRABALHOS FUTUROS

Para trabalhos futuros, pode ser desenvolvida uma função de depuração ou mesmo interface voltada para esse fim que permita consultar se durante a execução em segundo plano do aplicativo houve leituras e quais foram seus valores.

Outra possibilidade também seria a investigação de métodos alternativos de interação com o usuário, visando superar limitações impostas pelos sistemas operacionais quando executando a aplicação em segundo plano.

E por fim, o autor vê como viável também uma investigação de usabilidade do aplicativo e geração de novas interfaces de usuário, buscando a otimização da experiência de usuário.

REFERÊNCIAS

1 ADVANCED Usage: Adapting and Retrying Requests with RequestInterceptor. Disponível em: https://github.com/Alamofire/Alamofire/blob/master/Documentation/AdvancedUsage.md#adapting-and-retrying-requests-with-requestinterceptor. Acesso em: 27 nov. 2022.

2 AIRTAG. Disponível em: https://www.apple.com/airtag/. Acesso em: 1 dez. 2022.

3 ALCANCE usuários em todas as telas. **flutter.dev**. Disponível em: https://flutter-dev.translate.goog/?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt-BR&_x_tr_pto=sc. Acesso em: 24 jul. 2022.

4 ALTERNE entre apps abertos no iPhone. **Apple Inc.** Disponível em: https://support.apple.com/pt-br/guide/iphone/iph1a1f981ad/ios. Acesso em: 27 nov. 2022.

5 ANDROID Beacon Library: Distance estimates. Disponível em: https://altbeacon.github.io/android-beacon-library/distance-calculations.html. Acesso em: 1 dez. 2022.

6 AUTENTICAÇÃO Centralizada (CAS). Disponível em: https://servicosti.sistemas.ufsc.br/publico/detalhes.xhtml?servico=355. Acesso em: 24 jul. 2022.

7 BLUETOOTH Low Energy beacon. **Wikipedia**. Disponível em: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy_beacon. Acesso em: 28 out. 2022.

8 CABEÇALHO HTTP. Disponível em: https://developer.mozilla.org/pt-BR/docs/Glossary/HTTP_header. Acesso em: 27 nov. 2022.

9 CHECKIN@UFSC. Disponível em: https://checkin.ufsc.br/. Acesso em: 24 jul. 2022.

10 CONTACT tracing in the context of COVID-19. **World Health Organization**. Disponível em:

https://apps.who.int/iris/bitstream/handle/10665/339128/WHO-2019-nCoV-Contact_Tracing-2021.1-eng.pdf. Acesso em: 28 out. 2022.

11 COVID Community Alert. Disponível em: https://coronavirus-outbreak-control.github.io/web/. Acesso em: 24 jul. 2022.

12 COVID-19 Dashboard by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University (JHU). Disponível em: https://www.arcgis.com/apps/dashboards/bda7594740fd40299423467b48e9ecf6. Acesso em: 24 jul. 2022.

13 DART overview. **dart.dev**. Disponível em: https://dart.dev/overview. Acesso em: 24 jul. 2022.

14 DIGITAL Health. **ScienceDirect**. Disponível em: https://www.sciencedirect.com/topics/medicine-and-dentistry/digital-health. Acesso em: 24 jul. 2022.

15 EXTENSIONS. Disponível em: https://docs.swift.org/swift-book/LanguageGuide/Extensions.html. Acesso em: 27 nov. 2022.

16 FIELDING, Roy; NOTTINGHAM, Mark; RESCHKE, Julian. RFC9110 - HTTP Semantics: Name method definitions. Disponível em: https://www.rfc-editor.org/rfc/rfc9110#name-method-definitions. Acesso em: 27 nov. 2022.

17 FLUTTER. **Wikipedia**. Disponível em: https://pt.wikipedia.org/wiki/Flutter. Acesso em: 24 jul. 2022.

18 GETTING Started with iBeacon. Disponível em: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf. Acesso em: 28 out. 2022.

19 GROUP, Bluetooth Special Interest. Bluetooth Low Energy. **bluetooth.com**. Disponível em: https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2012_2/bluetooth/ble.htm. Acesso em: 24 jul. 2022.

20 HÁ 15 anos Steve Jobs apresentava iPhone e promovia revolução tecnológica. **CNN Brasil**. Disponível em: https://www.cnnbrasil.com.br/tecnologia/ha-15-anos-steve-jobs-apresentava-iphone-e-promovia-revolucao-tecnologica/. Acesso em: 24 jul. 2022.

21 HUNT, Andrew; THOMAS, David. **The Pragmatic Programmer**. 1. ed. Estados Unidos: Addison-Wesley Professional, 2010.

22 IBEACONS: how to get broadcasted beacon power (txPower). Disponível em: https://stackoverflow.com/questions/24001702/ibeacons-how-to-get-broadcasted-beacon-power-txpower. Acesso em: 1 dez. 2022.

23 IDENTIFICANDO recursos na web. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web. Acesso em: 27 nov. 2022.

24 IOS 16 and iPadOS 16 are compatible with these devices. **Apple Inc.** Disponível em: https://support.apple.com/en-us/HT213411. Acesso em: 24 jul. 2022.

25 MALIK, Yashpal Singh *et al.* Coronavirus Disease Pandemic (COVID-19): Challenges and a Global Perspective. **Pathogens**, v. 9, n. 7, 2020. ISSN 2076-0817. DOI: 10.3390/pathogens9070519. Disponível em: https://www.mdpi.com/2076-0817/9/7/519. Acesso em: 24 jul. 2022.

26 MOVE content manually from your Android device to your iPhone, iPad, or iPod touch. **Apple Inc.** Disponível em: https://support.apple.com/en-us/HT205063. Acesso em: 24 jul. 2022.

27 OMS afirma que COVID-19 é agora caracterizada como pandemia, mar. 2020. Disponível em: https://www.paho.org/pt/news/11-3-2020-who-characterizes-covid-19-pandemic. Acesso em: 24 jul. 2022.

28 PRIVACY-PRESERVING Contact Tracing. *In*: [*S.l.*: *s.n.*], 2020. Disponível em: https://covid19.apple.com/contacttracing.

29 TEAM, Covid Community Alert. Covid Community Alert - Tech Specs, abr. 2020. Disponível em: https://github.com/Coronavirus-Outbreak-

Control/Documentation/blob/master/Covid%20Community%20Alert%20-%20Protocol%20Decription.pdf. Acesso em: 24 jul. 2022.

30 TELEINFORMÁTICA E AUTOMAÇÃO DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, Grupo de. Bluetooth Technology Overview. **https://gta.ufrj.br**/. Disponível em: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/?utm_source=internal&utm_medium=blog&utm_campaign=technical&utm_content=the-bluetooth-low-energy-primer. Acesso em: 28 out. 2022.

31 TIPOS de Parâmetros nas requisições REST. Disponível em: https://blog.rocketseat.com.br/tipos-de-parametros-nas-requisicoes-rest/. Acesso em: 27 nov. 2022.

32 TYPES of beacons. **Wikipedia**. Disponível em: https://en.wikipedia.org/wiki/Types_of_beacons. Acesso em: 28 out. 2022.

33 URL Loading System. **Apple Inc.** Disponível em: https://developer.apple.com/documentation/foundation/url_loading_system. Acesso em: 27 nov. 2022.

34 WHITELAW, Sera; MAMAS, Mamas A; TOPOL, Eric; VAN SPALL, Harriette G C. Applications of digital technology in COVID-19 pandemic planning and response. **The Lancet Digital Health**, v. 2, n. 8, e435–e440, 2020. ISSN 2589-7500. DOI: https://doi.org/10.1016/S2589-7500(20)30142-4. Disponível em: https://www.sciencedirect.com/science/article/pii/S2589750020301424. Acesso em: 24 jul. 2022.

APÊNDICE A – RELATÓRIO NO FORMATO SBC

CheckIn@UFSC: Aplicativo iOS para a detecção de exposição à COVID-19 nas dependências da UFSC

Pedro A. F. Rodrigues[1], **Jônata T. Carvalho**[1], **Mateus Grellert**[1]

[1]Universidade Federal de Santa Catarina (UFSC)
Departamento de Informatica e Estatística — INE, Florianopolis, SC — Brasil

peafreddi@gmail, jonata.tyska@ufsc.br, mateus.grellert@ufsc.br

Abstract. *This paper presents the development of an iOS app in partnership with the CheckIn@UFSC research project. The app allows the identification of contact with other people who may have been infected by COVID-19 using the anonymous tracking protocol and open source Covid Community Alert. The main functionalities of the application will be: monitor nearby devices that have the application using Bluetooth connection, allow the user to submit your self-diagnosis, and notify you in case of contact with another user suspected of infection or even infected.*

Resumo. *Este trabalho apresenta o desenvolvimento de um aplicativo para o sistema operacional iOS em parceria com o projeto de pesquisa CheckIn@UFSC, que permita realizar a identificação de contato com outras pessoas que possam ter sido infectadas pelo COVID-19 utilizando o protocolo de rastreamento anônimo e open source Covid Community Alert. As principais funcionalidades do aplicativo serão: monitorar dispositivos próximos que possuam o aplicativo usando conexão bluetooth, permitir que o usuário realize o autodiagnóstico no aplicativo e notificá-lo caso aconteça contato com outro usuário suspeito de infecção ou mesmo infectado.*

1. Introdução

De acordo com dados cronológicos [Malik et al. 2020] se passaram 90 dias entre o primeiro surto de COVID-19 e a declaração de Pandemia realizada pela OMS [OMS 2020]. E enquanto a doença se espalhava de rapidamente pelo mundo, o rastreamento de contatos, um dos principais processos para a quebra da cadeia de contaminação pelo vírus, era realizado de maneira manual e totalmente dependente de equipes especializadas para sua realização.

Com esse contexto, nasce a iniciativa Covid Community Alert que cria um protocolo preparado para atender uma escala nacional e que faz o rastreamento de contatos que utilizando o Bluetooth Low Energy e realiza o registro de contatos digital e anonimamente. Esse protocolo serviu de base para o CheckIn@UFSC, que pretende utilizá-lo para o monitoramento de contatos nos campi da Universidade Federal de Santa Catarina.

2. Metodologia

Foi realizado um levantamento de trabalhos e publicações de cunho científico relacionadas ao tema do projeto, buscando entender qual era o atual panorama desse tópico.

Outro levantamento realizado foi sobre informações técnicas acerca do protocolo Covid Community Alert que pudessem dar suporte ao desenvolvimento do aplicativo iOS.

Também foi feita a elicitação de requisitos de desenvolvimento utilizando o aplicativo Android que havia sido desenvolvido pela equipe do CheckIn@UFSC como base para o desenvolvimento do aplicativo iOS. E execução de testes de comunicação do aplicativo iOS em ambiente controlado, buscando aferir se o registro de contatos próximos acontece quando o dispositivo encontra-se em primeiro plano.

3. Desenvolvimento

3.1. Aplicativo Android

Como o projeto já contava com um aplicativo Android desenvolvido, ele foi utilizado como base para o desenvolvimento da aplicação iOS. O aplicativo contava com dois fluxos de interfaces, sendo um deles o de interfaces de introdução ao aplicativo como mostra a figura 1:

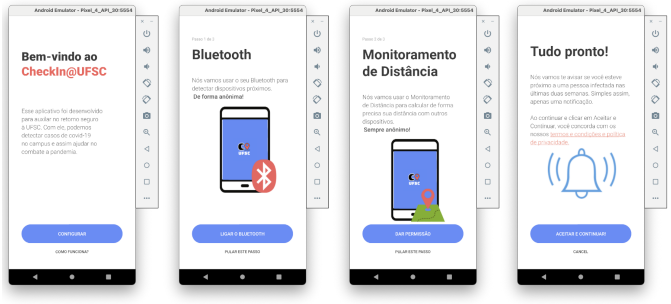


Figura 1. Telas do fluxo de introdução

E o outro fluxo contendo as interfaces da área autenticada do aplicativo, presentes na figura 2:

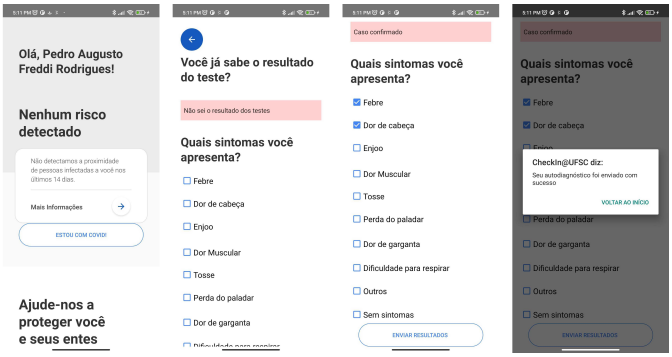


Figura 2. Telas do fluxo de introdução

Usando essas interfaces foram elicitados requisitos para o aplicativo iOS, sendo eles:

Tabela 1. Requisitos elicitados com base no aplicativo Android

REQ-001	Validar o usuário por meio de *Captcha*
REQ-002	Solicitar a permissão de uso do *bluetooth* ao usuário
REQ-003	Solicitar a permissão de uso da localização ao usuário
REQ-004	Realizar a autenticação com o CAS UFSC
REQ-005	Exibir a situação de exposição do usuário
REQ-006	Permitir que o usuário envie seu autodiagnóstico
REQ-007	Exibir mais informações sobre seu funcionamento
REQ-008	Permitir que o usuário compartilhe o aplicativo para que mais pessoas possam ser protegidas
REQ-009	Oferecer informações adicionais sobre a situação de exposição do usuário
REQ-010	Exibir a tela de abertura com a logo do CheckIn@UFSC
REQ-011	Salvar os dados da sessão do usuário
REQ-012	Exibir notificações de *push* quando o usuário estiver em risco
REQ-013	Ao ser colocado em segundo plano, deve escanear dispositivos e transmitir sua presença para dispositivos nas redondezas
REQ-014	Enviar dados para a infraestrutura do CheckIn@UFSC
REQ-015	Mostrar ao usuário quando há alguma permissão pendente que impeça o funcionamento do aplicativo

Com base nesses requisitos, notou-se a possibilidade de explorar uma alternativa de desenvolvimento multiplataforma para o desenvolvimento do aplicativo usando Flutter, uma SDK de desenvolvimento multiplataforma que poderia reduzir as bases de código iOS e Android para apenas uma. Porém, ao realizar a investigação de viabilidade do uso dessa ferramenta não foi possível manter o aplicativo em segundo plano e seguiu-se com o desenvolvimento nativo do aplicativo.

3.2. Desenvolvimento do aplicativo iOS

A equipe do projeto Covid Community Alert já possuía uma implementação de aplicativo iOS, portanto, optou-se por utilizar o código existente, assim como já havia sido feito com o aplicativo Android. Ao compilar o código foram encontrados os mesmos dois fluxos de interfaces presentes no aplicativo Android, mas com algumas diferenças visuais como cores, imagens e ausência de interfaces implementadas apenas no aplicativo Android CheckIn@UFSC.

Dessa forma, foi criada uma lista de tarefas com base nas diferenças encontradas entre os aplicativos iOS e Android descritas na tabela 2:

Tabela 2. Tarefas encontradas com base nas diferenças entre os aplicativos iOS e Android.

CHK-001	Atualizar o README do projeto: Inserir informações e imagens atualizadas sobre projeto, os passos necessários para colocar o aplicativo em funcionamento e outras informações de desenvolvimento que sejam relevantes
CHK-002	Atualizar as imagens do aplicativo para as correspondentes ao CheckIn@UFSC: Substituir os arquivos de imagem referentes à identidade visual do Covid Community Alert para os do CheckIn@UFSC
CHK-003	Criar documentação da classe IBeaconManager descrevendo o que cada componente dessa classe realiza.
CHK-004	Criar documentação da classe StorageManager descrevendo o que cada componente dessa classe realiza.
CHK-005	Criar documentação da classe BackgroundManager descrevendo o que cada componente dessa classe realiza.
CHK-006	Criar documentação da classe BluetoothManager descrevendo o que cada componente dessa classe realiza.
CHK-007	Implementar a interface de autodiagnóstico seguindo a implementação realizada no aplicativo *Android*
CHK-008	Implementar a integração do autodiagnóstico com o *backend* do CheckIn@UFSC
CHK-009	Implementar a interface de autenticação com os sistemas da Universidade
CHK-009	Integrar a interface de autenticação com os sistemas da Universidade
CHK-011	Implementar ajustes na localização para que ela apresente o idioma correto com base na região do dispositivo

Essas tarefas foram desenvolvidas, o que resultou em uma versão do aplicativo totalmente integrado com a infraestrutura do CheckIn@UFSC e assim sendo prosseguiu-se com a testagem do aplicativo.

4. Testes

Para a realização dos testes do aplicativo iOS foi usado como base os testes realizados pela equipe de desenvolvimento do aplicativo Android. A tabela a seguir lista os casos de teste que foram executados:

Tabela 3. Casos de teste

Cód.	Título
CT-1	Usuário realiza *onboarding*
CT-2	Usuário interrompe *onboarding*
CT-3	Usuário pula as etapas do *onboarding*
CT-4	Usuário obtém mais informações sobre sua situação
CT-5	Usuário consegue compartilhar o aplicativo
CT-6	Usuário obtém mais informações sobre como funciona o aplicativo
CT-7	Usuário reativa as permissões caso necessário
CT-8	As interações são registradas corretamente no banco de dados
CT-9	Usuário consegue enviar o autodiagnóstico com sucesso
CT-10	Os autodiagnósticos enviados são registrados corretamente no banco de dados

5. Conclusões

Com base no resultado dos testes pode-se concluir que o aplicativo iOS possui interface de usuário funcional, pois nos fluxos de introdução e o principal o usuário conseguiu alcançar os objetivos traçados nos casos de testes de 1 a 7. Já com relação ao caso de teste 8, foi possível notar que as interações detectadas por dispositivos iOS adotaram um padrão breve de aproximadamente 10 segundos, embora nem mesmo as interações Android tenham apresentado tempo de detecção equivalente ao tempo testado.

Parte das leituras possuíam uma estimação de distância próxima da realidade, mas com tempo inferior ao intervalo de tempo testado, indicando que provavelmente os aplicativos não conseguiram manter a execução em segundo plano por alguma restrição do sistema operacional, como, por exemplo, políticas de economia de energia.

Dessa forma, é possível concluir que o aplicativo possui potencial para servir de ferramenta de rastreamento de contatos, sendo necessárias melhorias em seu funcionamento em segundo plano, ou mesmo adoção de alguma estratégia que faça com que o usuário interaja com o aplicativo, fazendo com que ele execute em primeiro plano.

Referências

(2020). Oms afirma que covid-19 é agora caracterizada como pandemia.

Malik, Y. S., Kumar, N., Sircar, S., Kaushik, R., Bhat, S., Dhama, K., Gupta, P., Goyal, K., Singh, M. P., Ghoshal, U., El Zowalaty, M. E., O. R, V., Yatoo, M. I., Tiwari, R., Pathak, M., Patel, S. K., Sah, R., Rodriguez-Morales, A. J., Ganesh, B., Kumar, P., and Singh, R. K. (2020). Coronavirus disease pandemic (covid-19): Challenges and a global perspective. *Pathogens*, 9(7).

APÊNDICE B – CÓDIGO FONTE

```
//
//  AppDelegate.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 23/02/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit
import BackgroundTasks

// background fetch:
https://www.hackingwithswift.com/example-code/system/how-to-ru
n-code-when-your-app-is-terminated
//
https://medium.com/snowdog-labs/managing-background-tasks-with
-new-task-scheduler-in-ios-13-aaabdac0d95b
// scrollview:
https://fluffy.es/scrollview-storyboard-xcode-11/

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    let rootRouter = RootRouter.initModule()

    static let shared = UIApplication.shared.delegate as!
AppDelegate

    private func registerListeners() {
        BackgroundManager.backgroundOperations()
        if BluetoothManager.shared.isBluetoothUsable() &&
LocationManager.shared.getPermessionStatus() == .allowedAlways
{
            print("restarting ibeacon will resign")
            IBeaconManager.shared.registerListener()
            LocationManager.shared.startMonitoring()
```

```
        }
    }

    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application
launch.

        self.startup()

        // MARK: Registering Launch Handlers for Tasks
        if #available(iOS 13.0, *) {
            BGTaskScheduler.shared.register(
            forTaskWithIdentifier:
Constants.Setup.backgroundPushIdentifier,
            using: DispatchQueue.global()) {
                task in
                // Downcast the parameter to an app refresh
task as this identifier is used for a refresh request.
                print("HANDLE PUSH scheduled called")
                self.handlePushInteractions(task)
            }
        } else {
            // Fallback on earlier versions
        }

        self.registerListeners()

        return true
    }

    @available(iOS 13.0, *)
    func schedulePushInteractions() {
        BGTaskScheduler.shared.cancelAllTaskRequests()
        let request = BGAppRefreshTaskRequest(identifier:
Constants.Setup.backgroundPushIdentifier)
```

```
        // Push again no earlier than 15 minutes (15min *
60sec = 900sec) from now
        request.earliestBeginDate = Date(timeIntervalSinceNow:
900)
        do {
           try BGTaskScheduler.shared.submit(request)
            print("Push scheduled!")
        } catch {
           print("Could not schedule push: \(error)")
        }
    }

    @available(iOS 13.0, *)
    func handlePushInteractions(_ task: BGTask) {
        print("Processing scheduled push!")
        // Schedule a new refresh task

        BackgroundManager.backgroundOperations()
        task.expirationHandler = {}

        _ = Timer.scheduledTimer(withTimeInterval: 2, repeats:
false) {
            _ in
            print("Timer fired!")
            task.setTaskCompleted(success: true)
            self.schedulePushInteractions()
        }

        self.schedulePushInteractions()
    }

    private func startup() {

        NotificationManager.shared.getStatus()
        if StorageManager.shared.getIdentifierDevice() == nil
{
            print("Device not yet registered")
        } else {
```

```
            print("MY ID:",
StorageManager.shared.getIdentifierDevice())
            print("TOKEN:", StorageManager.shared.getToken())
            print("Token ID",
StorageManager.shared.getPushId())
        }

        NotificationManager.shared.getAuthorizationStatus({
            status in
            if status == .allowed {
                DispatchQueue.main.async {
                    print("FORCE REGISTER PUSH")

UIApplication.shared.registerForRemoteNotifications()
                }
            }
        })


UIApplication.shared.setMinimumBackgroundFetchInterval(UIAppli
cation.backgroundFetchIntervalMinimum)

        rootRouter.presentInitialScreen()
    }

    func application(_ application: UIApplication,
performFetchWithCompletionHandler completionHandler: @escaping
(UIBackgroundFetchResult) -> Void) {
        print("background fetchs")
        BackgroundManager.backgroundOperations()
        completionHandler(.newData)
    }

    func application(_ application: UIApplication,
willFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]? = nil) -> Bool {
        print("will finish launch with options")
        BackgroundManager.backgroundOperations()
```

```
        return true
    }

    func applicationWillResignActive(_ application:
UIApplication) {
        // Sent when the application is about to move from
active to inactive state. This can occur for certain types of
temporary interruptions (such as an incoming phone call or SMS
message) or when the user quits the application and it begins
the transition to the background state.
        // Use this method to pause ongoing tasks, disable
timers, and invalidate graphics rendering callbacks. Games
should use this method to pause the game.
        self.registerListeners()
        BackgroundManager.backgroundOperations()
    }

    func applicationDidEnterBackground(_ application:
UIApplication) {
        // Use this method to release shared resources, save
user data, invalidate timers, and store enough application
state information to restore your application to its current
state in case it is terminated later.
        // If your application supports background execution,
this method is called instead of applicationWillTerminate:
when the user quits.
        BackgroundManager.backgroundOperations()
        if #available(iOS 13.0, *) {
            self.schedulePushInteractions()
        } else {
            // Fallback on earlier versions
        }
    }

    func applicationWillEnterForeground(_ application:
UIApplication) {
```

```
        // Called as part of the transition from the
background to the active state; here you can undo many of the
changes made on entering the background.
    }

    func applicationDidBecomeActive(_ application:
UIApplication) {
        // Restart any tasks that were paused (or not yet
started) while the application was inactive. If the
application was previously in the background, optionally
refresh the user interface.
        print("did become active")
        NotificationManager.shared.getStatus()
        BackgroundManager.backgroundOperations()
    }

    func applicationWillTerminate(_ application:
UIApplication) {
        // Called when the application is about to terminate.
Save data if appropriate. See also
applicationDidEnterBackground:.
        // TODO:
        print("WILL TERMINATE")
        BackgroundManager.backgroundOperations()
        StorageManager.shared.saveContext()
        self.registerListeners()

LocationManager.shared.locationManager.startUpdatingLocation()
    }

    private func updatePushId(token: String) {
        if let idDevice =
StorageManager.shared.getIdentifierDevice() {
            AuthManager.RegisterDevice().request { result in
                switch result {
                case .failure(let error):
                    print("Error registering device", error)
                case .success(let handshakeResponse):
```

```
StorageManager.shared.setToken(handshakeResponse.token)

StorageManager.shared.setIdentifierDevice(handshakeResponse.id
)


AuthManager.RegisterPushToken(PushTokenParameters(id:
String(idDevice), pushId: handshakeResponse.token, platform:
"iOS")).request { result in
                        switch result {
                        case .failure(let error):
                            print("Error registering push
token", error)
                        case .success(_):
                            print("Sucess registering push
id")
                        }
                    }
                }
            }
        }
        StorageManager.shared.setPushId(token)
    }

    func application(_ application: UIApplication,
didRegisterForRemoteNotificationsWithDeviceToken deviceToken:
Data) {
        print("XXXXXXXXXXXXXXXXXX")
        // 1. Convert device token to string
        let tokenParts = deviceToken.map { data -> String in
            return String(format: "%02.2hhx", data)
        }
        let token = tokenParts.joined()
        // 2. Print device token to use for PNs payloads
        print("Device Token: \(token)")
        let bundleID = Bundle.main.bundleIdentifier
        print("Bundle ID: \(token) \(bundleID)")
```

```
        if let oldPushId = StorageManager.shared.getPushId() {
            if oldPushId != token {
                updatePushId(token: token)
            }
        } else {
            updatePushId(token: token)
        }
    }

    func application(_ application: UIApplication,
didFailToRegisterForRemoteNotificationsWithError error: Error)
{
        print("failed to register for remote notifications:
\(error.localizedDescription)")
    }

    func application(_ application: UIApplication,
didReceiveRemoteNotification userInfo: [AnyHashable: Any]) {
        print("REMOTE NOTIFICATION - normal")
        self.handleRemoteContent(userInfo)
    }

    func application(_ application: UIApplication,
didReceiveRemoteNotification userInfo: [AnyHashable: Any],
fetchCompletionHandler completionHandler: @escaping
(UIBackgroundFetchResult) -> Void) {

        BackgroundManager.backgroundOperations()
        print("REMOTE NOTIFICATION - fetch")
        self.handleRemoteContent(userInfo)
        completionHandler(.newData)
    }

    private func handleRemoteContent(_ userInfo: [AnyHashable:
Any]) {
        registerListeners()

        print("Received push notification: \(userInfo)")
```

```
        if let d = userInfo["data"] as? [String: Any] {
            print("DATA FOUND", d)
            // avoid updating status

            PushNotificationData.saveNotificationData(d)

            if let status = d["status"] as? Int {
                StorageManager.shared.setStatusUser(status)
            }

            if let warningLevel = d["warning_level"] as? Int {

StorageManager.shared.setWarningLevel(warningLevel)
            }

            NotificationCenter.default.post(name:
NSNotification.Name(Constants.Notification.patientChangeStatus
), object: nil)

            if let title = d["title"] as? String {
                print("title", title)
                let subtitle = d["subtitle"] as? String
                if let message = d["message"] as? String {
                    print("message", message)

NotificationManager.shared.showLocalNotification(title,
subtitle: subtitle, message: message)
                }
            }
        }
    }

    func application(_ application: UIApplication,
                     open url: URL,
                     options:
[UIApplication.OpenURLOptionsKey: Any] = [:] ) -> Bool {
```

```
        guard let components = NSURLComponents(url: url,
resolvingAgainstBaseURL: true),
            let host = components.host,
            let params = components.queryItems else {
                print("Invalid URL or path missing")
                return false
        }

        if let codeIndex = params.first(where: { $0.name ==
"code" })?.value {
            print("host: ", host)
            print("code: ", codeIndex)

            AuthManager.RequestAccessToken(code:
codeIndex).request { [weak self] result in
                switch result {
                case .failure(let error):
                    print("ERROR: ", error)
                case .success(let token):
                    print("TOKEN: ", token)
                    StorageManager.shared.setUFSCToken(token)

AuthManager.shared.requestUserBond(accessToken:
token.accessToken)
                    AuthManager.RequestUserData(accessToken:
token.accessToken).request { result in
                        switch result {
                        case .failure(let error):
                            print("ERROR: ", error)
                        case .success(let user):

StorageManager.shared.setUser(user)

self?.rootRouter.presentInitialScreen()
                        }
                    }
                }
            }
```

```
            return true
        } else {
            print("URL not redirected from auth")
            return false
        }
    }

}

//
//  RetryHandshake.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 25/11/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation
import Alamofire


class RetryHandshake: RequestInterceptor {
    func retry(_ request: Alamofire.Request, for session:
Alamofire.Session, dueTo error: Error, completion: @escaping
(Alamofire.RetryResult) -> Void) {
        print("CALLING RETRY")
        guard request.retryCount < 1 else {
                completion(.doNotRetry)
                return
        }

        if request.response?.statusCode == 401 {
            AuthManager.RegisterDevice().request { result in
                switch result {
                case .failure(let error):
                    print("Error updating token: ",
error.localizedDescription)
```

```
                    completion(.doNotRetry)
                case .success(let handshakeResponse):
                    print("HANDSHAKE: ", handshakeResponse)

StorageManager.shared.setToken(handshakeResponse.token)

StorageManager.shared.setIdentifierDevice(handshakeResponse.id
)
                    completion(.retry)
                }
            }
        }
    }
}

//
//  APIRequestable.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 14/08/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation
import Alamofire

protocol APIRequestable {
    associatedtype APIResponse: Decodable

    var dateDecodingStrategy: JSONDecoder.DateDecodingStrategy
{ get }
    var headers: HTTPHeaders? { get }
    var method: HTTPMethod { get }
    var parameters: Parameters? { get }
    var url: String { get set }
    var interceptor: RequestInterceptor? { get set }
```

```
    func convertResponse(from data: Decodable) -> APIResponse?
    func request(completion: @escaping
(APIResult<APIResponse>) -> Void) -> Request?
}

extension APIRequestable {
    public var dateDecodingStrategy:
JSONDecoder.DateDecodingStrategy {
        return JSONDecoder.DateDecodingStrategy.iso8601
    }

    func convertResponse(from data: Decodable) -> APIResponse?
{
        return data as? APIResponse
    }

    @discardableResult
    func request(completion: @escaping
(APIResult<APIResponse>) -> Void) -> Request? {
        let encoding: ParameterEncoding = method == .get ?
URLEncoding.default : JSONEncoding.default

        return AF.request(
            url,
            method: method,
            parameters: parameters,
            encoding: encoding,
            headers: headers,
            interceptor: interceptor
        ).validate().responseData { response in
            guard (response.error as NSError?)?.code !=
NSURLErrorCancelled else {
                debugPrint("[REQUEST ERROR] Cancelled")
                completion(.failure(.cancelled(response)))
                return
            }
```

```
            guard let statusCode =
response.response?.statusCode else {
                debugPrint("[REQUEST ERROR] Could not get
status code")
                completion(.failure(.noStatusCode(response)))
                return
            }

            guard 200...299 ~= statusCode else {
                debugPrint("[REQUEST ERROR] Status code out of
200...299")
                completion(.failure(.statusCode(statusCode,
response: response)))
                return
            }

            if statusCode == 204 {
                let emptyResponse = APIResponseEmpty()
                guard let emptyResponseObj = emptyResponse as?
APIResponse else {
                    debugPrint("[REQUEST ERROR] Could not
decode",
                               "Return type on API Request
must be of type APIResponseEmpty if response is empty"
                    )

completion(.failure(.parsingFailed(response)))
                    return
                }
                completion(.success(emptyResponseObj))
                return
            }

            guard let data = response.data else {
                if statusCode == 200, response.request?.method
== .put {
                    let emptyResponse = APIResponseEmpty()
```

```
                    guard let emptyResponseObj = emptyResponse
as? APIResponse else {
                        debugPrint("[REQUEST ERROR] Could not
decode",
                                   "Return type on API Request
must be of type APIResponseEmpty if response is empty"
                        )

completion(.failure(.parsingFailed(response)))
                        return
                    }
                    completion(.success(emptyResponseObj))
                    return
                } else {
                    debugPrint("[REQUEST ERROR] Could not get
data")
                    completion(.failure(.noData(response)))
                    return
                }
            }

            do {
                let responseAsObject = try
APIResponse.decoded(
                    from: data,
                    dateDecodingStrategy:
self.dateDecodingStrategy
                )
                guard let convertedResponseAsObject =
self.convertResponse(from: responseAsObject) else {
                    debugPrint("[REQUEST ERROR] Could not
convert response")

completion(.failure(.convertRespose(response)))
                    return
                }

completion(.success(convertedResponseAsObject))
```

```
            } catch let error as DecodingError {
                let emptyResponse = APIResponseEmpty()
                guard let emptyResponseObj = emptyResponse as?
APIResponse else {
                    debugPrint("[REQUEST ERROR] Could not
decode",
                        "Return type on API Request must be of
type APIResponseEmpty if response is empty"
                    )
                    completion(.failure(.objectMapping(error,
response)))
                    return
                }
                completion(.success(emptyResponseObj))
                return
            } catch {
                debugPrint("[REQUEST ERROR] Could not decode",
error)
                completion(.failure(.objectMapping(error,
response)))
                return
            }
        }
    }
}

//
//  APIRequestable.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 14/08/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation
import Alamofire
```

```
protocol APIRequestable {
    associatedtype APIResponse: Decodable

    var dateDecodingStrategy: JSONDecoder.DateDecodingStrategy
{ get }
    var headers: HTTPHeaders? { get }
    var method: HTTPMethod { get }
    var parameters: Parameters? { get }
    var url: String { get set }
    var interceptor: RequestInterceptor? { get set }

    func convertResponse(from data: Decodable) -> APIResponse?
    func request(completion: @escaping
(APIResult<APIResponse>) -> Void) -> Request?
}

extension APIRequestable {
    public var dateDecodingStrategy:
JSONDecoder.DateDecodingStrategy {
        return JSONDecoder.DateDecodingStrategy.iso8601
    }

    func convertResponse(from data: Decodable) -> APIResponse?
{
        return data as? APIResponse
    }

    @discardableResult
    func request(completion: @escaping
(APIResult<APIResponse>) -> Void) -> Request? {
        let encoding: ParameterEncoding = method == .get ?
URLEncoding.default : JSONEncoding.default

        return AF.request(
            url,
            method: method,
            parameters: parameters,
            encoding: encoding,
```

```
            headers: headers,
            interceptor: interceptor
        ).validate().responseData { response in
            guard (response.error as NSError?)?.code !=
NSURLErrorCancelled else {
                debugPrint("[REQUEST ERROR] Cancelled")
                completion(.failure(.cancelled(response)))
                return
            }

            guard let statusCode =
response.response?.statusCode else {
                debugPrint("[REQUEST ERROR] Could not get
status code")
                completion(.failure(.noStatusCode(response)))
                return
            }

            guard 200...299 ~= statusCode else {
                debugPrint("[REQUEST ERROR] Status code out of
200...299")
                completion(.failure(.statusCode(statusCode,
response: response)))
                return
            }

            if statusCode == 204 {
                let emptyResponse = APIResponseEmpty()
                guard let emptyResponseObj = emptyResponse as?
APIResponse else {
                    debugPrint("[REQUEST ERROR] Could not
decode",
                               "Return type on API Request
must be of type APIResponseEmpty if response is empty"
                    )

completion(.failure(.parsingFailed(response)))
                    return
```

```
                }
                completion(.success(emptyResponseObj))
                return
            }

            guard let data = response.data else {
                if statusCode == 200, response.request?.method
== .put {
                    let emptyResponse = APIResponseEmpty()
                    guard let emptyResponseObj = emptyResponse
as? APIResponse else {
                        debugPrint("[REQUEST ERROR] Could not
decode",
                                   "Return type on API Request
must be of type APIResponseEmpty if response is empty"
                        )

completion(.failure(.parsingFailed(response)))
                        return
                    }
                    completion(.success(emptyResponseObj))
                    return
                } else {
                    debugPrint("[REQUEST ERROR] Could not get
data")
                    completion(.failure(.noData(response)))
                    return
                }
            }

            do {
                let responseAsObject = try
APIResponse.decoded(
                    from: data,
                    dateDecodingStrategy:
self.dateDecodingStrategy
                )
```

```
                guard let convertedResponseAsObject =
self.convertResponse(from: responseAsObject) else {
                    debugPrint("[REQUEST ERROR] Could not
convert response")

completion(.failure(.convertRespose(response)))
                    return
                }

completion(.success(convertedResponseAsObject))
            } catch let error as DecodingError {
                let emptyResponse = APIResponseEmpty()
                guard let emptyResponseObj = emptyResponse as?
APIResponse else {
                    debugPrint("[REQUEST ERROR] Could not
decode",
                        "Return type on API Request must be of
type APIResponseEmpty if response is empty"
                    )
                    completion(.failure(.objectMapping(error,
response)))
                    return
                }
                completion(.success(emptyResponseObj))
                return
            } catch {
                debugPrint("[REQUEST ERROR] Could not decode",
error)
                completion(.failure(.objectMapping(error,
response)))
                return
            }
        }
    }
}

//
//  APIRequestable.swift
```

```
//  CheckInUFSC
//
//  Created by Pedro Freddi on 14/08/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation
import Alamofire

protocol APIRequestable {
    associatedtype APIResponse: Decodable

    var dateDecodingStrategy: JSONDecoder.DateDecodingStrategy
{ get }
    var headers: HTTPHeaders? { get }
    var method: HTTPMethod { get }
    var parameters: Parameters? { get }
    var url: String { get set }
    var interceptor: RequestInterceptor? { get set }

    func convertResponse(from data: Decodable) -> APIResponse?
    func request(completion: @escaping
(APIResult<APIResponse>) -> Void) -> Request?
}

extension APIRequestable {
    public var dateDecodingStrategy:
JSONDecoder.DateDecodingStrategy {
        return JSONDecoder.DateDecodingStrategy.iso8601
    }

    func convertResponse(from data: Decodable) -> APIResponse?
{
        return data as? APIResponse
    }

    @discardableResult
```

```
    func request(completion: @escaping
(APIResult<APIResponse>) -> Void) -> Request? {
        let encoding: ParameterEncoding = method == .get ?
URLEncoding.default : JSONEncoding.default

        return AF.request(
            url,
            method: method,
            parameters: parameters,
            encoding: encoding,
            headers: headers,
            interceptor: interceptor
        ).validate().responseData { response in
            guard (response.error as NSError?)?.code !=
NSURLErrorCancelled else {
                debugPrint("[REQUEST ERROR] Cancelled")
                completion(.failure(.cancelled(response)))
                return
            }

            guard let statusCode =
response.response?.statusCode else {
                debugPrint("[REQUEST ERROR] Could not get
status code")
                completion(.failure(.noStatusCode(response)))
                return
            }

            guard 200...299 ~= statusCode else {
                debugPrint("[REQUEST ERROR] Status code out of
200...299")
                completion(.failure(.statusCode(statusCode,
response: response)))
                return
            }

            if statusCode == 204 {
                let emptyResponse = APIResponseEmpty()
```

```
                guard let emptyResponseObj = emptyResponse as?
APIResponse else {
                    debugPrint("[REQUEST ERROR] Could not
decode",
                               "Return type on API Request
must be of type APIResponseEmpty if response is empty"
                    )

completion(.failure(.parsingFailed(response)))
                    return
                }
                completion(.success(emptyResponseObj))
                return
            }

            guard let data = response.data else {
                if statusCode == 200, response.request?.method
== .put {
                    let emptyResponse = APIResponseEmpty()
                    guard let emptyResponseObj = emptyResponse
as? APIResponse else {
                        debugPrint("[REQUEST ERROR] Could not
decode",
                                   "Return type on API Request
must be of type APIResponseEmpty if response is empty"
                        )

completion(.failure(.parsingFailed(response)))
                        return
                    }
                    completion(.success(emptyResponseObj))
                    return
                } else {
                    debugPrint("[REQUEST ERROR] Could not get
data")
                    completion(.failure(.noData(response)))
                    return
                }
```

```
            }

            do {
                let responseAsObject = try
APIResponse.decoded(
                    from: data,
                    dateDecodingStrategy:
self.dateDecodingStrategy
                )
                guard let convertedResponseAsObject =
self.convertResponse(from: responseAsObject) else {
                    debugPrint("[REQUEST ERROR] Could not
convert response")

completion(.failure(.convertRespose(response)))
                    return
                }

completion(.success(convertedResponseAsObject))
            } catch let error as DecodingError {
                let emptyResponse = APIResponseEmpty()
                guard let emptyResponseObj = emptyResponse as?
APIResponse else {
                    debugPrint("[REQUEST ERROR] Could not
decode",
                        "Return type on API Request must be of
type APIResponseEmpty if response is empty"
                    )
                    completion(.failure(.objectMapping(error,
response)))
                    return
                }
                completion(.success(emptyResponseObj))
                return
            } catch {
                debugPrint("[REQUEST ERROR] Could not decode",
error)
```

```
                completion(.failure(.objectMapping(error,
response)))
                return
            }
        }
    }
}

//
//  PeripheralDto.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 29/02/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import Foundation
import CoreLocation

class IBeaconDto: Codable, CustomDebugStringConvertible {
    public var timestamp: Date
    public var identifier: Int64
    public var rssi: Int64
    public var interval: Double
    public var platform: String
    public var distance: Int
    public var accuracy: Double
    public var lat: Double
    public var lon: Double
    public var timestampEnd: Date

    public init(identifier: Int64, timestamp: Date, rssi:
Int64, distance: Int, accuracy: Double, interval: Double =
Constants.Setup.minimumIntervalTime) {
        self.timestamp = timestamp
        self.identifier = identifier
        self.rssi = rssi
        self.interval = interval
```

```
        self.platform = "i"
        self.distance = distance
        self.accuracy = accuracy
        self.lat = 0.0
        self.lon = 0.0
        self.timestampEnd = timestamp
    }

    public init(identifier: Int64, timestamp: Date, rssi:
Int64, distance: Int, accuracy: Double, lat: Double, lon:
Double, interval: Double =
Constants.Setup.minimumIntervalTime) {
        self.timestamp = timestamp
        self.identifier = identifier
        self.rssi = rssi
        self.interval = interval
        self.platform = "i"
        self.distance = distance
        self.accuracy = accuracy
        self.lat = lat
        self.lon = lon
        self.timestampEnd = timestamp
    }

    public func setLocation(_ location: CLLocation) {
        self.lat = location.coordinate.latitude
        self.lon = location.coordinate.longitude
    }

    public func setTimestampEnd(_ timestampEnd: Date) {
        self.timestampEnd = timestampEnd
    }

    var debugDescription: String {
        var d = "f"
        if self.distance == 1 {
            d = "i"
        }
```

```
        if self.distance == 2 {
            d = "n"
        }

        return "id: \(self.identifier), timestamp:
\(self.timestamp), rssi: \(self.rssi), distance: \(d),
interval: \(self.interval), space: \(self.accuracy), endedAt:
\(self.timestampEnd) "
    }

    var asParameters: [String: Any]? {
        return [
            "o": identifier,
            "w": Int(timestamp.timeIntervalSince1970),
            "t": Int(interval),
            "r": abs(rssi) ,
            "s": distance,
//            "x": lat,
//            "y": lon,
            "w_plus_t":
Int(timestampEnd.timeIntervalSince1970)
        ]
    }
}

//
//  PushNotificationData.swift
//  CovidApp - Covid Community Alert
//
//  Created by Antonio Romano on 06/04/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import Foundation
class PushNotificationData {

    private(set) var title: String?
    private(set) var message: String?
```

```
    private(set) var filterId: Int?
    private(set) var link: String?
    private(set) var language: String?
    private(set) var contentLanguage: String?
    private(set) var status: Int?
    private(set) var warningLevel: Int?

    private init() {

    }

    public static func saveNotificationData(_ value: Any) {
        StorageManager.shared.setNotificationData(value)
    }

    public static func readNotificationDate() ->
PushNotificationData? {

        var p: PushNotificationData?

        if let v = StorageManager.shared.getNotificationData()
{
            print("PUSH FOUND", v)
            if let data = v as? [String: AnyObject] {
                p = PushNotificationData()

                if let fid = data["filter_id"] as? Int {
                    p!.filterId = fid
                }

                if let status = data["status"] as? Int {
                    p!.status = status
                }

                if let warning = data["warning_level"] as? Int
{
                    p!.warningLevel = warning
                }
```

```
                if let title = data["title"] as? String {
                    p!.title = title
                }

                if let message = data["message"] as? String {
                    p!.message = message
                }

                if let link = data["link"] as? String {
                    p!.link = link
                }

                if let language = data["language"] as? String
{
                    p!.language = language
                }

                if let content = data["content"] as? [String:
AnyObject] {
                    if let contentLanguage =
content["language"] as? String {
                        p!.contentLanguage = contentLanguage
                    }
                }
            }
        }
        print("TITLE", p?.title, p?.contentLanguage)
        return p
    }

}

//
//  PushNotificationData.swift
//  CovidApp - Covid Community Alert
//
//  Created by Antonio Romano on 06/04/2020.
```

```
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import Foundation
class PushNotificationData {

    private(set) var title: String?
    private(set) var message: String?
    private(set) var filterId: Int?
    private(set) var link: String?
    private(set) var language: String?
    private(set) var contentLanguage: String?
    private(set) var status: Int?
    private(set) var warningLevel: Int?

    private init() {

    }

    public static func saveNotificationData(_ value: Any) {
        StorageManager.shared.setNotificationData(value)
    }

    public static func readNotificationDate() ->
PushNotificationData? {

        var p: PushNotificationData?

        if let v = StorageManager.shared.getNotificationData()
{
            print("PUSH FOUND", v)
            if let data = v as? [String: AnyObject] {
                p = PushNotificationData()

                if let fid = data["filter_id"] as? Int {
                    p!.filterId = fid
                }
```

```
                if let status = data["status"] as? Int {
                    p!.status = status
                }

                if let warning = data["warning_level"] as? Int
{
                    p!.warningLevel = warning
                }

                if let title = data["title"] as? String {
                    p!.title = title
                }

                if let message = data["message"] as? String {
                    p!.message = message
                }

                if let link = data["link"] as? String {
                    p!.link = link
                }

                if let language = data["language"] as? String
{
                    p!.language = language
                }

                if let content = data["content"] as? [String:
AnyObject] {
                    if let contentLanguage =
content["language"] as? String {
                        p!.contentLanguage = contentLanguage
                    }
                }
            }
        }
        print("TITLE", p?.title, p?.contentLanguage)
        return p
    }
```

```
}

//
//  UFSCToken.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 14/08/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation

struct UFSCToken: Codable {
    let accessToken: String
    let expiresIn: Int
    let tokenType: String
}

//
//  DeviceData.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 10/09/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation

struct DeviceData: Codable {
    let id: String
    let challenge: String
    let os: OperationalSystem
    let device: DeviceDescription
}
```

```
struct OperationalSystem: Codable {
    let name: String
    let version: String
}

struct DeviceDescription: Codable {
    let manufacturer: String
    let model: String
}

struct HandshakeResponse: Codable {
    let cache: Bool
    let id: Int
    let token: String
}

//
//  Symptoms.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 18/09/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation

struct SymptomsParameters: Codable {
    let patientId: Int
    let symptoms: String
    let oldStatus: Int
    let actualStatus: Int
}

//
//  Symptoms.swift
//  CheckInUFSC
//
```

```
//  Created by Pedro Freddi on 18/09/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation

struct SymptomsParameters: Codable {
    let patientId: Int
    let symptoms: String
    let oldStatus: Int
    let actualStatus: Int
}

//
//  ApiManager.swift
//  CheckInUFSC
//
//  Created by Neil Kakkar on 02/03/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

// Adapted from
https://developer.apple.com/documentation/foundation/url_loadi
ng_system/downloading_files_in_the_background
//
https://developer.apple.com/documentation/foundation/url_loadi
ng_system/uploading_data_to_a_website
//
https://developer.apple.com/documentation/foundation/urlsessio
n
//
// stagger requests to API to ease load

import Foundation

class ApiManager: NSObject, URLSessionDelegate,
URLSessionTaskDelegate, URLSessionDataDelegate {
```

```
    private let endpoint_string =
"https://api.coronaviruscheck.org"

    private lazy var urlSession: URLSession = {
        let config =
URLSessionConfiguration.background(withIdentifier:
"coronavirus-app")
        config.isDiscretionary = false
        config.sessionSendsLaunchEvents = false
        return URLSession(configuration: config, delegate:
self, delegateQueue: nil)
    }()

    public static let shared = ApiManager()

    private override init() {
        super.init()
    }

    private struct pushResponse: Codable {
        let data: String?
        let next_try: TimeInterval
        let location: Bool?
        let exclude_far: Bool?
    }

    func urlSession(_ session: URLSession, task:
URLSessionTask, didCompleteWithError error: Error?) {
        print("Data upload in background completed")
        if let error = error {
            // error handling
            print(error)
            return
        }
    }
```

```
    func urlSession(_ session: URLSession, dataTask:
URLSessionDataTask, didReceive response: URLResponse,
completionHandler: @escaping (URLSession.ResponseDisposition)
-> Void) {
        completionHandler(.allow)
    }

    func urlSession(_ session: URLSession, dataTask:
URLSessionDataTask, didReceive data: Data) {
        // response background push interaction
        DispatchQueue.main.async {
            do {
                let response = try
JSONDecoder().decode(pushResponse.self, from: data)
                print(response)
                let next_try = response.next_try +
Double.random(in: -0.25 ... 0.25) * response.next_try

StorageManager.shared.setPushInterval(next_try)

                if let b = response.location {
                    StorageManager.shared.setLocationNeeded(b)
                }
                if let ex = response.exclude_far {
//                    StorageManager.shared.setExcludeFar(ex)
                }

                StorageManager.shared.resetPushInProgress()
              } catch let parsingError {
                 print("Error", parsingError)
            }
        }
    }

    public func uploadInteractionsInBackground(_ devices:
[IBeaconDto], token: String) {
```

```
        //
https://medium.com/livefront/uploading-data-in-the-background-
in-ios-f93722013c6a
        // We need to write to tempDir to make things work
here
        //

        if devices.isEmpty {
            print("Ending task. No interactions.")
            return
        }

        if devices.count == 0 {
            StorageManager.shared.resetPushInProgress()
        }

        let endpoint = URL(string:
"\(endpoint_string)/interaction/report")
        var request = URLRequest(url: endpoint!)
        request.httpMethod = "POST"
        request.addValue("Bearer " + token,
forHTTPHeaderField: "Authorization")
        request.setValue("application/json",
forHTTPHeaderField: "Content-Type")

        if let uploadData = generatePayload(devices) {

            let tempDir =
FileManager.default.temporaryDirectory
            let localURL =
tempDir.appendingPathComponent("throwaway")
            try? uploadData.write(to: localURL)

            let backgroundTask = urlSession.uploadTask(with:
request, fromFile: localURL)
            // backgroundTask.earliestBeginDate =
Date().addingTimeInterval(60 * 60)
```

```
            // backgroundTask.countOfBytesClientExpectsToSend
= 200
            //
backgroundTask.countOfBytesClientExpectsToReceive = 500 * 1024
            backgroundTask.resume()
        }
    }

    public func uploadInteractions(_ devices: [IBeaconDto],
token: String, handler: @escaping (TimeInterval) -> Void) {
        print("Upload called")

        if devices.count == 0 {

handler(Constants.Setup.defaultSecondsIntervalBetweenPushes)
        }

        if devices.isEmpty {
            print("Ending task. No interactions.")

handler(Constants.Setup.defaultSecondsIntervalBetweenPushes)
            return
        }

        print("gonna PUSH intereactions")

        let endpoint = URL(string:
"\(endpoint_string)/interaction/report")
        var request = URLRequest(url: endpoint!)
        request.httpMethod = "POST"
        request.addValue("Bearer " + token,
forHTTPHeaderField: "Authorization")
        request.setValue("application/json",
forHTTPHeaderField: "Content-Type")

        if let uploadData = generatePayload(devices) {
            let task = URLSession.shared.uploadTask(with:
request, from: uploadData) { data, response, error in
```

```
                if let error = error {
                    // error handling
                    print(error)
                    return
                }
                guard let httpResponse = response as?
HTTPURLResponse,

(200...299).contains(httpResponse.statusCode) else {
                        // error handling
                        print(response ?? "Unknown server
error")
                        return
                }
                if let data = data {
                    do {
                        let response = try
JSONDecoder().decode(pushResponse.self, from: data)
                        print(response)
                        let next_try = response.next_try +
Double.random(in: -0.25 ... 0.25) * response.next_try

StorageManager.shared.setPushInterval(next_try)
                        if let b = response.location {

StorageManager.shared.setLocationNeeded(b)
                        }
                        if let ex = response.exclude_far {
//
StorageManager.shared.setExcludeFar(ex)
                        }
                        handler(next_try)
                      } catch let parsingError {
                         print("Error", parsingError)
                    }
                }
            }
            task.resume()
```

```
        }
    }

    private func generatePayload(_ devices: [IBeaconDto]) ->
Data? {
        struct Interaction: Codable {
            let i: Int64  // id of this device
            let o: Int64  // id of the interacted device
            let w: Int64  // unix time expressed in seconds
            let t: Int    // time of interaction, default is
10
            let x: Double // longitude
            let y: Double // latitude
            let r: Int64 // rssi value
            let p: String
            let d: String
            let s: Double
            let v: Int
        }

        var payload: [Interaction] = []
        let deviceID =
StorageManager.shared.getIdentifierDevice()!

        for device in devices {
            var distance = "f"
            if device.distance == 1 {
                distance = "i"
            }
            if device.distance == 2 {
                distance = "n"
            }

            let interaction = Interaction(
                i: Int64(deviceID),
                o: device.identifier,
                w:
Int64(device.timestamp.timeIntervalSince1970),
```

```
                t: Int(device.interval),
                x: device.lon,
                y: device.lat,
                r: device.rssi,
                p: device.platform,
                d: distance,
                s: device.accuracy,
                v: Constants.Setup.version)
            payload.append(interaction)
        }

        print("PAYLOAD", payload)

        guard let uploadData = try?
JSONEncoder().encode(payload) else {
            print("Failed to encode interactions")
            return nil
        }
        print("GONNA UPLOAD", uploadData)
        return uploadData

    }

    public func getIsInfected(handler: @escaping (Bool) ->
Void) {
        let url = URL(string: "\(endpoint_string)/infected/")!
        let task = URLSession.shared.dataTask(with: url) {
data, response, error in
            if let error = error {
                // error handling
                print(error)
                return
            }
            guard let httpResponse = response as?
HTTPURLResponse,
                (200...299).contains(httpResponse.statusCode)
else {
                    // error handling
```

```
                    print(response ?? "Unknown server error")
                    return
                }
                if let data = data {
                    DispatchQueue.main.async {
                        do {
                            let jsonResponse = try
JSONSerialization.jsonObject(with: data, options: [])
                            print(jsonResponse)
                            // will probably need some pre
processing first
                            handler(jsonResponse as? Bool ??
false)
                          } catch let parsingError {
                             print("Error", parsingError)
                        }
                    }
                }
            }
            task.resume()
        }

        public func setPushNotificationId(deviceId: Int64,
notificationId: String, token: String) {
            let url = URL(string: "\(endpoint_string)/device")!

            var request = URLRequest(url: url)
            request.httpMethod = "PUT"
            request.addValue("Bearer " + token,
forHTTPHeaderField: "Authorization")
            request.setValue("application/json",
forHTTPHeaderField: "Content-Type")

            struct ApiRequest: Codable {
                let id: Int64
                let push_id: String
                let platform: String
            }
```

```
        print("ID \(deviceId)")
        print("JWT \(token)")
        print("PUSH \(notificationId)")

        let apiRequest = ApiRequest(id: deviceId, push_id:
notificationId, platform: "iOS")

        guard let uploadData = try?
JSONEncoder().encode(apiRequest) else {
            print("Failed to encode request")
            return
        }

        request.httpBody = uploadData

        let task = URLSession.shared.dataTask(with: request) {
_, response, error in
            if let error = error {
                // error handling
                print(error)
                return
            }
            guard let httpResponse = response as?
HTTPURLResponse,
                (200...299).contains(httpResponse.statusCode)
else {
                    // error handling
                    print(response ?? "Unknown server error")
                    return
            }
            print("set push notification ID for device
correctly")
        }
        task.resume()

    }
    // Shouldn't have to call this more than once, ever.
```

```
    public func handshakeNewDevice(googleToken: String?,
handler: @escaping (Int64?, String?, String?) -> Void) {
        let id = DeviceInfoManager.getId()
        let model = DeviceInfoManager.getModel()
        let version = DeviceInfoManager.getVersion()

        let url = URL(string:
"\(endpoint_string)/device/handshake")!
        var request = URLRequest(url: url)
        request.httpMethod = "POST"
        request.setValue("application/json",
forHTTPHeaderField: "Content-Type")

        struct DeviceModel: Codable {
            let manufacturer: String
            let model: String
        }

        struct DeviceOS: Codable {
            let name: String
            let version: String
        }
        struct DeviceInfo: Codable {
            let id: String // unique generated ID
            let device: DeviceModel
            let os: DeviceOS
            let challenge: String?
        }

        struct ApiResponse: Codable {
            let id: Int64
            let token: String
        }

        let deviceModel = DeviceModel(manufacturer: "Apple",
model: model)
        let deviceOS = DeviceOS(name: "iOS", version: version)
```

```
        let deviceInfo = DeviceInfo(id: id, device:
deviceModel, os: deviceOS, challenge: googleToken)
        print(deviceInfo)
        guard let uploadData = try?
JSONEncoder().encode(deviceInfo) else {
            print("Failed to encode deviceInfo")
            handler(nil, nil, "Failed to encode deviceInfo")
            return
        }

        request.httpBody = uploadData

        let task = URLSession.shared.dataTask(with: request) {
data, response, error in
            if let error = error {
                // error handling
                print(error)
                handler(nil, nil, error.localizedDescription)
                return
            }
            guard let httpResponse = response as?
HTTPURLResponse,
                (200...299).contains(httpResponse.statusCode)
else {
                    // error handling
                    print(response ?? "Unknown server error")
                    handler(nil, nil, "Unknown server error")
                    return
            }
            if let dataResponse = data {
                do {
                    let response = try
JSONDecoder().decode(ApiResponse.self, from: dataResponse)
                    print("HANDSHAKE", response)

StorageManager.shared.setTokenJWT(response.token)
                    handler(response.id, response.token, nil)
                  } catch let parsingError {
```

```
                    print("Error", parsingError)
                    handler(nil, nil, "parsing error")
                }
            }
        }
        task.resume()
    }

}

//
//  AlamofireManager.swift
//  CovidApp - Covid Community Alert
//
//  Created by Antonio Romano on 03/04/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import Foundation
import Alamofire

class AlamofireManager {

    static public let shared = AlamofireManager()
    private let endpoint_string =
"https://api.coronaviruscheck.org"

    private init() {

    }

    public func pushInteractions(_ interactions: [IBeaconDto],
token: String) {
        var params = [String: Any]()
        params["p"] = "i"
        params["v"] = Constants.Setup.version
        params["i"] =
StorageManager.shared.getIdentifierDevice()!
```

```
        var its = [[String: Any]]()
        for interaction in interactions {
            var p = [String: Any]()
            var distance = "f"
            if interaction.distance == 1 {
                distance = "i"
            }
            if interaction.distance == 2 {
                distance = "n"
            }
            p["o"] = interaction.identifier
            p["w"] =
Int64(interaction.timestamp.timeIntervalSince1970)
            p["t"] = Int(interaction.interval)
            p["r"] = abs(interaction.rssi)
            p["s"] =
Utils.roundToDecimals(interaction.accuracy, digits: 1)
            p["d"] = distance
            print("LON", interaction.lon)
            if !interaction.lon.isZero {
                p["x"] =
Utils.roundToDecimals(interaction.lon, digits: 3)
                p["y"] =
Utils.roundToDecimals(interaction.lat, digits: 3)
            }
            its.append(p)
        }
        params["z"] = its

        let headers: HTTPHeaders = [
            "Authorization": "Bearer " + token,
            "Content-type": "application/json"
        ]

        let endpoint = "\(endpoint_string)/interaction/report"

        AF.request(endpoint, method: .post, parameters:
params, encoding: JSONEncoding.default, headers: headers)
```

```
            .validate(statusCode: 200..<300)
            .responseJSON {
                response in

                print("ALAMOFIRE RESPONSE", response)
                print(Date())
                print(params)
                switch response.result {
                case .success(let j):
                    print("success", j)

StorageManager.shared.setLastTimePush(interactions[interaction
s.count-1].timestampEnd.addingTimeInterval(Constants.Setup.min
imumIntervalTime))
                    print("last time pushed",
interactions[interactions.count-1].timestampEnd.addingTimeInte
rval(Constants.Setup.minimumIntervalTime))

                    if let json = j as? [String: Any] {
                        print("JSON", json)
                        if let location = json["location"] as?
Bool {

StorageManager.shared.setLocationNeeded(location)
                        }
                        if let nextTry = json["next_try"] as?
Double {
                            let nt =  nextTry +
Double.random(in: -0.25 ... 0.25) * nextTry
                            print("setting next_try as", nt)

StorageManager.shared.setPushInterval(nt)
                        }
                        if let distanceFilter =
json["distance_filter"] as? Double {

StorageManager.shared.setDistanceFilter(distanceFilter)
                        } else {
```

```
StorageManager.shared.removeDistanceFilter()
                        }
                    }

                    break
                case .failure(let e):
                    print("pushInteractions - error upload",
e)
                    break
                }

                StorageManager.shared.resetPushInProgress()
            }
    }

    public func downloadDataNotification(_ url: String,
callback: @escaping((Any, Bool) -> Void)) {

        AF.request(url)
        .responseJSON {
            response in
            print("RESPONE", response)

            switch response.result {
            case .success(let j):
                print("success", j)
                callback(j, true)
                break
            case .failure(let e):
                print("downloadDataNotification - error
upload", e)
                callback(0, false)
                break
            }

        }
```

```
    }

}

//
//  AuthManager.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 09/08/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation
import Alamofire
import UIKit

class AuthManager {

    static public let shared = AuthManager()

    struct RequestAccessToken: APIRequestable {
        typealias APIResponse = UFSCToken
        var method: HTTPMethod = .get
        var parameters: Parameters?
        var url: String
        var headers: HTTPHeaders?
        var interceptor: Alamofire.RequestInterceptor?

        init(code: String) {
            url =
"https://sistemas.ufsc.br/oauth2.0/accessToken?grant_type=auth
orization_code&code=\(code)&client_id=covidapp&redirect_uri=co
vidappufsc%3A%2F%2Fcovidapp.setic_oauth.ufsc.br&client_secret=
EkhivyaicOssyauptOijzatyuefloc"
        }
    }
```

```
    struct RequestUserData: APIRequestable {
        typealias APIResponse = User
        var method: HTTPMethod = .get
        var parameters: Parameters?
        var url: String
        var headers: HTTPHeaders?
        var interceptor: Alamofire.RequestInterceptor?

        init(accessToken: String) {
            url =
"https://sistemas.ufsc.br/oauth2.0/profile?access_token=\(acce
ssToken)"
        }
    }

    // TODO: Review the usage of this request
    public func requestUserBond(accessToken: String) {
        let endpoint = "https://ws.ufsc.br"
        let headers: HTTPHeaders = [
            "Authorization": "Bearer " + accessToken,
            "Content-type": "application/json"
        ]

        AF.request(endpoint, method: .get, encoding:
JSONEncoding.default, headers: headers)
            .validate(statusCode: 200..<300)
            .responseJSON {
                response in
                print("requestUserBond", response)
            }

        AF.request(endpoint, method: .get, encoding:
URLEncoding.default, headers: headers)
            .validate(statusCode: 200..<300)
            .responseString {
                response in
                print("requestUserBond2", response)
                switch response.result {
```

```
                case .success(let j):
                    print("success", j)

                    if let json = j as? [String: Any] {
                        print("JSON", json)
                    }
                case .failure(let e):
                    print("requestUserBond - error upload", e)
                }
        }
    }

    /// This request registers the device on back-end in the
first access of the user
    struct RegisterDevice: APIRequestable {
        typealias APIResponse = HandshakeResponse
        var headers: HTTPHeaders?
        var method: HTTPMethod = .post
        var parameters: Parameters?
        var url: String = Constants.Endpoints.deviceHandshake
        var interceptor: Alamofire.RequestInterceptor?

        /// This request registers the device, and on success
returns a ``HandshakeResponse`` object
        /// - Parameters:
        ///   - params: A struct that stores:
        ///     - id: The device id: A unique key associated
to the device
        ///     - os: The OS name and version
        ///     - device: The device manufacturer and model
        ///     - challenge: The authentication token retrived
from captcha validation
        init() {
            parameters = DeviceData(
                id: StorageManager.shared.getUUID()!,
                challenge:
StorageManager.shared.getCaptchaCode() ?? "",
                os: OperationalSystem(
```

```
                    name: UIDevice.current.systemName,
                    version: UIDevice.current.systemVersion),
                device: DeviceDescription(
                    manufacturer: "Apple",
                    model: UIDevice.modelName)).asParameters
        }
    }

    /// Registers the device push notification id on App
back-end
    struct RegisterPushToken: APIRequestable {
        typealias APIResponse = APIResponseEmpty

        var headers: HTTPHeaders?
        var method: HTTPMethod = .post
        var parameters: Parameters?
        var url: String = Constants.Endpoints.device
        var interceptor: Alamofire.RequestInterceptor?

        /// Sends the device id, the push notification id and
the device platform
        /// - Parameters:
        ///   - params: A struct that stores: the device id,
the push notification id,
        ///   and the device platform
        init(_ params: PushTokenParameters) {
            parameters = params.asParameters
        }
    }
}

//
//  SelfDiagnosisManager.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 11/09/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
```

```
//

import Foundation
import Alamofire

class SelfDiagnosisManager {
    struct SendDiagnosis: APIRequestable {
        typealias APIResponse = APIResponseEmpty
        var method: HTTPMethod = .put
        var parameters: Parameters?
        var url: String = Constants.Endpoints.selfDiagnosis
        var headers: HTTPHeaders?
        var interceptor: Alamofire.RequestInterceptor?

        init(_ params: SymptomsParameters) {
            parameters = params.asParameters
            headers = [
                "Authorization": "Bearer " +
(StorageManager.shared.getToken() ?? "")
            ]
        }
    }
}

//
//  InteractionManager.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 18/09/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import Foundation
import Alamofire

class InteractionManager {
```

```
    struct PushInteractions: APIRequestable {

        var headers: HTTPHeaders?
        var method: HTTPMethod = .post
        var parameters: Parameters?
        var url: String = Constants.Endpoints.pushInteractions
        var interceptor: RequestInterceptor?
        typealias APIResponse = APIResponseEmpty

        init(_ beacons: [IBeaconDto]) {
            guard let deviceId =
StorageManager.shared.getIdentifierDevice() else { return }
            let encodedBeacons = beacons.map({ $0.asParameters
})

            headers = [
                "Authorization": "Bearer " +
(StorageManager.shared.getToken() ?? "")
            ]

            interceptor = RetryHandshake()
            self.parameters = [
                "i" : deviceId,
                "p" : "i",
                "v" : 1,
                "z" : encodedBeacons
            ]
        }
    }
}

//
//  BaseViewController.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 08/07/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
```

```
//

import UIKit
import RxSwift

class BaseViewController: UIViewController {

    // MARK: - Properties

    let disposeBag = DisposeBag()

    override var preferredStatusBarStyle: UIStatusBarStyle {
        if #available(iOS 13.0, *) {
            return .darkContent
        } else {
            // Fallback on earlier versions
            return .default
        }
    }

    // MARK: - Life cycle

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
    }

    deinit {
        print("Deinit of type \(type(of: self))")
    }
    // MARK: - Private

}

//
```

```
//  AuthenticationViewController.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 08/07/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import UIKit

class AuthenticationViewController: BaseViewController,
StoryboardLoadable {

    // MARK: - Factory

    static func initModule(viewModel: AuthenticationViewModel)
-> AuthenticationViewController {
        let viewController = loadFromStoryboard()
        viewController.viewModel = viewModel
        return viewController
    }

    // MARK: - IBOutlets

    @IBOutlet var authenticationButton: RoundButton!

    // MARK: - Properties

    private var viewModel: AuthenticationViewModel! {
          didSet {
              setupObservables()
          }
      }

    // MARK: - Life cycle

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
    }

    // MARK: - Private

    private func setupObservables() {
        authenticationButton.rx.tap.subscribe { [weak self] _
in
            self?.viewModel.handleAuthenticationButtonTap()
        }.disposed(by: disposeBag)
    }
}

//
//  AuthenticationViewModel.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 08/07/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import RxSwift
import Foundation
import UIKit

class AuthenticationViewModel {

    // MARK: - Factory

//    init(withRepository repository: Repository) {
//        self.repository = repository
//    }

    // MARK: - Properties
```

```
    private let disposeBag = DisposeBag()

    func handleAuthenticationButtonTap() {
        print("Handling Tap")

        if let url = URL(string:
"https://sistemas.ufsc.br/oauth2.0/authorize?response_type=cod
e&client_id=covidapp&state=E3ZYKC1T6H2yP4z&redirect_uri=covida
ppufsc%3A%2F%2Fcovidapp.setic_oauth.ufsc.br"),
UIApplication.shared.canOpenURL(url) {
            UIApplication.shared.open(url)
        }
    }
}

//
//  NewSymptomViewController.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 26/06/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import UIKit
import RxSwift
import RxCocoa

class NewCaseViewController: UIViewController,
StoryboardLoadable {

    // MARK: - Factory

    static func initModule(viewModel: NewCaseViewModel) ->
NewCaseViewController {
        let viewController = loadFromStoryboard()
        viewController.viewModel = viewModel
```

```
        return viewController
    }

    // MARK: - IBOutlets

    @IBOutlet private var testResultPicker: UIPickerView!
    @IBOutlet private var sendResultButton: UIButton!
    @IBOutlet private var symptomsSwitches: [UISwitch]!

    // MARK: - Properties

    private var viewModel: NewCaseViewModel! {
          didSet {
              setupObservables()
          }
    }

    private let pickerOptions: Observable<[String]> =
Observable.of([
        "i-dont-know-tests-results".localized,
        "confirmed-case".localized,
        "negative".localized,
        "cured".localized
    ])

    override var preferredStatusBarStyle: UIStatusBarStyle {
        if #available(iOS 13.0, *) {
            return .darkContent
        } else {
            return .default
        }
    }

    private var disposeBag = DisposeBag()

    // MARK: - Life cycle

    override func viewDidLoad() {
```

```
        super.viewDidLoad()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
    }

    // MARK: - Private

    private func setupObservables() {
        setupSwitches()
        setupSendResultButton()
        setupPicker()
        self.viewModel.error.subscribe { error in
            if error {
                // TODO: Review the l10n of this
                let alert = UIAlertController(
                    title: "try-again-later".localized,
                    message:
"we-are-facing-issues-to-send-your-data".localized,
                    preferredStyle: .alert
                )
                alert.addAction(UIAlertAction(
                    title: "ok".localizedUppercase,
                    style: .default,
                    handler: { [weak self] _ in

self?.navigationController?.popViewController(animated: true)
                    }))
                self.present(alert, animated: true)
            }
        }.disposed(by: disposeBag)

        self.viewModel.symptomsSent.subscribe { symptomsSent
in
            if symptomsSent {
                // TODO: Review the l10n of this
```

```
                let alert = UIAlertController(title:
"sent!".localized, message:
"your-self-diagnosis-were-sent-successfully".localized,
preferredStyle: .alert)
                alert.addAction(UIAlertAction(title:
"ok".localizedUppercase, style: .default, handler: { [weak
self] _ in

self?.navigationController?.popViewController(animated: true)
                }))
                self.present(alert, animated: true)
            }
        }.disposed(by: disposeBag)
    }

    private func setupSendResultButton() {
        sendResultButton.layer.borderColor =
UIColor.blue.cgColor
        sendResultButton.layer.borderWidth = 1
        sendResultButton.layer.cornerRadius = 25


sendResultButton.rx.controlEvent(.touchUpInside).subscribe(onN
ext: { [unowned self] in
            self.viewModel.handleSendResults()

            let activityIndicator = UIActivityIndicatorView()
            activityIndicator.hidesWhenStopped = true
            activityIndicator.color = UIColor.blue

activityIndicator.translatesAutoresizingMaskIntoConstraints =
false

self.sendResultButton.addSubview(activityIndicator)

self.sendResultButton.addConstraint(self.sendResultButton.cent
erXAnchor.constraint(equalTo:
activityIndicator.centerXAnchor))
```

```
self.sendResultButton.addConstraint(self.sendResultButton.cent
erYAnchor.constraint(equalTo:
activityIndicator.centerYAnchor))
            activityIndicator.startAnimating()
            self.sendResultButton.isEnabled = false
        }).disposed(by: disposeBag)
    }

    private func setupSwitches() {
        for item in symptomsSwitches {
            item.rx.controlEvent(.valueChanged).map { _ in
                return (item.tag, item.isOn)
            }.subscribe { [unowned self] (tag, value) in
                self.toogleSwitch(tag: tag, value: value)
            }.disposed(by: disposeBag)
        }
    }

    private func setupPicker() {
        pickerOptions.bind(to: testResultPicker.rx.itemTitles)
{ (_, element) in
            return element
        }.disposed(by: disposeBag)

        testResultPicker.rx.itemSelected.subscribe(onNext: {
[unowned self] (row, _) in
            self.viewModel.handleTestStatusChanged(row)
        }).disposed(by: disposeBag)

        testResultPicker.setValue(UIColor.black, forKey:
"textColor")
        testResultPicker.selectRow(viewModel.testStatus.value,
inComponent: 0, animated: false)
    }

    private func toogleSwitch(tag: Int, value: Bool) {
```

```
        self.viewModel.handleSymptomsChanged(tag, value:
value)

        if tag == 9 && value {
            for s in symptomsSwitches {
                if s.tag != 9 {
                    s.setOn(false, animated: true)
                }
            }
        }
        if tag != 9 && value {
            if let noSymptoms =
self.symptomsSwitches.first(where: { $0.tag == 9}) {
               noSymptoms.setOn(false, animated: true)
            }
        }
    }
}

//
//  NewCaseViewModel.swift
//  CheckInUFSC
//
//  Created by Pedro Freddi on 26/06/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import RxSwift
import RxRelay

class NewCaseViewModel {

    // MARK: - Properties

    private let disposeBag = DisposeBag()
    private(set) var symptoms = BehaviorRelay<[Int]>(value:
Array(repeating: false.intValue(), count: 10))
```

```
    private(set) var testStatus = BehaviorRelay<Int>(value: 0)
    private(set) var error = BehaviorSubject<Bool>(value:
false)
    private(set) var symptomsSent =
BehaviorSubject<Bool>(value: false)
    // MARK: - Actions

    func handleSymptomsChanged(_ option: Int, value: Bool) {
        if option == 9 && value {
            var newSymptoms = Array(repeating:
false.intValue(), count: 9)
            newSymptoms.append(true.intValue())
            symptoms.accept(newSymptoms)
        }

        if option != 9 {
            var newSymptoms = symptoms.value
            newSymptoms[option] = value.intValue()
            symptoms.accept(newSymptoms)
        }
    }

    func handleTestStatusChanged(_ newStatus: Int) {
        self.testStatus.accept(newStatus)
    }

    func handleSendResults() {
        print("Results: \(testStatus.value),
\(symptoms.value)")
        let symptoms = self.symptoms.value.map({ String($0)
}).joined()

        AuthManager.RegisterDevice().request { [weak self]
result in
            switch result {
            case .failure(let error):
                print(error.localizedDescription)
                self?.error.onNext(true)
```

```
            case .success(let handshakeResponse):
                guard let self = self else { return }
                print(handshakeResponse)

StorageManager.shared.setToken(handshakeResponse.token)

StorageManager.shared.setIdentifierDevice(handshakeResponse.id
)

                SelfDiagnosisManager.SendDiagnosis(
                    SymptomsParameters(
                        patientId:
StorageManager.shared.getIdentifierDevice() ?? 0,
                        symptoms: symptoms,
                        oldStatus:
StorageManager.shared.getTestState() ?? 0,
                        actualStatus: self.testStatus.value
                    )).request { result in
                        switch result {
                        case .failure(let error):
                            debugPrint(error)
                            self.error.onNext(true)
                        case .success(_):
                            self.error.onNext(false)
                            self.symptomsSent.onNext(true)

StorageManager.shared.setTestState(self.testStatus.value)
                        }
                    }
            }
        }
    }
}

//
//  LocationAlwaysViewController.swift
//  CovidApp - Covid Community Alert
//
```

```
//  Created by Antonio Romano on 23/03/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit

class LocationAlwaysViewController: StatusBarViewController,
StoryboardLoadable {

    static func initModule() -> LocationAlwaysViewController {
        let viewController = loadFromStoryboard()
        return viewController
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
typically from a nib.

    }

    override func viewDidAppear(_ animated: Bool) {
        print("LOCATION VIEW CONTROLLER ALWAYS")
        NotificationCenter.default.addObserver(self, selector:
#selector(changedLocationAuthorization(notification:)), name:
NSNotification.Name(Constants.Notification.locationChangeStatu
s), object: nil)
        self.run()
    }

    @objc func changedLocationAuthorization(notification:
NSNotification) {
        print("change location status notification received
ALWAYS")
        if let status = notification.object as?
LocationManager.AuthorizationStatus {
            print(status)
            if status == .allowedAlways {
```

```
                return self.goNext()
            }
//            return handleChangeAuthorizationStatus(status)
        } else {
            print("WTF LOCATION? ALWAYS")
        }
    }

    func goNext() {
        print("dismissing view location ALWAYS")

AppDelegate.shared.rootRouter.presentNotificationScreen()
    }

    private func handleChangeAuthorizationStatus(_ status:
LocationManager.AuthorizationStatus) {
        print("handleChangeAuthorizationStatus ALWAYS",
status)
        switch status {
        case .allowedAlways:
            return self.goNext()
        case .allowedWhenInUse:
            let alert =
AlertManager.getAlertConfirmation(title:
NSLocalizedString("Location", comment: "location title
alert"), message: NSLocalizedString("We need to access always
the location, please Open Settings -> CovidApp -> Location ->
Always", comment: "location open always"), confirmAction: {_
in
                guard let settingsUrl = URL(string:
UIApplication.openSettingsURLString) else {
                    print("NO SETTINGS URL")
                    return
                }
                if
UIApplication.shared.canOpenURL(settingsUrl) {
                    UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
```

```
                        print("Settings opened: \(success)")
// Prints true
                    })
                }
            })
            self.present(alert, animated: true)
        case .notAvailable:
            let alert: UIAlertController =
AlertManager.getAlert(title: NSLocalizedString("Location",
comment: "location title alert"), message:
NSLocalizedString("The location seems to be unavailable on
your device", comment: "location unavailable"))
            self.present(alert, animated: true)
            break
        case .notDetermined:
            print("GONNA ASK USER ALWAYS")
            LocationManager.shared.requestAlwaysPermission()
            break
        case .denied:
            if LocationManager.shared.isServiceEnabledForApp()
{
                let alert =
AlertManager.getAlertConfirmation(title:
NSLocalizedString("Location", comment: "location title
alert"), message: NSLocalizedString("We need to access the
location, please Open Settings -> CovidApp -> enable location
access", comment: "location open "), confirmAction: {_ in
                    guard let settingsUrl = URL(string:
UIApplication.openSettingsURLString) else {
                        print("NO SETTINGS URL")
                        return
                    }
                    if
UIApplication.shared.canOpenURL(settingsUrl) {
                        UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
                            print("Settings opened:
\(success)") // Prints true
```

```
                        })
                    }
                })
                self.present(alert, animated: true)
            } else {
                let alert =
AlertManager.getAlertConfirmation(title:
NSLocalizedString("Location", comment: "location title
alert"), message: NSLocalizedString("You need to enable the
location, please Open Settings -> Privacy -> Location
services", comment: "location denied"), confirmAction: {_ in

                    guard let settingsUrl = URL(string:
"App-Prefs:root=LOCATION_SERVICES") else {
                        print("NO SETTINGS GENERAL URL")
                        return
                    }
                    if
UIApplication.shared.canOpenURL(settingsUrl) {
                        UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
                            print("Settings general opened:
\(success)") // Prints true
                        })
                    }
                })
                self.present(alert, animated: true)
            }
            // waiting from user action
            break
        }
    }

    @IBAction func enableLocationAction(_ sender: Any) {
        print("asking to enable location ALWAYS")

self.handleChangeAuthorizationStatus(LocationManager.shared.ge
tPermessionStatus())
```

```
    }

    private func run() {
        print("location status ALWAYS",
LocationManager.shared.getPermessionStatus())
        switch LocationManager.shared.getPermessionStatus() {
        case .allowedAlways:
            return self.goNext()
        case .notAvailable:
            break
        case .notDetermined, .denied, .allowedWhenInUse:
            print("not determined waiting for user ALWAYS")
            // waiting from user action
            break
        }
    }

    @IBAction func skipNext(_ sender: Any) {
        self.goNext()
    }

}

//
//  NotificationViewController.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 16/03/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit

class NotificationViewController: StatusBarViewController,
StoryboardLoadable {

    static func initModule() -> NotificationViewController {
        let viewController = loadFromStoryboard()
```

```
        return viewController
    }

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func viewDidAppear(_ animated: Bool) {
        StorageManager.shared.setFirstAccess(false)
        print("NOTIFICATION VIEW CONTROLLER")
    }

    private func run() {
        print("running")
        NotificationManager.shared.getAuthorizationStatus({
        status in
            print("STATUS", status)
            if status == .allowed {
                DispatchQueue.main.sync {
                    self.goNext()
                    NotificationCenter.default.post(name:
NSNotification.Name(Constants.Notification.notificationChangeS
tatus), object: true)
                }
            }
        })
    }

    private func manageStatus(_ status:
NotificationManager.PermissionStatus) {
        switch status {
        case .allowed:
            self.goNext()
            NotificationCenter.default.post(name:
NSNotification.Name(Constants.Notification.notificationChangeS
tatus), object: true)
            break
        case .denied:
```

```
            let alert =
AlertManager.getAlertConfirmation(title:
NSLocalizedString("Notification", comment: "notification title
alert"), message: NSLocalizedString("If you’ve been close to
an infected person in the past two weeks we will notify you.
That’s it, just one notification.", comment: "notification
open"), confirmAction: {_ in
                guard let settingsUrl = URL(string:
UIApplication.openSettingsURLString) else {
                    print("NO SETTINGS URL")
                    return
                }
                if
UIApplication.shared.canOpenURL(settingsUrl) {
                    UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
                        print("Settings opened: \(success)")
// Prints true
                    })
                }
            })

            self.present(alert, animated: true)
            break
        case .notDetermined:
            NotificationManager.shared.requestPermission({
                granted in
                if granted {
                    DispatchQueue.main.sync {
                        self.goNext()
                        NotificationCenter.default.post(name:
NSNotification.Name(Constants.Notification.notificationChangeS
tatus), object: true)
                    }
                }
            })
            break
        }
```

```
    }

    private func handleStatus() {
        print("handling status")

        if let status = NotificationManager.shared.getStatus()
{
            print("cached status", status)
            return manageStatus(status)
        }

        NotificationManager.shared.getAuthorizationStatus({
            status in
            print("requested status", status)
            DispatchQueue.main.sync {
                return self.manageStatus(status)
            }

        })
    }

    private func goNext() {
        print("going next notification")
        self.dismiss(animated: true, completion: nil)

        if let did = StorageManager.shared.getUFSCToken() {
            if Utils.isActive() {

AppDelegate.shared.rootRouter.presentHomeScreen()
            } else {
                dismiss(animated: true)
            }
        } else {

AppDelegate.shared.rootRouter.presentAuthentication()
        }
    }
```

```
    @IBAction func giveNotification(_ sender: Any) {
        print("give notification")
        self.handleStatus()
    }

    @IBAction func skipNext(_ sender: Any) {
        self.goNext()
    }

}

//
//  BluetoothOffViewController.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 27/02/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit

class BluetoothOffViewController: StatusBarViewController,
StoryboardLoadable {

    static func initModule() -> BluetoothOffViewController {
        let viewController = loadFromStoryboard()
        return viewController
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
typically from a nib.

    }

    override func viewDidAppear(_ animated: Bool) {
        print("BLUETOOTH OFF VIEW CONTROLLER")
```

```
        self.run()
    }

    func run() {
        NotificationCenter.default.addObserver(self, selector:
#selector(changedBluetoothStatus(notification:)), name:
NSNotification.Name(Constants.Notification.bluetoothChangeStat
us), object: nil)
        print("RECEIVED STATUS",
BluetoothManager.shared.getPermissionStatus(),
BluetoothManager.shared.getBluetoothStatus())
        switch BluetoothManager.shared.getPermissionStatus() {
        case .allowed:
//            #if targetEnvironment(simulator)
//            // your code
//            print("is simulator")
//            self.goNext()
//            #endif
            if BluetoothManager.shared.getBluetoothStatus() ==
.on {
                self.goNext()
            }
            if BluetoothManager.shared.getBluetoothStatus() ==
.off {
            }
            if BluetoothManager.shared.getBluetoothStatus() ==
.resetting {
            }
            break
        case .denied, .notDetermined:
            // we will wait for user to click on the button
            break
        case .notAvailable:
            self.goNext()
            break
        }
    }
```

```
    @objc func changedBluetoothStatus(notification:
NSNotification) {
        print("change status notification received")
        if let status = notification.object as?
BluetoothManager.Status {
            return handleBluetoothStatus(status)
        } else {
            print("WTF?")
        }
    }

    private func handleBluetoothStatus(_ status:
BluetoothManager.Status) {
        print("handling status: ", status)
        switch status {
        case .on:
            self.goNext()
        case .off:
            self.bluetoothOff()
            break
        case .resetting:
            let alert: UIAlertController =
AlertManager.getAlert(title: NSLocalizedString("Bluetooth",
comment: "bluetooth title alert"), message:
NSLocalizedString("The bluetooth seems to be resetting, please
try later", comment: "bluetooth unavailable"))
            self.present(alert, animated: true)
            break
        case .notAvailable:
            self.goNext()
            break
        case .unauthorized:
            self.bluetoothDeniedOrUnauthorized()
        }
    }

    @IBAction func openBluetoothAction(_ sender: Any) {
```

```
        switch BluetoothManager.shared.getPermissionStatus() {
        case .allowed:

self.handleBluetoothStatus(BluetoothManager.shared.getBluetoot
hStatus())
            break
        case .denied:
            self.bluetoothDeniedOrUnauthorized()
            break
        case .notDetermined:
            print("bluetooth off view asking permission")
            // we will wait for user to click on the button
            BluetoothManager.shared.askUserPermission()
            break
        case .notAvailable:
            self.goNext()
            break
        }
    }

    func goNext() {
        print("dismissing view bluetooth")

        if StorageManager.shared.isFirstAccess() {

AppDelegate.shared.rootRouter.presentLocationScreen()
        } else {
            if Utils.isActive() {

AppDelegate.shared.rootRouter.presentHomeScreen()
            } else {

AppDelegate.shared.rootRouter.presentInactiveHomeScreen()
            }
        }
    }

    func bluetoothOff() {
```

```
        let alert = AlertManager.getAlertConfirmation(title:
NSLocalizedString("Bluetooth", comment: "bluetooth title
alert"), message: NSLocalizedString("You need to enable the
bluetooth, please Open Settings -> Bluetooth -> enable
bluetooth", comment: "bluetooth off"), confirmAction: {_ in

            guard let settingsUrl = URL(string:
"App-Prefs:root=General&path=Bluetooth") else {
                print("NO SETTINGS GENERAL URL")
                return
            }
            if UIApplication.shared.canOpenURL(settingsUrl) {
                UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
                    print("Settings general opened:
\(success)") // Prints true
                })
            }
        })
        self.present(alert, animated: true)
    }

    func bluetoothDeniedOrUnauthorized() {
        let alert = AlertManager.getAlertConfirmation(title:
NSLocalizedString("Bluetooth", comment: "bluetooth title
alert"), message: NSLocalizedString("We need to access the
bluetooth, please Open Settings -> CovidApp -> enable
bluetooth access", comment: "bluetooth open app settings"),
confirmAction: {_ in
            guard let settingsUrl = URL(string:
UIApplication.openSettingsURLString) else {
                print("NO SETTINGS URL")
                return
            }
            if UIApplication.shared.canOpenURL(settingsUrl) {
                UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
```

```
                    print("Settings opened: \(success)") //
Prints true
                })
            }
        })
        self.present(alert, animated: true)
    }

    func bluetoothNotAvailable() {
        let alert: UIAlertController =
AlertManager.getAlert(title: NSLocalizedString("Bluetooth",
comment: "bluetooth title alert"), message:
NSLocalizedString("The bluetooth seems to be unavailable on
your device", comment: "bluetooth unavailable"))
        self.present(alert, animated: true)
    }

    // TODO: Review this IBAction
    @IBAction func howCanIHelpMoreAction(_ sender: Any) {
        print("Not available")
//        let storyBoard : UIStoryboard = UIStoryboard(name:
"Main", bundle:nil)
//        let nextViewController =
storyBoard.instantiateViewController(withIdentifier:
"HelpMoreViewController") as! HelpMoreViewController
//        self.present(nextViewController, animated:true,
completion:nil)
    }

    @IBAction func skipNext(_ sender: Any) {
        self.goNext()
    }

}

//
//  RootRouter.swift
//  CheckInUFSC
```

```
//
//  Created by Pedro Freddi on 26/06/22.
//  Copyright © 2022 Coronavirus-Herd-Immunity. All rights
reserved.
//

import UIKit

final class RootRouter: PresentView {
    // MARK: Static

    static func initModule() -> RootRouter {
        return RootRouter()
    }

    // MARK: Actions

    func presentInitialScreen() {
        if StorageManager.shared.getIdentifierDevice() == nil
{
            presentWelcomeScreen()
        } else {
            if Utils.isActive() {
                presentHomeScreen()
            } else {
                presentInactiveHomeScreen()
            }
        }
    }

    // MARK: Private

    func presentWelcomeScreen() {
        let viewController =
WelcomeViewController.initModule()
        presentView(viewController)
    }
```

```
    func presentHomeScreen() {
        let viewController = MainViewController.initModule()
        let navController =
NavigationController(rootViewController: viewController)
        navController.navigationBar.barStyle = .black
        presentView(navController)
    }

    func presentInactiveHomeScreen() {
        let viewController =
InactiveViewController.initModule()
        presentView(viewController)
    }

    func presentBluetoothOffScreen() {
        let viewController =
BluetoothOffViewController.initModule()
        presentView(viewController)
    }

    func presentLocationScreen() {
        let viewController =
LocationViewController.initModule()
        presentView(viewController)
    }

    func presentNotificationScreen() {
        let viewController =
NotificationViewController.initModule()
        presentView(viewController)
    }

    func getNewSymptomScreen() -> UIViewController {
        let viewModel = NewCaseViewModel()
        let viewController =
NewCaseViewController.initModule(viewModel: viewModel)
        return viewController
    }
```

```
    func presentAuthentication() {
        let viewModel = AuthenticationViewModel()
        let viewController =
AuthenticationViewController.initModule(viewModel: viewModel)
        presentView(viewController)
    }

    func presentLocationAlways() {
        let viewController =
LocationAlwaysViewController.initModule()
        presentView(viewController)
    }
}

class NavigationController: UINavigationController {
    // Return the visible child view controller which
determines the status bar style.
    override var childForStatusBarStyle: UIViewController? {
return visibleViewController }
}

//
//  VirologistInformationViewController.swift
//  CovidApp - Covid Community Alert
//
//  Created by Antonio Romano on 06/04/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit

class VirologistInformationViewController:
StatusBarViewController, StoryboardLoadable {

    static func initModule() ->
VirologistInformationViewController {
        let viewController = loadFromStoryboard()
```

```
        return viewController
    }

    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var shortDescriptionLabel: UILabel!
    @IBOutlet weak var spinner: UIActivityIndicatorView!
    @IBOutlet weak var contentLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func viewDidAppear(_ animated: Bool) {

        if let data =
PushNotificationData.readNotificationDate() {
//            if let title = data.title{
//                self.titleLabel.text = title
//            }

            if let message = data.message {
                self.shortDescriptionLabel.text = message
            }

            // TODO: handle language
            if var link = data.link {
                var otherLink: String?
                if link.contains("{language}") {
                    if Locale.preferredLanguages.count > 0 {
                        link = link.replacingOccurrences(of:
"", with: Locale.preferredLanguages[0])
                    }
                    if let localizedLanguage = data.language {
                        link = link.replacingOccurrences(of:
"", with: localizedLanguage)
                    }
                }
```

```
                self.loadNotificatonData(data: data, link:
link, otherLink: otherLink)
            } else {
                print("no link found")
                self.rollbackStatusNoLink()
            }
        } else {
            print("no notification found")
            self.rollbackStatusNoLink()
        }
    }

    private func loadNotificatonData(data:
PushNotificationData, link: String, otherLink: String?) {
        print("gonna load from", link)
        AlamofireManager.shared.downloadDataNotification(link,
callback: {
            response, success in

            print("LOADED VIROLOGISTS INFORMATIONS")

            if success {

                self.spinner.isHidden = true

                if let res = response as? [String: Any] {
                    if let filters = res["filters"] as?
[[String: Any]] {
                        if let filterId = data.filterId {
                            for filter in filters {
                                if let fid =
filter["filter_id"] as? Int {
                                    if fid == filterId {
                                        if let language =
filter["language"] as? String {
                                            if let content =
filter["content"] as? [String: Any] {
```

```
                                                    // TODO:
search local language
                                                    if let
localizedContent = content[language] as? [String: Any] {

                                                        if let
short = localizedContent["shortDescription"] as? String {

self.shortDescriptionLabel.text = short
                                                        }

                                                        if let
long = localizedContent["description"] as? String {

self.contentLabel.text = long
                                                        }

                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            } else {
                                if let shortDescription =
res["shortDescription"] as? String {
                                    self.shortDescriptionLabel.text =
shortDescription
                                }

                                if let description =
res["description"] as? String {
                                    self.contentLabel.text =
description
                                }
                            }
                        }
```

```
            } else {
                print("NO DATA FOUND IN LINK", otherLink)
                if let otherL = otherLink {
                    self.loadNotificatonData(data: data, link:
otherL, otherLink: nil)
                } else {
                    self.rollbackStatusNoLink()
                }
            }
        })
    }

    private func rollbackStatusNoLink() {
        print("rollback status")
        self.spinner.isHidden = true
        if let data =
PushNotificationData.readNotificationDate() {
            if data.status != nil && data.status! == 1 {
                // infected case
                self.shortDescriptionLabel.text =
NSLocalizedString("Short description infected", comment:
"Short description infected virologist panel")
                self.contentLabel.text =
NSLocalizedString("Content description infected", comment:
"Content description infected virologist panel")
            } else {
                self.shortDescriptionLabel.text =
NSLocalizedString("Short description normal", comment: "Short
description normal virologist panel")
                self.contentLabel.text =
NSLocalizedString("Content description normal", comment:
"Content description normal virologist panel")
            }
        } else {
            self.shortDescriptionLabel.text =
NSLocalizedString("Short description normal", comment: "Short
description normal virologist panel")
```

```
            self.contentLabel.text =
NSLocalizedString("Content description normal", comment:
"Content description normal virologist panel")
        }
    }

    @IBAction func backButton(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
    }

}

//
//  StatusBarViewController.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 14/03/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit
import Social
import MessageUI

class StatusBarViewController: UIViewController,
MFMailComposeViewControllerDelegate,
MFMessageComposeViewControllerDelegate {

    override var preferredStatusBarStyle: UIStatusBarStyle {
        if #available(iOS 13.0, *) {
            return .darkContent
        } else {
            // Fallback on earlier versions
            return .default
//            return .lightContent
        }
    }
```

```
    func messageComposeViewController(_ controller:
MFMessageComposeViewController, didFinishWith result:
MessageComposeResult) {
        print("Result SMS", result.rawValue)
        controller.dismiss(animated: true, completion: nil)
    }

    func mailComposeController(_ controller:
MFMailComposeViewController, didFinishWith result:
MFMailComposeResult, error: Error?) {
        print("Result Email", result.rawValue)
        controller.dismiss(animated: true, completion: nil)
    }

}

//
//  LocationViewController.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 02/03/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit

// https://forums.developer.apple.com/thread/117256

class LocationViewController: UIViewController,
StoryboardLoadable {

    static func initModule() -> LocationViewController {
        let viewController = loadFromStoryboard()
        return viewController
    }

    override func viewDidLoad() {
        super.viewDidLoad()
```

```
        // Do any additional setup after loading the view,
typically from a nib.

    }

    override func viewDidAppear(_ animated: Bool) {
        print("LOCATION VIEW CONTROLLER")
        NotificationCenter.default.addObserver(self, selector:
#selector(changedLocationAuthorization(notification:)), name:
NSNotification.Name(Constants.Notification.locationChangeStatu
s), object: nil)
        self.run()
    }

    @objc func changedLocationAuthorization(notification:
NSNotification) {
        print("change location status notification received")
        if let status = notification.object as?
LocationManager.AuthorizationStatus {
            print(status)
            if status == .allowedAlways {
                return self.goNext()
            }
            if status == .allowedWhenInUse {

                print("SWITCHING NEXT")
                self.switchAlwaysPermission()
                return
            }
//            return handleChangeAuthorizationStatus(status)
        } else {
            print("WTF LOCATION?")
        }
    }

    func goNext() {

AppDelegate.shared.rootRouter.presentNotificationScreen()
```

```
    }

    private func handleChangeAuthorizationStatus(_ status:
LocationManager.AuthorizationStatus) {
        print("handleChangeAuthorizationStatus", status)
        switch status {
        case .allowedAlways:
            return self.goNext()
        case .allowedWhenInUse:
            print("SWITCHING NEXT")
            self.switchAlwaysPermission()
            break
        case .notAvailable:
            let alert: UIAlertController =
AlertManager.getAlert(title: NSLocalizedString("Location",
comment: "location title alert"), message:
NSLocalizedString("The location seems to be unavailable on
your device", comment: "location unavailable"))
            self.present(alert, animated: true)
            break
        case .notDetermined:
            print("GONNA ASK USER")
            self.switchAlwaysPermission()
            LocationManager.shared.requestAlwaysPermission()
            break
        case .denied:
            if LocationManager.shared.isServiceEnabledForApp()
{
                let alert =
AlertManager.getAlertConfirmation(title:
NSLocalizedString("Location", comment: "location title
alert"), message: NSLocalizedString("We need to access the
location, please Open Settings -> CovidApp -> enable location
access", comment: "location open "), confirmAction: {_ in
                    guard let settingsUrl = URL(string:
UIApplication.openSettingsURLString) else {
                        print("NO SETTINGS URL")
                        return
```

```
                    }
                    if
UIApplication.shared.canOpenURL(settingsUrl) {
                        UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
                            print("Settings opened:
\(success)") // Prints true
                        })
                    }
                })
                self.present(alert, animated: true)
            } else {
                let alert =
AlertManager.getAlertConfirmation(title:
NSLocalizedString("Location", comment: "location title
alert"), message: NSLocalizedString("You need to enable the
location, please Open Settings -> Privacy -> Location
services", comment: "location denied"), confirmAction: {_ in

                    guard let settingsUrl = URL(string:
"App-Prefs:root=LOCATION_SERVICES") else {
                        print("NO SETTINGS GENERAL URL")
                        return
                    }
                    if
UIApplication.shared.canOpenURL(settingsUrl) {
                        UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
                            print("Settings general opened:
\(success)") // Prints true
                        })
                    }
                })
                self.present(alert, animated: true)
            }
            // waiting from user action
            break
        }
```

```
    }

    private func switchAlwaysPermission() {
        NotificationCenter.default.removeObserver(self)
        AppDelegate.shared.rootRouter.presentLocationAlways()
    }

    @IBAction func enableLocationAction(_ sender: Any) {
        print("asking to enable location")

self.handleChangeAuthorizationStatus(LocationManager.shared.ge
tPermessionStatus())
    }

    private func run() {
        print("location status",
LocationManager.shared.getPermessionStatus())
        switch LocationManager.shared.getPermessionStatus() {
        case .allowedAlways:
            return self.goNext()
        case .notAvailable:
            break
        case .allowedWhenInUse:
            NotificationCenter.default.removeObserver(self)
            self.switchAlwaysPermission()
            break
        case .notDetermined, .denied:
            print("not determined waiting for user LOCATION")
            // waiting from user action
            break
        }
    }

    @IBAction func skipNext(_ sender: Any) {
        self.goNext()
    }

}
```

```
//
//  HowItWorks.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 14/03/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit

class HowItWorksViewController: StatusBarViewController,
StoryboardLoadable {

    static func initModule() -> HowItWorksViewController {
        let viewController = loadFromStoryboard()
        return viewController
    }

    @IBOutlet weak var scrollView: UIScrollView!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
typically from a nib.
        print("HOW IT WORKS CONTROLLER")
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
    }

    @IBAction func backButton(_ sender: Any) {
        self.dismiss(animated: true, completion: nil)
    }

}
```

```
//
//  MainViewController.swift
//  CheckInUFSC
//
//  Created by Antonio Romano on 25/02/2020.
//  Copyright © 2022 CheckInUFSC. All rights reserved.
//

import UIKit
import CoreBluetooth

class MainViewController: StatusBarViewController,
StoryboardLoadable {

    static func initModule() -> MainViewController {
        let viewController = loadFromStoryboard()
        return viewController
    }

    @IBOutlet private var scrollView: UIScrollView!
    @IBOutlet private var debugButton: UIButton!
    @IBOutlet private var titleLabel: UILabel!
    @IBOutlet private var messageLabel: UILabel!
    @IBOutlet private var messageButton: UILabel!
    @IBOutlet private var greetingsLabel: UILabel!
    private var counterHidden: Int = 0

    override var preferredStatusBarStyle: UIStatusBarStyle {
        if #available(iOS 13.0, *) {
            return .darkContent
        } else {
            return .default
        }
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        self.updateStatus()
```

```
        setupGreetingsLabel()
    }

    override func viewWillDisappear(_ animated: Bool) {
        self.updateStatus()
    }

    override func viewWillAppear(_ animated: Bool) {
        self.updateStatus()
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        print("LOADING MAIN")
        self.run()

        self.updateStatus()

        if Bundle.main.bundleIdentifier!.contains("debug") {
            self.debugButton.isHidden = false
            self.debugButton.isEnabled = true
        }
    }

    private func run() {
        self.updateStatus()
        NotificationCenter.default.addObserver(self, selector:
#selector(statusChanged), name:
NSNotification.Name(Constants.Notification.bluetoothChangeStat
us), object: nil)
        NotificationCenter.default.addObserver(self, selector:
#selector(statusChanged), name:
NSNotification.Name(Constants.Notification.locationChangeStatu
s), object: nil)
        NotificationCenter.default.addObserver(self, selector:
#selector(statusChanged), name:
NSNotification.Name(Constants.Notification.notificationChangeS
tatus), object: nil)
```

```
        NotificationCenter.default.addObserver(self, selector:
#selector(updatePatientStatus), name:
NSNotification.Name(Constants.Notification.patientChangeStatus
), object: nil)
    }

    @objc private func statusChanged() {
        if !Utils.isActive() {
            let routeRouter = AppDelegate.shared.rootRouter
            DispatchQueue.main.async {
                routeRouter.presentInactiveHomeScreen()
            }
        }
    }

    @objc func updatePatientStatus() {
        DispatchQueue.main.async {
            self.updateStatus()
        }
    }

    private func updateStatus() {
        if StorageManager.shared.getPushId() == nil {
            print("no push if, gonna register")
            if NotificationManager.shared.getStatus() ==
.allowed {
                NotificationManager.shared.requestPermission({
                    granted in
                    print("permission grant is", granted)
                })
            }
        }

        if let data =
PushNotificationData.readNotificationDate() {

            if let status = data.status {
                let warningLevel = data.warningLevel ?? 0
```

```
                var text: String = NSLocalizedString("No risk
detected", comment: "No risk detected")
                var msg: String = NSLocalizedString("Normal
Status message", comment: "Normal Status message")
                var color = Constants.UI.colorStandard

                if warningLevel <
Constants.UI.warningLevelColors.count {
                    color =
Constants.UI.warningLevelColors[warningLevel]
                }

                var titleColor = UIColor(red: 51 / 255, green:
51 / 255, blue: 51 / 255, alpha: 1)

                switch status {
                case 1:
                    print("infected status")
                    text = NSLocalizedString("Infected",
comment: "Infected")
                    msg = NSLocalizedString("Infected Status
message", comment: "Infected Status message")
                    break
                case 2:
                    print("suspect status")
                    text = NSLocalizedString("Suspect",
comment: "Suspect status")
                    break
                case 3:
                    print("healed status")
                    text = NSLocalizedString("Healed",
comment: "Healed status")
                    msg = NSLocalizedString("Healed Status
message", comment: "Healed Status message")
                    titleColor = .white
                    break
                case 4:
```

```
                    print("low risk")
                    text = NSLocalizedString("Low risk",
comment: "Low risk")
                    msg = NSLocalizedString("Quarantine Status
message", comment: "Quarantine Status message")
                    titleColor = .white
                    break
                case 5:
                    print("mid risk")
                    text = NSLocalizedString("Mid risk",
comment: "Mid risk")
                    msg = NSLocalizedString("Quarantine Status
message", comment: "Quarantine Status message")
                    titleColor = .white
                    break
                case 6:
                    print("high risk")
                    text = NSLocalizedString("High risk",
comment: "High risk")
                    msg = NSLocalizedString("Quarantine Status
message", comment: "Quarantine Status message")
                    titleColor = .white
                    break
                default:
                    text = NSLocalizedString("No risk
detected", comment: "No risk detected")
                    msg = NSLocalizedString("Normal Status
message", comment: "Normal Status message")
                    color = Constants.UI.colorStandard
                }
                self.titleLabel.text = text
                self.titleLabel.textColor = titleColor
                if let message = data.message {
                    self.messageLabel.text = message
                } else {
                    self.messageLabel.text = msg
                }
            }
```

```
        }
    }

    private func setupGreetingsLabel() {
        guard let user = StorageManager.shared.getUser() else
{
            greetingsLabel.isHidden = true
            return
        }

        greetingsLabel.text = "Olá, \(user.attributes.name)!"
    }

    @IBAction func howItWorks(_ sender: Any) {
        let nextViewController =
HowItWorksViewController.initModule()
        self.present(nextViewController, animated: true,
completion: nil)
    }

    @IBAction func shareFacebook(_ sender: Any) {
        print("facebook")
        ShareManager.shareFacebook(self)
    }

    @IBAction func shareTwitter(_ sender: Any) {
        print("twitter")
        ShareManager.shareTwitter(self)
    }

    @IBAction func shareWhatsapp(_ sender: Any) {
        print("whatsapp")
        ShareManager.shareWhatsapp(self)
    }

    @IBAction func shareSMS(_ sender: Any) {
        print("SMS")
        ShareManager.shareSMS(self)
```

```
    }

    @IBAction func shareEmail(_ sender: Any) {
        print("email")
        ShareManager.shareEmail(self)
    }

    @IBAction func shareCopyLink(_ sender: Any) {
        print("copylink")
        ShareManager.copyLink(self)
    }

    @IBAction func debugInteractions(_ sender: Any) {
        let storyBoard: UIStoryboard = UIStoryboard(name:
"Main", bundle: nil)
        let nextViewController =
storyBoard.instantiateViewController(withIdentifier:
"BluetoothTableViewController") as!
BluetoothTableViewController
        self.present(nextViewController, animated: true,
completion: nil)
    }

    @IBAction func messageActionMoreInfo(_ sender: Any) {
        let nextViewController =
VirologistInformationViewController.initModule()
        self.present(nextViewController, animated: true,
completion: nil)
    }

    @IBAction func showID(_ sender: Any) {
        if let did =
StorageManager.shared.getIdentifierDevice() {

            var sum = 0
            for char in did.description {
                sum += Int(char.description) ?? 0
            }
```

```
            let didChecksum = sum.description.last ?? "0"

            let alert: UIAlertController =
AlertManager.getAlert(title: "ID", message: did.description +
didChecksum.description)
            self.present(alert, animated: true)
        }
    }

    @IBAction func onNewSymptomTap(_ sender: Any) {

self.navigationController?.pushViewController(AppDelegate.shar
ed.rootRouter.getNewSymptomScreen(), animated: true)
    }
}
```

ANEXO A – CASOS DE TESTE - COVIDAPP

Casos de Teste - CovidApp

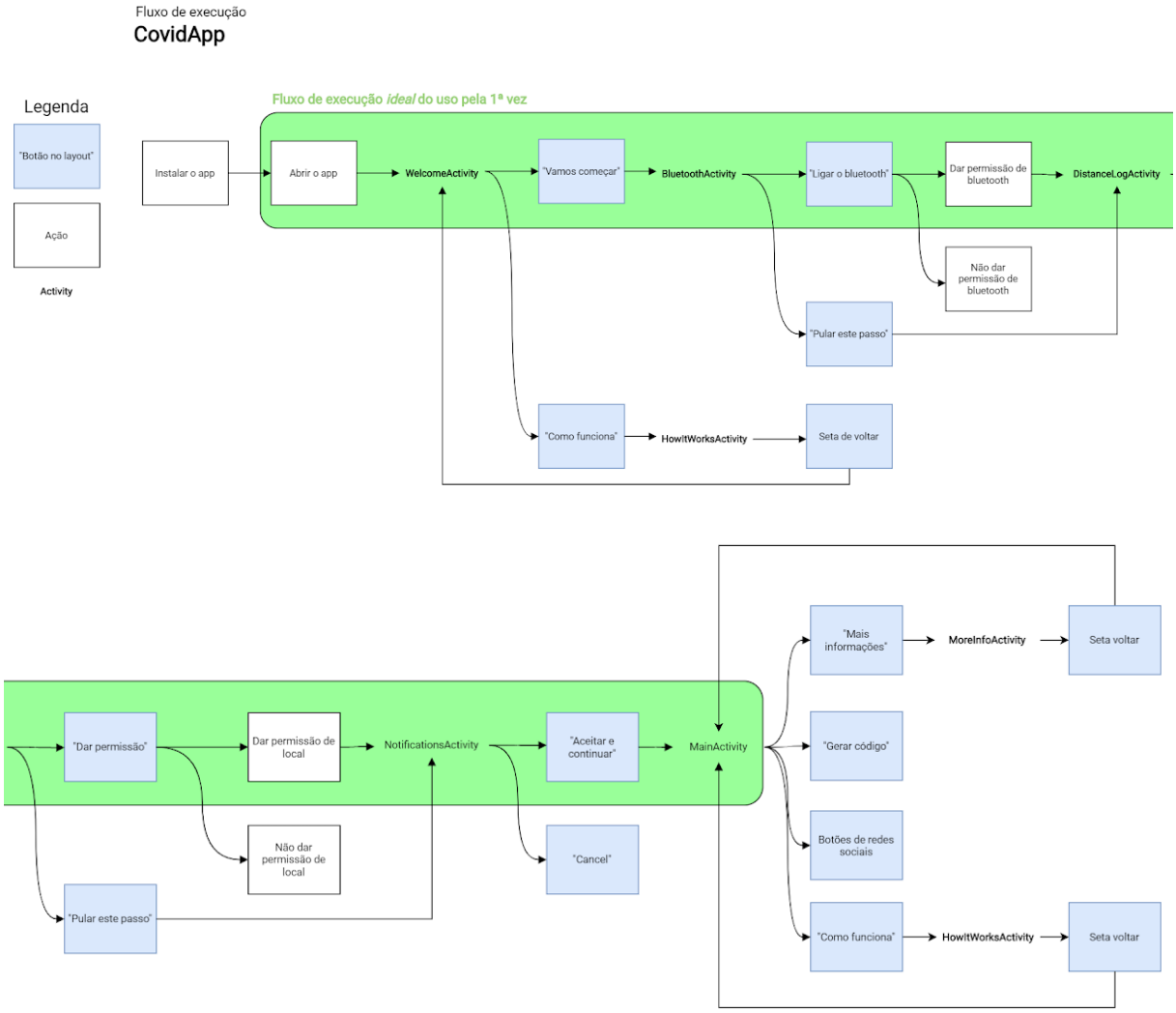
Larissa G. Rosa e Lucas Barzan

Tabela de conteúdos

👩‍🏫 Introdução

Fluxo de execução

Primeiramente, identificamos o fluxo de execução do CovidApp, isto é, que “caminhos” podem ser tomados dentro da aplicação, para que pudéssemos gerar casos de teste com a maior cobertura possível. Segue abaixo o diagrama (dividido em duas partes para caber). Link para o diagrama (ao acessar pelo link, recomenda-se dar zoom na imagem ou abri-la por meio do aplicativo “diagrams.net”).



Como os casos de teste estão organizados

Os casos de teste são numerados e cada um possui um título breve que resume o cenário abordado no mesmo. Todo caso de teste é composto de duas partes:

- **Pré requisitos:** as condições iniciais necessárias para reproduzir o teste adequadamente (ex.: ter instalado o CovidApp);

- **Descrição:** O objetivo do teste. Nesta seção também são discriminados ordenadamente os passos necessários e os resultados esperados a cada passo para que se execute o teste do início ao fim.

Tanto os pré-requisitos como os passos necessários podem variar de acordo com a plataforma (Android ou iOS) e, neste caso, são descritos separadamente dentro do mesmo caso de teste.

Conhecimento prévio

Para referenciar as telas da aplicação nos resultados esperados, utilizamos seu "nome" a partir do código para Android (como *MainActivity*, *WelcomeActivity*, etc.), porém elas se diferenciam em alguns momentos, entre os dois sistemas operacionais.

📋 Caso de teste 1: Usuário realiza onboarding

Pré-requisitos

- Ter o CovidApp instalado para Android ou iOS, sem tê-lo aberto desde a instalação.

Descrição

Este teste tem como objetivo verificar se o uso do app pela primeira vez é bem-sucedido.

Android

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Como funciona".	É iniciada e exibida a HowItWorksActivity.
3	Rolar a tela até o final e clicar no botão com a seta para voltar.	Volta-se à WelcomeActivity.
4	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity.
5	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	É iniciada e exibida a DistanceLogActivity.
6	Clicar no botão "Dar permissão" e permitir o uso da localização no diálogo do sistema que aparece.	É iniciada e exibida a NotificationsActivity.
7	Clicar no botão "Aceitar e continuar!".	É iniciada e exibida a MainActivity, com o QR Code do paciente.

iOS

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Como funciona".	É iniciada e exibida a HowItWorksActivity.
3	Rolar a tela até o final e clicar no botão com a seta para voltar.	Volta-se à WelcomeActivity.
4	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity. (Antes disso pode ser exibido um Captcha a ser completado).
5	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	É iniciada e exibida a DistanceLogActivity.
6	Clicar no botão "Conceder permissão" e permitir o uso da localização *durante o uso do app* no diálogo do sistema que aparece. Clicar novamente em "Conceder permissão".	É exibida novamente a DistanceLogActivity porém agora com texto que explica a necessidade de "sempre permitir" o uso da localização.

7	Clicar novamente em "Conceder permissão". Clicar em OK no diálogo que aparece explicando como ativar a permissão. Clicar em "Localização" e selecionar "Sempre". Voltar para o CovidApp e clicar em "Conceder permissão".	É iniciada e exibida a NotificationsActivity.
8	Clicar no botão "Permitir notificações" e permitir o uso de notificações no diálogo do sistema que aparece.	É iniciada e exibida a MainActivity, com o QR Code do paciente.

📋 Caso de teste 2: Usuário interrompe onboarding

Pré-requisitos

- Ter o CovidApp instalado para Android ou iOS, sem tê-lo aberto desde a instalação.

Descrição

Este teste tem como objetivo verificar se, depois de interromper o processo de onboarding, ele consegue retomá-lo ou conceder as permissões na MainActivity.

Android

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity.
3	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	É iniciada e exibida a DistanceLogActivity.
4	Encerrar a aplicação e abrí-la novamente.	É iniciada e exibida a WelcomeActivity.
5	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity.
6	Clicar no botão "Ligar o bluetooth".	É iniciada e exibida a DistanceLogActivity.
7	Clicar no botão "Dar permissão" e permitir o uso da localização no diálogo do sistema que aparece.	É iniciada e exibida a NotificationsActivity.
8	Encerrar a aplicação e abrí-la novamente.	É iniciada e exibida a WelcomeActivity.
9	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity.
10	Clicar no botão "Ligar o bluetooth".	É iniciada e exibida a DistanceLogActivity.
11	Clicar no botão "Dar permissão".	É iniciada e exibida a NotificationsActivity.
12	Clicar no botão "Aceitar e continuar!".	É iniciada e exibida a MainActivity, com o QR Code do paciente.

iOS

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity. (Antes disso pode ser exibido um Captcha a ser completado).
3	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	É iniciada e exibida a DistanceLogActivity.
4	Encerrar a aplicação e abrí-la novamente.	É iniciada e exibida a DistanceLogActivity.

5	Clicar no botão "Conceder permissão" e permitir o uso da localização *durante o uso do app* no diálogo do sistema que aparece. Clicar novamente em "Conceder permissão".	É exibida novamente a DistanceLogActivity porém agora com texto que explica a necessidade de "sempre permitir" o uso da localização.
6	Clicar novamente em "Conceder permissão". Clicar em OK no diálogo que aparece explicando como ativar a permissão. Clicar em "Localização" e selecionar "Sempre". Voltar para o CovidApp e clicar em "Conceder permissão".	É iniciada e exibida a NotificationsActivity.
7	Encerrar a aplicação e abrí-la novamente.	É iniciada e exibida a MainActivity, porém com um texto para cada permissão indicando seu status.
8	Clicar em "Conceder permissão de notificações", clicar em "Permitir notificações" e permitir no diálogo do sistema que aparece.	Volta-se à MainActivity, agora exibindo o QR Code do paciente.

📋 Caso de teste 3: Usuário pula as etapas do onboarding

Pré-requisitos

- Possuir o CovidApp instalado para Android ou iOS, sem tê-lo aberto desde a instalação.

Descrição

Este teste tem como objetivo verificar se o app requisita na MainActivity as permissões necessárias caso o usuário não as tenha concedido durante o onboarding ao pular as etapas.

Android

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity.
3	Clicar no botão "Pular este passo".	É iniciada e exibida a DistanceLogActivity.
4	Clicar no botão "Pular este passo".	É iniciada e exibida a NotificationsActivity.
5	Clicar no botão "Aceitar e continuar!".	É iniciada e exibida a MainActivity. Aparece diálogo do app solicitando que o usuário habilite o bluetooth.
6	Clicar no botão "OK" do diálogo.	É redirecionado à BluetoothActivity.
7	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	É redirecionado de volta à MainActivity. Aparece diálogo do app solicitando que o usuário habilite a localização.
8	Clicar no botão "OK" do diálogo.	É redirecionado à DistanceLogActivity.
9	Clicar no botão "Dar permissão" e permitir o uso da localização no diálogo do sistema que aparece.	É redirecionado de volta à MainActivity.

iOS

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity. (Antes disso pode ser exibido um Captcha a ser completado).
3	Clicar no botão "Pular este passo".	É iniciada e exibida a DistanceLogActivity.
4	Clicar no botão "Pular este passo".	É iniciada e exibida a NotificationsActivity.

5	Clicar no botão "Pular este passo".	É iniciada e exibida a MainActivity, porém com um texto para cada permissão indicando seu status. No caso todos estarão vermelhos.
6	Clicar no botão "Conceder permissão bluetooth".	É iniciada e exibida a BluetoothActivity.
7	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	Volta-se à MainActivity.
8	Clicar no botão "Conceder permissão de notificações".	É iniciada e exibida a NotificationsActivity.
9	Clicar no botão "Permitir notificações" e permitir o uso de notificações no diálogo do sistema que aparece.	Volta-se à MainActivity.
10	Clicar no botão "Conceder monitoramento de distância".	É iniciada e exibida a DistanceLogActivity.
11	Clicar no botão "Conceder permissão" e clicar no botão OK no diálogo que aparece explicando como ativar. Clicar em "Localização" e selecionar "Sempre". Voltar para o CovidApp e clicar novamente em "Conceder permissão".	Volta-se à MainActivity, agora exibindo o QR Code do paciente.

📋 Caso de teste 4: Usuário obtém mais informações sobre status

Pré-requisitos

- Possuir o CovidApp instalado;
- Ter finalizado com sucesso o onboarding;
- Fechar qualquer instância já aberta do app.

Descrição

Este teste tem como objetivo verificar se o usuário consegue obter mais informações sobre seu status.

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a MainActivity.
2	Clicar no botão "Mais informações".	É iniciada e exibida a MoreInfoActivity com detalhes sobre o status atual do paciente.

📋 Caso de teste 5: Usuário gera seu código de paciente

Pré-requisitos

- Possuir o CovidApp instalado para Android ou iOS;
- Ter finalizado com sucesso o onboarding;
- Fechar qualquer instância já aberta do app.

Descrição

Este teste tem como objetivo verificar se o usuário consegue gerar seu código ID (para informá-lo a um profissional da saúde, por exemplo).

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a MainActivity.
2	Clicar no botão "Gerar código".	É aberto um modal com o código ID do paciente no formato DD-DD- (onde D é um dígito).

📋 Caso de teste 6: Usuário compartilha o CovidApp nas redes sociais

Pré-requisitos

Android

- Possuir o CovidApp instalado para Android;
- Ter finalizado com sucesso o onboarding;
- Possuir instaladas no mesmo dispositivo as seguintes redes sociais, tendo logado nas que necessitam login:
  - Facebook;
  - Twitter;
  - LinkedIn;
  - Messenger;
  - WhatsApp;
  - Aplicativo de SMS;
  - Aplicativo de e-mail;
- Fechar qualquer instância já aberta do app.

iOS

- Possuir o CovidApp instalado para **iOS**;
- Ter finalizado com sucesso o onboarding;
- Possuir instaladas no mesmo dispositivo as seguintes redes sociais, tendo logado nas que necessitam login:
  - Facebook;
  - Twitter;
  - WhatsApp;
  - Aplicativo nativo de SMS;
  - Aplicativo nativo de e-mail;
- Fechar qualquer instância já aberta do app.

Descrição

Este teste tem como objetivo verificar se o usuário consegue compartilhar o CovidApp em todas as redes sociais disponíveis.

Android

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a MainActivity.
2	Clicar no botão "Facebook".	É aberto o Facebook pronto para publicação de um post com o link para o GitHub do Covid Community

		Alert.
3	Cancelar e descartar o post. Retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.
4	Clicar no botão "Twitter".	É aberto o Twitter pronto para publicação de um tweet com o link para o GitHub do Covid Community Alert.
5	Cancelar o tweet sem salvar o rascunho e retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.
6	Clicar no botão "LinkedIn".	É aberto um diálogo inferior do sistema (Android) solicitando que o usuário escolha entre "Compartilhar em uma publicação" ou "Mensagem privada".
7	Clicar em "Compartilhar em uma publicação".	É aberto o LinkedIn pronto para compartilhar uma publicação com o link para o GitHub do Covid Community Alert.
8	Cancelar a publicação e retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.
9	Clicar no botão "LinkedIn" novamente.	É aberto um diálogo inferior do sistema (Android) solicitando que o usuário escolha entre "Compartilhar em uma publicação" ou "Mensagem privada".
10	Clicar em "Mensagem privada".	É aberto o LinkedIn solicitando o destinatário para quem se deseja enviar a nova mensagem.
11	Selecione qualquer contato.	Será preenchida a mensagem com o link para o GitHub do Covid Community Alert.
12	Cancelar o envio da mensagem e retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.
13	Clicar no botão "WhatsApp".	É aberta a janela do WhatsApp para que selecione um contato.
14	Selecionar um contato e clicar em "Avançar".	É aberta uma caixa de texto que já vem com o link para o GitHub do Covid Community Alert.
15	Cancelar o envio da mensagem e retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.
16	Clicar no botão "SMS".	É aberta uma janela do app de SMS que permite informar o contato para o qual se quer enviar a mensagem, que virá preenchida com o link para o GitHub do Covid Community Alert.
17	Cancelar o envio da mensagem e retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.

18	Clicar no botão "E-mail".	É aberto o app de e-mail que permite informar o destinatário para envio do e-mail, cujo corpo vem preenchido com o link para o GitHub do Covid Community Alert.
19	Cancelar o envio do e-mail e retornar ao CovidApp caso isso já não seja feito automaticamente.	Volta-se à MainActivity.

iOS

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a MainActivity.
2	Clicar no botão "Facebook".	É aberto um diálogo inferior do sistema (iOS) que permite o compartilhamento.
3	Clicar no ícone do app Facebook.	É aberto o Facebook pronto para publicação de um post com o link para o GitHub do Covid Community Alert.
4	Cancelar e descartar o post. Fechar o diálogo do sistema para compartilhamento.	Volta-se à MainActivity.
5	Clicar no botão "Twitter".	É aberto um diálogo inferior do sistema (iOS) que permite o compartilhamento.
6	Clicar no ícone do app Twitter.	É aberto o Twitter pronto para publicação de um tweet com o link para o GitHub do Covid Community Alert.
7	Cancelar o tweet. Fechar o diálogo do sistema para compartilhamento.	Volta-se à MainActivity.
8	Clicar no botão "WhatsApp".	É aberto um diálogo inferior do sistema (iOS) que permite o compartilhamento.
9	Clicar no ícone do app WhatsApp.	É aberta a janela do WhatsApp para que selecione um contato.
10	Selecionar um contato e clicar em "Avançar".	É aberta uma caixa de texto que já vem com o link para o GitHub do Covid Community Alert.
11	Clicar em "Voltar" e em "Cancelar". Fechar o diálogo do sistema para compartilhamento.	Volta-se à MainActivity.
12	Clicar no botão "SMS".	É aberta uma janela que permite informar o contato para o qual se quer enviar a mensagem que vem preenchida com o link para o GitHub do Covid Community Alert.
13	Clicar em "Cancelar".	Volta-se à MainActivity.
14	Clicar no botão "E-mail".	É aberta uma janela que permite informar os campos do e-mail cujo corpo vem preenchido com o link para o GitHub do Covid Community Alert.
15	Clicar em "Cancelar" e em "Apagar Rascunho".	Volta-se à MainActivity.

📋 Caso de teste 7: Usuário entende como funciona o app

Pré-requisitos

- Possuir o CovidApp instalado para Android ou iOS;
- Ter finalizado com sucesso o onboarding;
- Fechar qualquer instância já aberta do app.

Descrição

Este teste tem como objetivo verificar se o usuário consegue acessar a página de “Como funciona” a partir da MainActivity.

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a MainActivity.
2	Clicar no botão "Como funciona".	É iniciada e exibida a HowItWorksActivity

📋 Caso de teste 8: Usuário reativa as permissões caso necessário

Pré-requisitos

- Possuir o CovidApp instalado para Android ou iOS;
- Ter finalizado com sucesso o onboarding;

Descrição

Este teste tem como objetivo verificar se o app solicita novamente as permissões que eventualmente tenham sido desativadas/revogadas em algum momento após o onboarding.

Android

Passo	Ação	Resultado esperado
1	**Desativar o bluetooth do dispositivo**.	O bluetooth é desativado.
2	**Desativar a permissão de local** para o CovidApp (o processo varia entre fabricantes e versões do Android, mas de modo geral é preciso: ir nas config. do dispositivo > config. de aplicativos > config. específicas do CovidApp > permissões do app > desativar "Local").	O app tem a permissão de local revogada.
3	Abrir o app	É iniciada a MainActivity, na qual é exibido um diálogo pedindo a ativação do bluetooth.
4	Clique em "OK" no diálogo.	É iniciada e exibida a BluetoothActivity.
5	Clique em "Ligar bluetooth" e conceda a permissão.	Volta-se à MainActivity, na qual é exibido um diálogo pedindo a ativação da localização.
6	Clique em "OK" no diálogo.	É iniciada e exibida a DistanceLogActivity.
7	Clique em "Dar permissão" e conceda a permissão.	Volta-se à MainActivity.

iOS

Passo	Ação	Resultado esperado
1	Desativar o bluetooth do dispositivo.	O bluetooth é desativado.
2	Em Ajustes, role até encontrar "CovidApp - Covid Community Alert". Dentro dessa opção, clicar em "Localização" e selecionar "Nunca".	O app tem a permissão de local revogada.
3	Clicar em Voltar e desativar o switch de "Bluetooth".	O app tem a permissão de bluetooth revogada.

4	Clicar em "Notificações" e desativar o switch de "Permitir notificações".	O app tem a permissão de notificações revogada.
5	Abrir o app	É iniciada a MainActivity, mas no lugar do QR Code há um texto para cada permissão necessária, demonstrando a incapacidade de acessá-las.
6	Clicar no botão "Conceder permissão bluetooth".	É iniciada e exibida a BluetoothActivity.
7	Clicar no botão "Ligar o bluetooth", clicar em OK no diálogo que aparece. Na página de ajustes que é aberta, ativar o switch de "Bluetooth". Voltar ao CovidApp.	Volta-se à MainActivity.
8	Clicar no botão "Conceder permissão de notificações".	É iniciada e exibida a NotificationsActivity.
9	Clicar no botão "Permitir notificações", clicar em OK no diálogo que aparece. Na página de ajustes que é aberta, clicar em Notificações, ativar o switch de "Permitir notificações". Voltar ao CovidApp.	Volta-se à MainActivity.
10	Clicar no botão "Conceder monitoramento de distância".	É iniciada e exibida a DistanceLogActivity.
11	Clicar no botão "Conceder permissão" e clicar no botão OK no diálogo que aparece explicando como ativar. Clicar em "Localização" e selecionar "Sempre". Voltar para o CovidApp e clicar novamente em "Conceder permissão".	Volta-se à MainActivity, agora exibindo o QR Code do paciente.

📋 Caso de teste 9: As interações são registradas corretamente em CSVs do Amazon S3 bucket

Pré-requisitos

- Possuir o CovidApp instalado para Android ou iOS em pelo menos 2 dispositivos (de acordo com cada subcaso abaixo);
- Ter finalizado com sucesso o onboarding do CovidApp;
- Possuir as credenciais e permissões de leitura necessárias para acessar o S3 Bucket do app na AWS;
- Ter configurado uma forma de visualizar os arquivos do S3 Bucket para acessar os CSVs, como a que explicamos aqui.

Descrição

Este teste tem como objetivo verificar se, feitas certas interações entre dois ou mais dispositivos, as mesmas são registradas corretamente nos arquivos CSV que se encontram no S3 Bucket do app na AWS.

Observações

Os CSVs seguem a seguinte “estrutura”:

Abreviatura	Significado	Tipo de dado
i	ID de um dispositivo	LONG
o	ID do outro dispositivo	LONG
w	UNIX Time em segundos	LONG
w_plus_t	UNIX Time em segundos do fim da interação	LONG
t	Duração da interação (em segundos)	LONG
r	Valor RSSI -- intensidade do sinal	DOUBLE
s	Distância da interação (em metros)	DOUBLE
x	Latitude no momento da interação	DOUBLE (opcional)
y	Longitude no momento da interação	DOUBLE (opcional)
p	“a” para Android e “i” para iOS	STRING

As grandezas consideradas para este caso de teste são distância e tempo. Ao variá-las dentro de alguns intervalos formam-se 9 subcasos, com um total de 27 combinações possíveis (quando levado em conta, também, o número de dispositivos), conforme mostra a tabela abaixo.

Subcaso	Variante	Distância (d)	Tempo	N. de dispositivos
1	1.1	d < 0,40 m	1 min	2
	1.2	d < 0,40 m	1 min	3
	1.3	d < 0,40 m	1 min	5
2	2.1	d < 0,40 m	3 min	2
	2.2	d < 0,40 m	3 min	3
	2.3	d < 0,40 m	3 min	5
3	3.1	d < 0,40 m	10 min	2
	3.2	d < 0,40 m	10 min	3
	3.3	d < 0,40 m	10 min	5
4	4.1	0,40 m ≤ d ≤ 2 m	1 min	2
	4.2	0,40 m ≤ d ≤ 2 m	1 min	3
	4.3	0,40 m ≤ d ≤ 2 m	1 min	5
5	5.1	0,40 m ≤ d ≤ 2 m	3 min	2
	5.2	0,40 m ≤ d ≤ 2 m	3 min	3
	5.3	0,40 m ≤ d ≤ 2 m	3 min	5
6	6.1	0,40 m ≤ d ≤ 2 m	10 min	2
	6.2	0,40 m ≤ d ≤ 2 m	10 min	3
	6.3	0,40 m ≤ d ≤ 2 m	10 min	5
7	7.1	d ≥4 m	1 min	2
	7.2	d ≥4 m	1 min	3
	7.3	d ≥4 m	1 min	5
8	8.1	d ≥4 m	3 min	2
	8.2	d ≥4 m	3 min	3
	8.3	d ≥4 m	3 min	5
9	9.1	d ≥4 m	10 min	2
	9.2	d ≥4 m	10 min	3
	9.3	2,00 m < d	10 min	5

Instruções

As instruções abaixo são genéricas, podendo ser reproduzidas para qualquer **subcaso + variante**, com seus respectivos valores para **distância** e **tempo** especificados na tabela acima. Dessa forma, o

resultado esperado mudará de acordo com cada **subcaso + variante**, e também cada sistema operacional dos dispositivos utilizados (ver “Notas sobre o Resultado esperado” abaixo).

Passo	Ação	Resultado esperado
1	Com os dispositivos distantes (a pelo menos 5 metros), ativar o bluetooth em todos eles.	O CovidApp começa a rodar em segundo plano em cada dispositivo.
2	Posicionar os dispositivos conforme o **subcaso + variante** em teste, respeitando a **distância** necessária entre eles.	O CovidApp de cada dispositivo reconhecerá os demais dispositivos ao redor caso eles estejam a um raio reconhecível e iniciará a contagem de tempo da interação.
3	Aguardar o **tempo** necessário e, então, afastar todos os dispositivos, de forma que nenhum dispositivo esteja a menos de 5 metros de distância de outro.	As interações são registradas e enviadas para o servidor para processamento.
4	Aguardar o período de espera para que os logs do servidor NGINX sejam processados e enviados para o Amazon S3 Bucket. Visualizar os CSVs com as interações processadas pelo Log Processor (usando os comandos presentes no AWS CLI ou, em posse da permissão necessária, o Dashboard da AWS).	O arquivo CSV contendo as interações será localizado para que as mesmas serão verificadas.

Notas sobre o Resultado esperado

- Por via de regra, o contato entre dois ou mais dispositivos iOS pela proximidade do seu sinal Bluetooth não é registrado pelo CovidApp, pois eles não conseguem se reconhecer. Portanto, o teste, quando feito **apenas entre dispositivos iOS, é caracterizado como “Não-registrado”**.

🔍 Resultados

Legenda
✔️ - Teste passou
❌ - Teste falhou
❓ - Teste não pôde ser realizado

Android

Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: Asus Zenfone 3 (Android 8.0.0)

✔️ CT-1: Usuário realiza onboarding

✔️ CT-2: Usuário interrompe onboarding

✔️ CT-3: Usuário pula as etapas do onboarding

✔️ CT-4: Usuário obtém mais informações sobre status

✔️ CT-5: Usuário gera seu código de paciente

✔️ CT-6: Usuário compartilha o CovidApp nas redes sociais

✔️ CT-7: Usuário entende como funciona o app

✔️ CT-8: Usuário reativa as permissões caso necessário

Android

Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: Motorola G8 Play (Android 9)

✔️ CT-1: Usuário realiza onboarding

✔️ CT-2: Usuário interrompe onboarding

✔️ CT-3: Usuário pula as etapas do onboarding

✔️ CT-4: Usuário obtém mais informações sobre status

✔️ CT-5: Usuário gera seu código de paciente

✔️ CT-6: Usuário compartilha o CovidApp nas redes sociais

✔️ CT-7: Usuário entende como funciona o app

iOS

Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: iPhone XR (iOS 13.6)

✔️ CT-1: Usuário realiza onboarding

❌ CT-2: Usuário interrompe onboarding

O app **crashou** após permitir as notificações no diálogo do sistema.
No entanto, ao abrí-lo novamente, a MainActivity exibe o QR Code do paciente como esperado.

Passo	Ação	Resultado esperado
1	Abrir o app.	É iniciada e exibida a WelcomeActivity.
2	Clicar no botão "Vamos começar".	É iniciada e exibida a BluetoothActivity.
3	Clicar no botão "Ligar o bluetooth" e permitir o uso do bluetooth no diálogo do sistema que aparece.	É iniciada e exibida a DistanceLogActivity.
4	Encerrar a aplicação e abrí-la novamente.	É iniciada e exibida a DistanceLogActivity.
5	Clicar no botão "Conceder permissão" e permitir o uso da localização *durante o uso do app* no diálogo do sistema que aparece. Clicar novamente em "Conceder permissão".	É exibida novamente a DistanceLogActivity porém agora com texto que explica a necessidade de "sempre permitir" o uso da localização.
6	Clicar novamente em "Conceder permissão". Clicar em OK no diálogo que aparece explicando como ativar a permissão. Clicar em "Localização" e selecionar "Sempre". Voltar para o CovidApp e clicar em "Conceder permissão".	É iniciada e exibida a NotificationsActivity.
7	Encerrar a aplicação e abrí-la novamente.	É iniciada e exibida a MainActivity, porém com um texto para cada permissão indicando seu status.
8	Clicar em "Conceder permissão de notificações", clicar em "Permitir notificações" e permitir no diálogo do sistema que aparece.	Volta-se à MainActivity, agora exibindo o QR Code do paciente.

✔️ CT-3: Usuário pula as etapas do onboarding

✔️ CT-4: Usuário obtém mais informações sobre status

✔️ CT-5: Usuário gera seu código de paciente

✔️ CT-6: Usuário compartilha o CovidApp nas redes sociais

✔️ CT-7: Usuário entende como funciona o app

❌ CT-8: Usuário reativa as permissões caso necessário

Não é feito o retorno esperado à MainActivity. O app permanece na NotificationsActivity e quando se clica em "Permitir notificações" novamente, aparece um diálogo que explica o propósito das notificações e, ao clicar em OK, leva de volta aos ajustes.

A ativação da permissão de notificações não é reconhecida, somente quando fechei e abri novamente a aplicação.

...	...	...
8	Clicar no botão "Conceder permissão de notificações".	É iniciada e exibida a NotificationsActivity.
9	Clicar no botão "Permitir notificações", clicar em OK no diálogo que aparece. Na página de ajustes que é aberta, clicar em Notificações, ativar o switch de "Permitir notificações". Voltar ao CovidApp.	Volta-se à MainActivity.
10	Clicar no botão "Conceder monitoramento de distância".	É iniciada e exibida a DistanceLogActivity.
...	...	...

iOS

Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: iPhone 6S (iOS 12.1.4)

✔️ CT-1: Usuário realiza onboarding

✔️ CT-2: Usuário interrompe onboarding

✔️ CT-3: Usuário pula as etapas do onboarding

✔️ CT-4: Usuário obtém mais informações sobre status

✔️ CT-5: Usuário gera seu código de paciente

✔️ CT-6: Usuário compartilha o CovidApp nas redes sociais

✔️ CT-7: Usuário entende como funciona o app

Data: 05/09/2020 e 06/09/2020
Versão: Firebase (Android) e TestFlight (iOS)
Dispositivos utilizados:

- **XR:** iPhone XR (iOS 13.6.1), **RN:** Redmi Note 8 (Android 9.0) e **ZF:** Zenfone 3 (Android 8.0)
- **6S:** iPhone 6S (iOS 12.1.4), **G7:** Motorola G7 (Android 10.0) e **G8:** Motorola G8 (Android 9.0)

❌ CT-9: As interações são registradas corretamente em CSVs do Amazon S3 bucket

Legenda quanto às interações nos CSVs:
Incorretas
Parcialmente corretas
Corretas

Subcaso	Variante	Distância (d)	Tempo	N. de dispositivos
1	1.1	d < 0,40 m	1 min	2
	1.2	d < 0,40 m	1 min	3
	1.3	d < 0,40 m	1 min	5
2	2.1	d < 0,40 m	3 min	2
	2.2	d < 0,40 m	3 min	3
	2.3	d < 0,40 m	3 min	5
3	3.1	d < 0,40 m	10 min	2
	3.2	d < 0,40 m	10 min	3
	3.3	d < 0,40 m	10 min	5
4	4.1	0,40 m ≤ d ≤ 2 m	1 min	2
	4.2	0,40 m ≤ d ≤ 2 m	1 min	3
	4.3	0,40 m ≤ d ≤ 2 m	1 min	5
5	5.1	0,40 m ≤ d ≤ 2 m	3 min	2
	5.2	0,40 m ≤ d ≤ 2 m	3 min	3
	5.3	0,40 m ≤ d ≤ 2 m	3 min	5
6	6.1	0,40 m ≤ d ≤ 2 m	10 min	2
	6.2	0,40 m ≤ d ≤ 2 m	10 min	3
	6.3	0,40 m ≤ d ≤ 2 m	10 min	5
7	7.1	d ≥4 m	1 min	2
	7.2	d ≥4 m	1 min	3
	7.3	d ≥4 m	1 min	5
8	8.1	d ≥4 m	3 min	2
	8.2	d ≥4 m	3 min	3
	8.3	d ≥4 m	3 min	5
9	9.1	d ≥4 m	10 min	2
	9.2	d ≥4 m	10 min	3

	9.3	2,00 m < d	10 min	5

Seguem os logs dos nossos testes (horários/ações) e as interações dos respectivos subcasos, retiradas dos CSVs. As colunas “Duração” e “Distância” das tabelas com as interações estão destacadas para melhor legibilidade.

Legenda das tabelas de interações a seguir:
Em itálico: O que se acredita ser ruído ou interação inesperada
Em negrito: O que se acredita ser de fato o registro das interações executadas na testagem do respectivo subcaso (podendo ou não estar correto)

Variante 1.1

Dispositivos: 6S: 17-75 (Nenhum risco) G8: 19-92 (Nenhum risco)
22:08 | 05/09/2020 - Os dispositivos foram aproximados a uma distância de 30 centímetros
22:09 | 05/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 2.1

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada)
18:00 - Foram aproximados lado-a-lado RN e XR por 3 min
18:03 - Foram afastados (> 5m) RN e XR

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.
17:59:55	*179*	*191*	*5*	*41*	*0.1*	*i*
17:59:55	*191*	*179*	*5*	*41*	*0.1*	*i*
17:59:55	*179*	*191*	*8*	*41*	*0.1*	*i*
17:59:55	*191*	*179*	*8*	*41*	*0.1*	*i*
18:00:04	**179**	**191**	**184**	**33**	**0.0**	**i**
18:00:04	**191**	**179**	**184**	**33**	**0.0**	**i**
18:02:18	**191**	**179**	**112**	**29**	**0.0**	**a**
18:02:18	**179**	**191**	**112**	**29**	**0.0**	**a**
18:03:04	*179*	*191*	*61*	*88*	*48.5*	*i*

18:03:04	*191*	*179*	*61*	*88*	*48.5*	*i*

Conclusão: O iOS registrou a interação corretamente como 3min4s, mas o Android registrou como 1min52s (só a “segunda metade” da interação, a grosso modo). Ambos a 0.0 de distância.

Variante 2.2

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco) G7: 14-83 (Risco Médio)
9:20 | 06/09/2020 - Os dispositivos foram aproximados a uma distância de 30 centímetros
9:23 | 06/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 3.1

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco)
22:17 | 05/09/2020 - Os dispositivos foram aproximados a uma distância de 30 centímetros
22:28 | 05/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.
22:17:59	**192**	**1177**	**28**	**28**	**0.0**	**a**
22:17:59	**177**	**192**	**28**	**28**	**0.0**	**a**
22:26:58	**192**	**177**	**58**	**40**	**0.0**	**a**
22:26:58	**177**	**192**	**58**	**40**	**0.0**	**a**
22:27:17	**192**	**177**	**0**	**43**	**0.0**	**a**
22:27:17	**177**	**192**	**0**	**43**	**0.0**	**a**
22:27:20	**192**	**177**	**50**	**46**	**0.1**	**a**
22:27:20	**177**	**192**	**50**	**46**	**0.1**	**a**

Conclusão: Os dados foram enviados e registrados com sucesso, porém, ao invés de marcar 10 minutos de contato, percebe-se 4 marcações menores de tempo que desenrolam-se dentro desse período. A marcação de Sistema Operacional também apresenta erro, visto que, o dispositivo com ID 177 possui sistema iOS mas foi marcado como Android.

Variante 4.1

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada)
18:09 - Foram aproximados a 1m RN e XR por 1 min
18:10 - Foram afastados (> 5m) RN e XR

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.
18:09:37	**179**	**191**	**14**	**58**	**0.9**	**i**
18:09:37	**191**	**179**	**14**	**58**	**0.9**	**i**
18:13:16	*179*	*191*	*5*	*88*	*40.8*	*i*
18:13:16	*191*	*179*	*5*	*88*	*40.8*	*i*

Conclusão: O iOS registrou erroneamente a interação como 14s a 0.9 de distância. O Android não registrou nada desta interação.

Variante 4.2

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco) G7: 14-83 (Risco Médio)
9:26 | 06/09/2020 - Os dispositivos foram aproximados a uma distância de 1 metro
9:27 | 06/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 5.2

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco) G7: 14-83 (Risco Médio)
9:32 | 06/09/2020 - Os dispositivos foram aproximados a uma distância de 1 metro
9:35 | 06/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 6.1

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada)
18:15 - Foram aproximados a 1m RN e XR por 10 min

18:25 - Foram afastados (> 5m) RN e XR

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.
18:15:17	**179**	**191**	**13**	**50**	**0.3**	**i**
18:15:17	**191**	**179**	**13**	**50**	**0.3**	**i**
18:23:49	**179**	**191**	**10**	**44**	**0.2**	**i**
18:23:49	**191**	**179**	**10**	**44**	**0.2**	**i**

Conclusão: A interação, que era de 10 min a 1 metro, só foi registrada pelo iOS e com dados incorretos: duração de 23s (13s + 10s), a 0.2~0.3 de distância.

Variante 6.2

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco) G7: 14-83 (Risco Médio)
22:36 | 05/09/2020 - Os dispositivos foram aproximados a uma distância de 1 metro
22:46 | 05/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 7.1

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada)
18:30 - Foram aproximados a 4m RN e XR por 1 min
18:31 - Foram afastados (> 5m) RN e XR

Conclusão: Não foi registrado por nenhum dos dispositivos.

Variante 7.2

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco)
22:53 | 05/09/2020 - Os dispositivos foram aproximados a uma distância de 4 metros
22:54| 06/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 8.1

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada)
18:36 - Foram aproximados a 4m RN e XR por 3 min
18:39 - Foram afastados (> 5m) RN e XR

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.
18:36:16	**179**	**191**	**5**	**89**	**46.4**	**i**
18:36:16	**191**	**179**	**5**	**89**	**46.4**	**i**

Conclusão: A interação, que era de 3 min a 4 metros de distância, foi registrada somente pelo iOS e erroneamente, como sendo de 5 segundos a 46.4 de distância.

Variante 8.2

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco) G7: 14-83 (Risco Médio)
9:40 | 06/09/2020 - Os dispositivos foram aproximados a uma distância de 4 metros
9:44 | 06/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Variante 9.1

Dispositivos: 6S: 17-75(Nenhum risco) G8: 19-92 (Nenhum risco)
23:00 | 05/09/2020 - Os dispositivos foram aproximados a uma distância de 4 metros
23:10 | 05/09/2020 - Os dispositivos são separados e colocados em cômodos diferentes

Os Logs não foram localizados

Conclusão: Os dados da interação não foram enviados e/ou processados pelo Log Processor

Logs “perdidos”

Não encontrei correspondência desses logs com nenhum dos subcasos.

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.

18:44:18	**179**	**191**	**86**	**87**	**35.4**	**i**
18:44:18	**191**	**179**	**86**	**87**	**35.4**	**i**
18:44:18	**179**	**191**	**86**	**87**	**35.4**	**i**
18:44:18	**191**	**179**	**86**	**87**	**35.4**	**i**
18:48:23	**183**	**191**	**188**	**88**	**7.8**	**a**
18:48:23	**191**	**183**	**188**	**88**	**7.8**	**a**

Variante 1.2

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada) ZF: 18-32- (Médio risco detectado)
18:50 - Foram aproximados lado-a-lado RN, XR e ZF por 1 min
18:51 - Foram afastados (> 5m) RN, XR e ZF

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.
18:50:44	**179**	**183**	**10**	**66**	**3.9**	**i**
18:50:44	**183**	**179**	**10**	**66**	**3.9**	**i**
18:50:50	**179**	**183**	**17**	**52**	**0.5**	**i**
18:50:50	**183**	**179**	**17**	**52**	**0.5**	**i**

Conclusão: A interação entre os 3 dispositivos foi captada somente pelo iOS XR, entre ele e o Android ZF. Sua duração e distância parecem inconsistentes.

Variante 3.2

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada) ZF: 18-32- (Médio risco detectado)
18:56 - Foram aproximados lado-a-lado RN, XR e ZF por 10 min
19:06 - Foram afastados (> 5m) RN, XR e ZF

Pares de interação:
(A) XR e RN
(B) XR e ZF
(C) ZF e RN

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.	
18:55:03	*179*	*183*	*76*	*86*	*31.6*	*i*	*(B) XR – ZF*

18:55:03	*183*	*179*	*76*	*86*	*31.6*	*i*	
18:55:06	*179*	*191*	*78*	*89*	*44.0*	*i*	*(A) XR – RN*
18:55:06	*191*	*179*	*78*	*89*	*44.0*	*i*	
18:57:24	**183**	**191**	**189**	**41**	**0.0**	a	(C) ZF – RN
18:57:24	**191**	**183**	**189**	**41**	**0.0**	a	
18:59:37	**179**	**191**	**184**	**37**	**0.1**	i	(A) XR – RN
18:59:37	**191**	**179**	**184**	**37**	**0.1**	i	
18:59:37	**179**	**183**	**184**	**53**	**0.5**	i	(B) XR – ZF
18:59:37	**183**	**179**	**184**	**53**	**0.5**	i	
19:00:24	**183**	**191**	**189**	**41**	**0.0**	a	(C) ZF – RN
19:00:24	**191**	**183**	**189**	**41**	**0.0**	a	
19:02:37	**179**	**191**	**38**	**38**	**0.1**	i	(A) XR – RN
19:02:37	**191**	**179**	**38**	**38**	**0.1**	i	
19:02:37	**179**	**183**	**38**	**53**	**0.5**	i	(B) XR – ZF
19:02:37	**183**	**179**	**38**	**53**	**0.5**	i	
19:03:24	**183**	**191**	**188**	**41**	**0.0**	a	(C) ZF – RN
19:03:24	**191**	**183**	**188**	**41**	**0.0**	a	
19:06:38	**179**	**183**	**16**	**50**	**0.4**	i	(C) ZF – RN
19:06:38	**183**	**179**	**16**	**50**	**0.4**	i	
19:06:38	**179**	**191**	**16**	**36**	**0.1**	i	(A) XR – RN
19:06:38	**191**	**179**	**16**	**36**	**0.1**	i	
19:08:42	*179*	*183*	*11*	*86*	*52.0*	*i*	*(B) XR – ZF*
19:08:42	*183*	*179*	*11*	*86*	*52.0*	*i*	
19:09:06	*191*	*183*	*15*	*46*	*0.1*	*a*	*(C) ZF – RN*
19:09:06	*183*	*191*	*15*	*46*	*0.1*	*a*	

Conclusão:

O par “(A) XR - RN” totalizou apenas 3min58s de interação com distâncias de [0.1, 0.1, 0.1].
O par “(B) XR - ZF” totalizou apenas 3min42s de interação com distâncias de [0.5, 0.5].
O par “(C) ZF - RN” totalizou 9min42s de interação com distâncias de [0.0, 0.0, 0.0, 0.4].

Variante 5.2

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada) ZF: 18-32- (Médio risco detectado)
19:13 - Foram aproximados a 1m RN, XR e ZF por 3 min
19:16 - Foram afastados (> 5m) RN, XR e ZF

Pares de interação:
(A) XR e RN
(B) XR e ZF
(C) ZF e RN

Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.	
19:12:25	**183**	**191**	**187**	**67**	**1.7**	**a**	(C) ZF – RN
19:12:25	**191**	**183**	**187**	**67**	**1.7**	**a**	
19:14:07	**179**	**191**	**27**	**68**	**3.1**	**i**	(A) XR – RN
19:14:07	**191**	**179**	**27**	**68**	**3.1**	**i**	
19:15:33	**179**	**183**	**13**	**62**	**1.6**	**i**	(B) XR – ZF
19:15:33	**183**	**179**	**13**	**62**	**1.6**	**i**	
19:15:33	**179**	**191**	**13**	**69**	**3.2**	**i**	(A) XR – RN
19:15:33	**191**	**179**	**13**	**69**	**3.2**	**i**	

Conclusão:
O par “(A) XR - RN” totalizou apenas 46s de interação com distâncias de [3.1, 3.2].
O par “(B) XR - ZF” totalizou apenas 13s de interação com distância de [1.6].
O par “(C) ZF - RN” totalizou 3min7s de interação com distância de [1.7].

Observação: Nota-se semelhança com o resultado da variante anterior. As interações em que o par de dispositivos é formado por um sendo iOS e o outro Android foram registradas pelo aparelho iOS, incorretamente nos dois casos. O par Android-Android — neste caso o (C) — registrou a interação corretamente.

Variante 7.2

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada) ZF: 18-32- (Médio risco detectado)
19:22 - Foram aproximados a 4m RN, XR e ZF por 1 min
19:23 - Foram afastados (> 5m) RN, XR e ZF

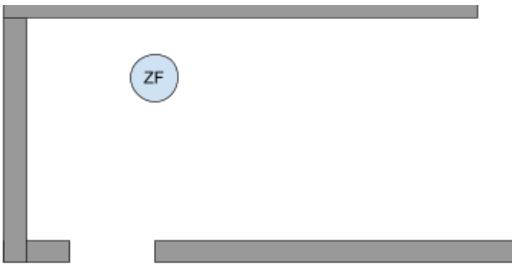
Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.	
19:18:26	*183*	*191*	*2106*	*89*	*8.5*	*a*	
19:18:26	*191*	*183*	*2106*	*89*	*8.5*	*a*	
19:20:09	*183*	*191*	*141*	*92*	*9.7*	*a*	
19:20:09	*191*	*183*	*141*	*92*	*9.7*	*a*	
19:22:12	**179**	**183**	**168**	**83**	**24.5**	**i**	(B) XR – ZF
19:22:12	**183**	**179**	**168**	**83**	**24.5**	**i**	
19:22:22	**183**	**191**	**13**	**96**	**10.2**	**a**	(C) ZF – RN
19:22:22	**191**	**183**	**13**	**96**	**10.2**	**a**	
19:22:25	*183*	*191*	*0*	*98*	*10.2*	*a*	
19:22:25	*191*	*183*	*0*	*98*	*10.2*	*a*	
19:22:26	*183*	*191*	*0*	*98*	*10.4*	*a*	
19:22:26	*191*	*183*	*0*	*98*	*10.4*	*a*	
19:22:27	*183*	*191*	*0*	*93*	*10.5*	*a*	
19:22:27	*191*	*183*	*0*	*93*	*10.5*	*a*	
19:22:29	*183*	*191*	*0*	*95*	*10.6*	*a*	
19:22:29	*191*	*183*	*0*	*95*	*10.6*	*a*	
19:22:29	*183*	*191*	*0*	*95*	*10.6*	*a*	
19:22:29	*191*	*183*	*0*	*95*	*10.6*	*a*	
19:24:51	*179*	*191*	*5*	*90*	*73.5*	*i*	
19:24:51	*191*	*179*	*5*	*90*	*73.5*	*i*	
19:26:22	*179*	*183*	*10*	*87*	*52.7*	*i*	
19:26:22	*183*	*179*	*10*	*87*	*52.7*	*i*	

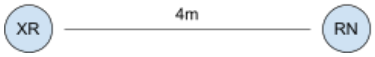
Conclusão: As interações não foram registradas corretamente.

Observações: Presença de interação antes do intervalo de tempo compreendido por este subcaso, com longa duração (2106 segundos = 35min06s) e distância de 8.5. Presença de vários logs com duração de 0 segundos.

Variante 9.2

Dispositivos: RN: 19-11- (Nenhum risco) XR: 17-97 (Infectada) ZF: 18-32- (Médio risco detectado)
19:30 - Foram aproximados a 4m RN, XR e ZF por 10 min (conforme gravura abaixo)
19:40 - Foram afastados (> 5m) RN, XR e ZF





Horário início (UTC-03:00)	ID de um	ID do outro	**Duração**	RSSI	**Distância**	S.O.	
19:28:48	**179**	**191**	**261**	**87**	**42.0**	**i**	(A) XR – RN
19:28:48	**191**	**179**	**261**	**87**	**42.0**	**i**	
19:32:43	**179**	**183**	**264**	**86**	**38.3**	**i**	(B) XR – ZF
19:32:43	**183**	**179**	**264**	**86**	**38.3**	**i**	
19:36:39	**179**	**183**	**162**	**84**	**23.9**	**i**	(B) XR – ZF
19:36:39	**183**	**179**	**162**	**84**	**23.9**	**i**	
19:38:11	**179**	**191**	**159**	**89**	**51.0**	**i**	(A) XR – RN
19:38:11	**191**	**179**	**159**	**89**	**51.0**	**i**	
19:40:36	**179**	**191**	**13**	**79**	**15.0**	**i**	(A) XR – RN
19:40:36	**191**	**179**	**13**	**79**	**15.0**	**i**	
20:05:20	**179**	**191**	**5**	**86**	**31.6**	**i**	(A) XR – RN
20:05:20	**191**	**179**	**5**	**86**	**31.6**	**i**	

Conclusão:

O par “(A) XR - RN” totalizou somente 7min18s de interação com distâncias de [42.0, 51.0, 15.0, 31.6].
O par “(B) XR - ZF” totalizou somente 7min6s de interação com distâncias de [38.3, 23.9].
O par “(C) ZF - RN” (que estava separado pela parede) não teve interações registradas.

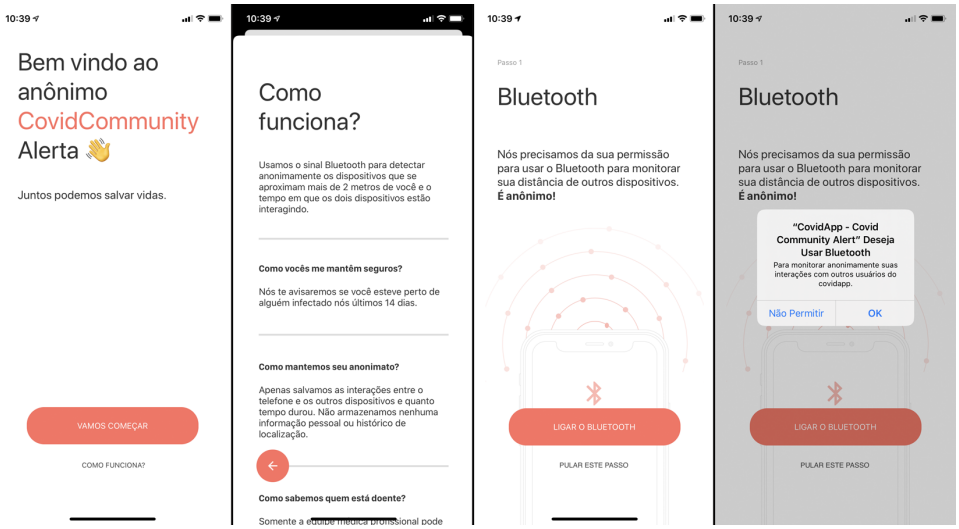
Observações: Todos os registros vieram do iOS. O obstáculo entre ZF e RN pode ter afetado o resultado.

📸 Capturas de tela

iOS

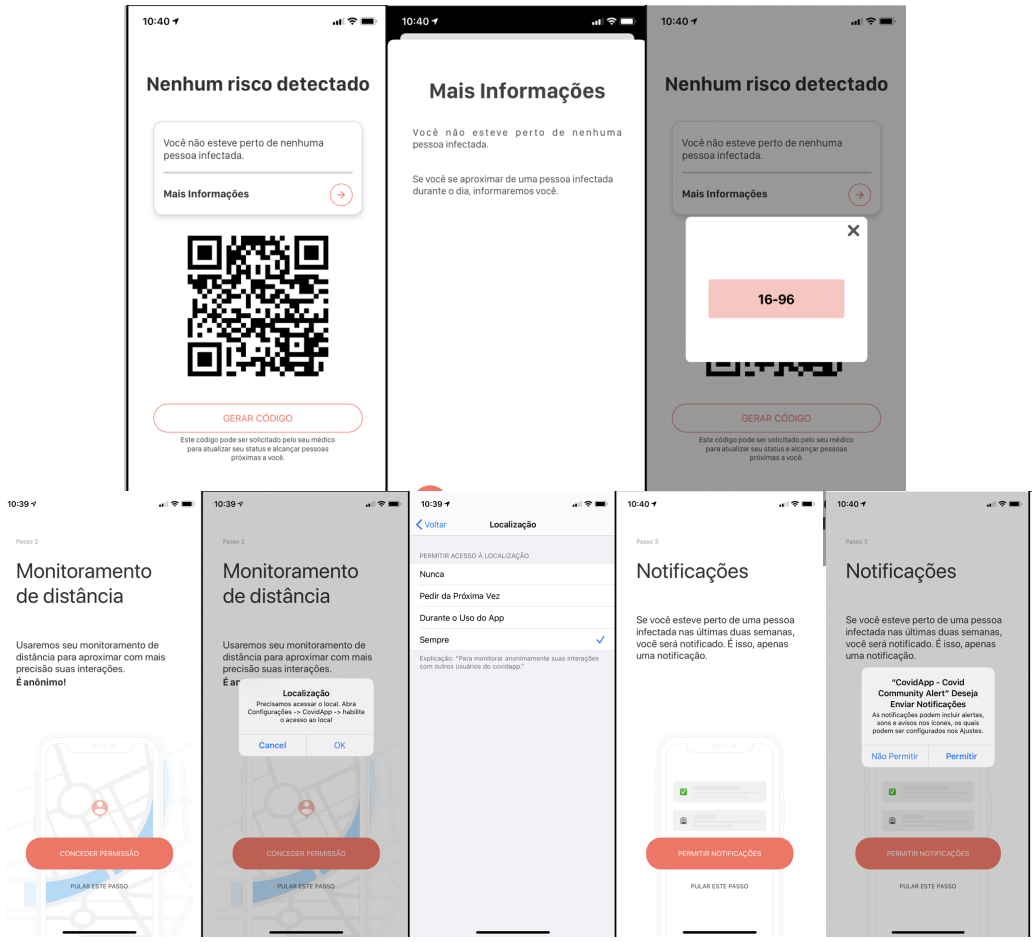
Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: iPhone XR (iOS 13.6)

Imagem 1 - Onboarding do usuário



Fonte: Desenvolvida pelos autores

Imagem 2 - QR Code do usuário, mais informações e geração do código de paciente



Fonte: Desenvolvida pelos autores

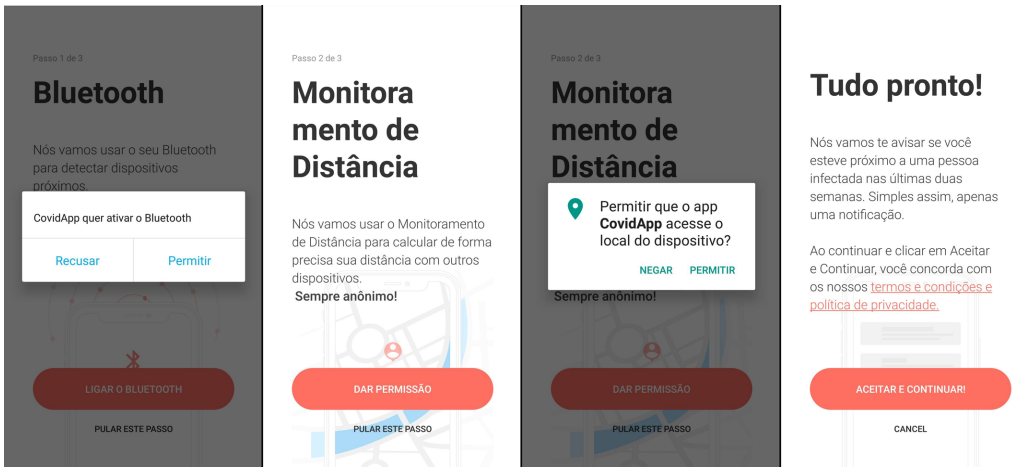
Imagem 3 - Opções para compartilhamento nas redes sociais e página de Como funciona

Fonte: Desenvolvida pelos autores

Android

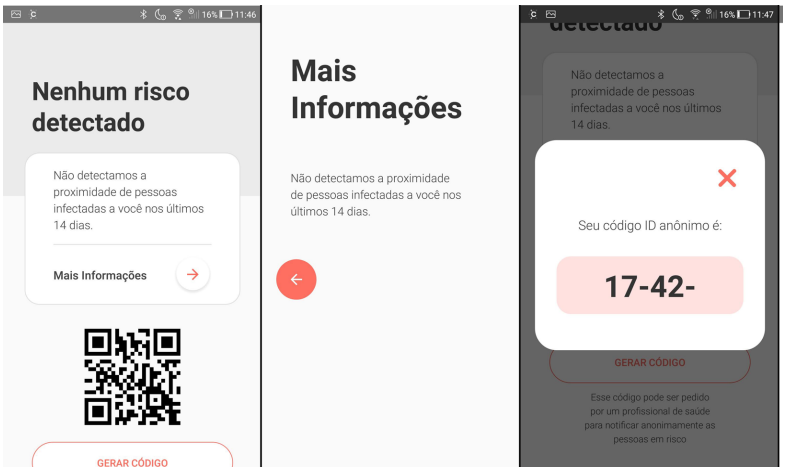
Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: Asus Zenfone 3 (Android 8.0.0)

Imagem 1 - Onboarding do usuário



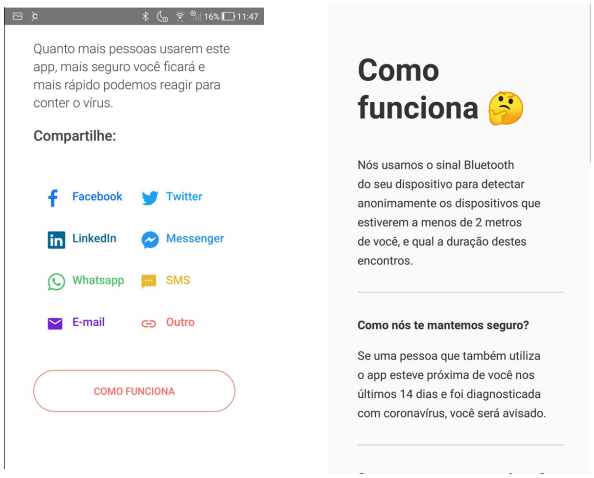
Fonte: Desenvolvida pelos autores

Imagem 2 - QR Code do usuário, mais informações e geração do código de paciente



Fonte: Desenvolvida pelos autores

Imagem 3 - Opções para compartilhamento nas redes sociais e página de Como funciona

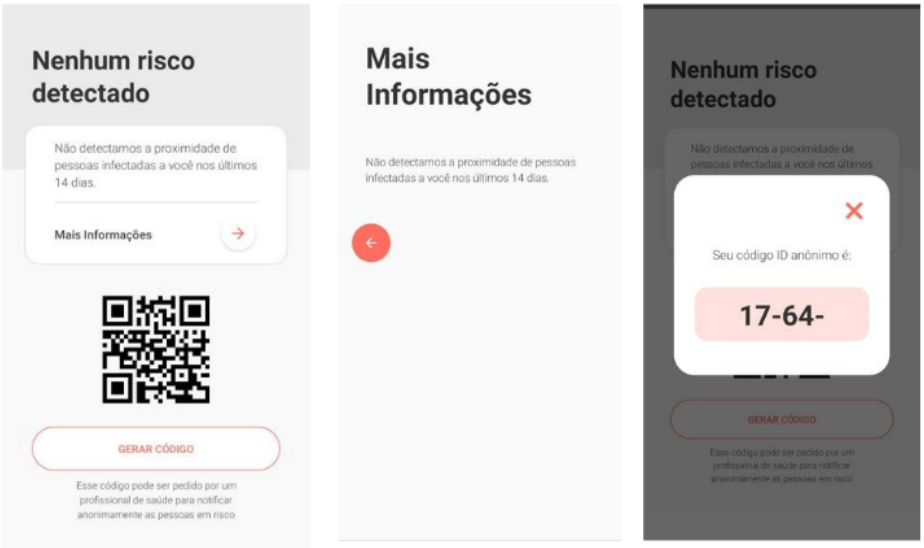


Fonte: Desenvolvida pelos autores

Android

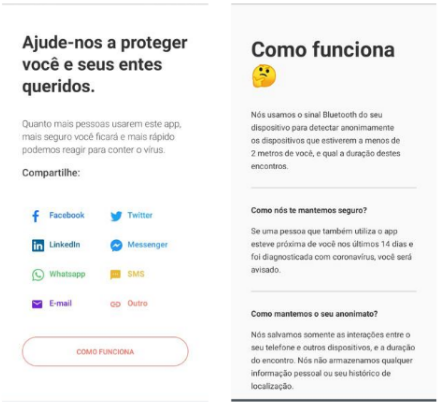
Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: Motorola G8 Play (Android 9)

Imagem 1 - Funções principais



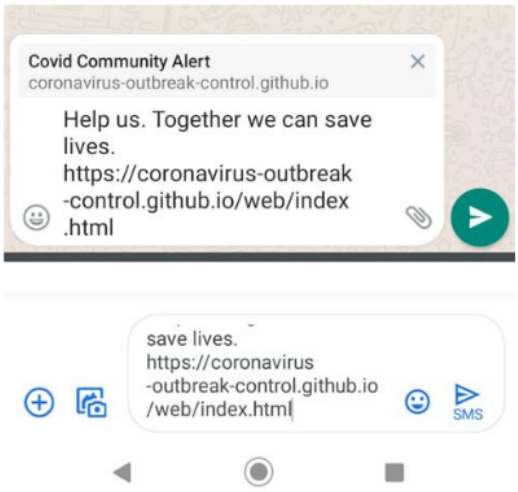
Fonte: Desenvolvida pelos autores

Imagem 2 - Compartilhamento e saiba mais



Fonte: Desenvolvida pelos autores

Imagem 3 - Mensagens de compartilhamento

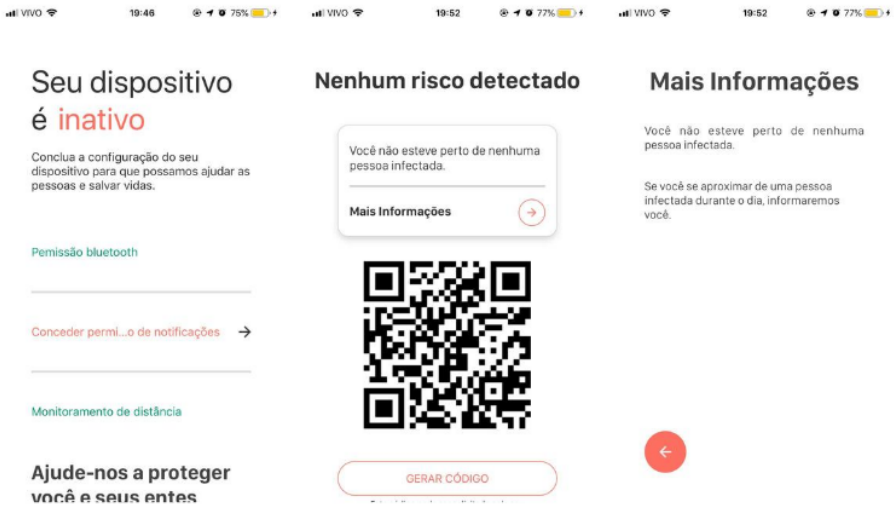


Fonte: Desenvolvida pelos autores

iOS

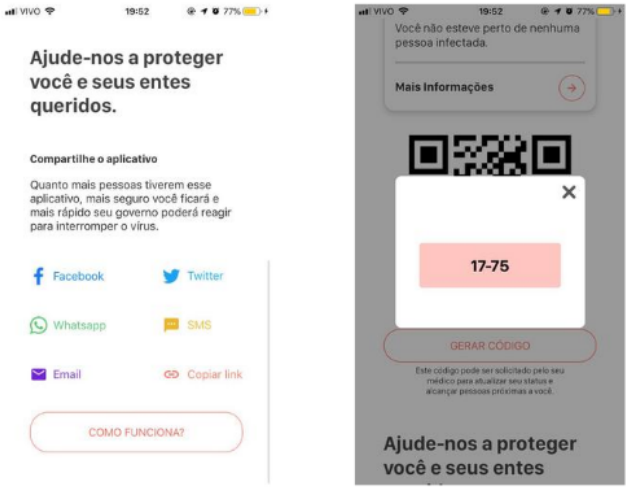
Data: 13/08/2020
Versão: Firebase
Dispositivo utilizado: iPhone 6S (iOS 12.1.4)

Imagem 1 - Tela de configuração descontinuada e tela inicial



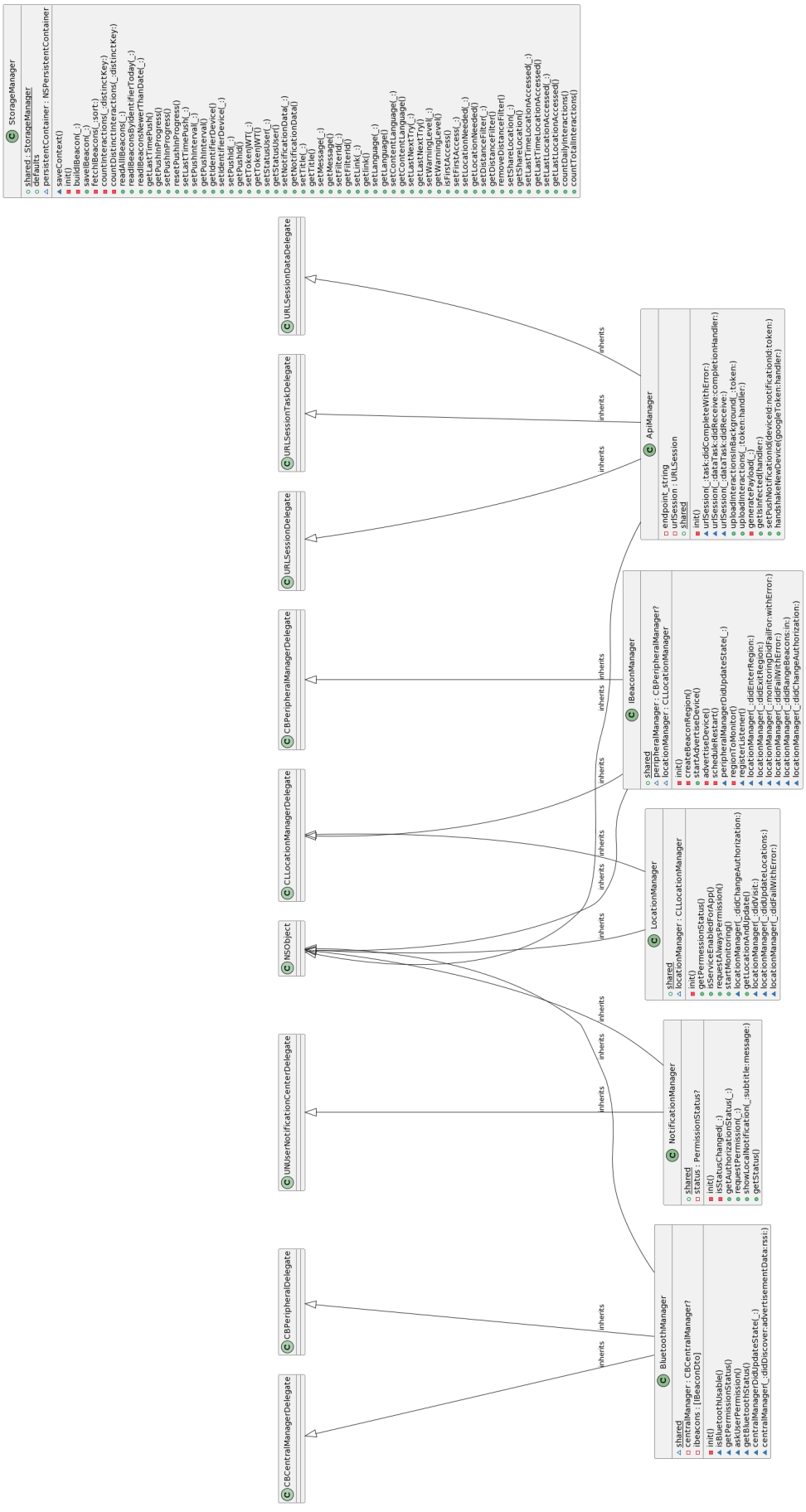
Fonte: Desenvolvida pelos autores

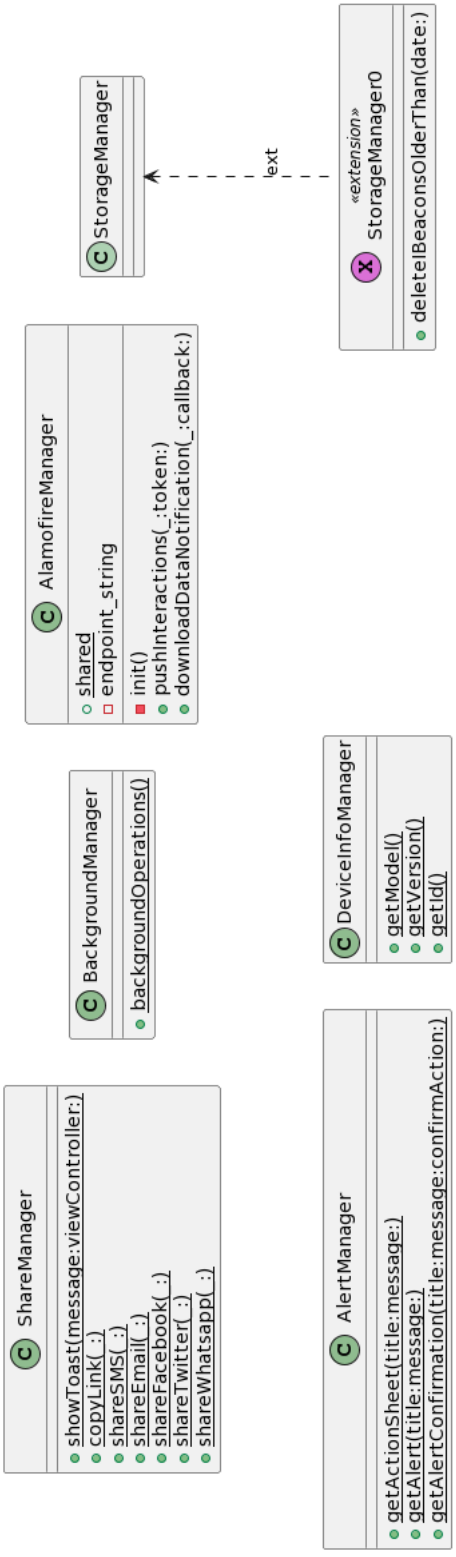
Imagem 2 - ID do usuário e compartilhe



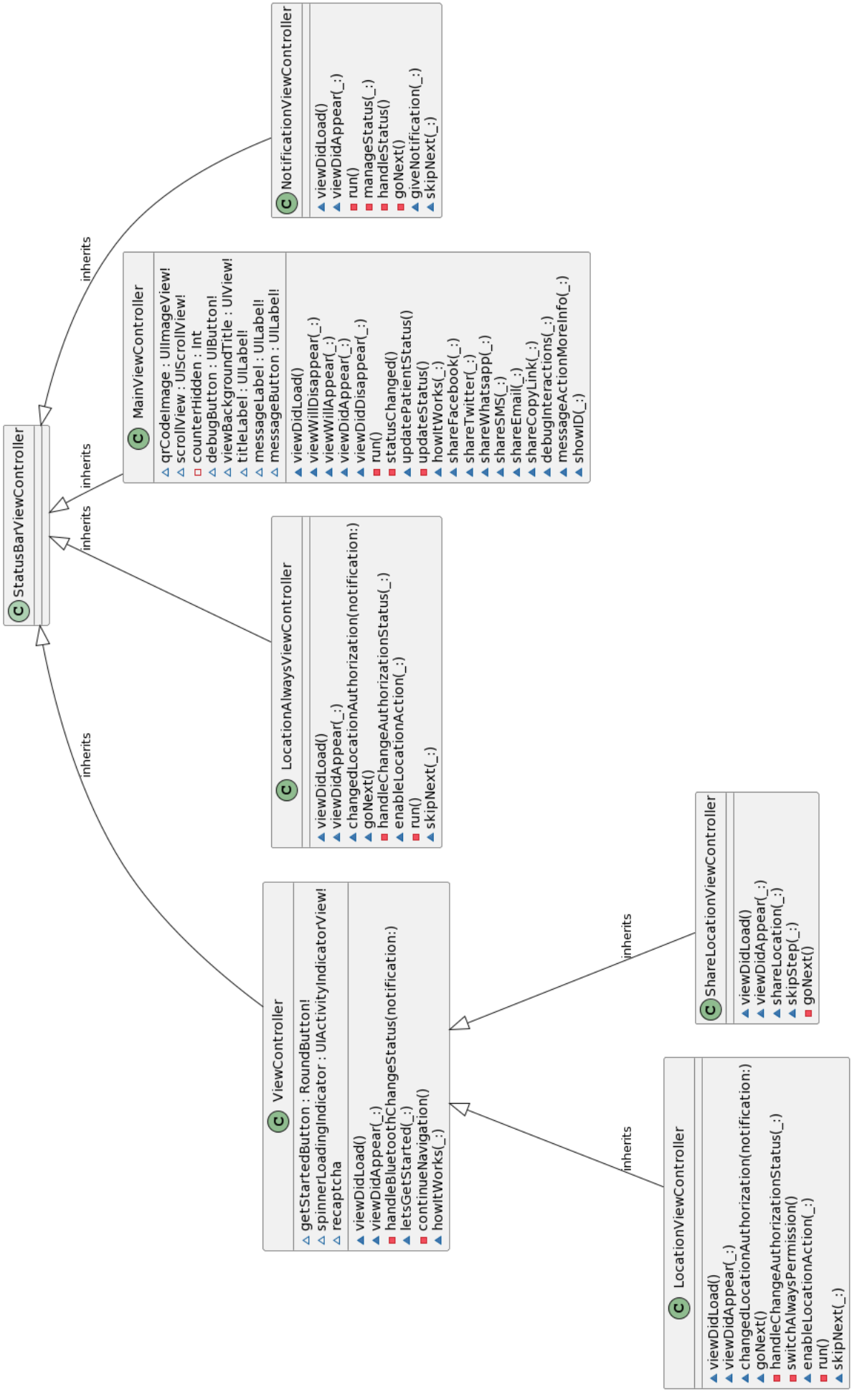
Fonte: Desenvolvida pelos autores

ANEXO B – CLASSES *UML*




StorageManager
«extension»
StorageManager0
deleteIBeaconsOlderThan(date:)
ext
AlamofireManager
shared
endpoint_string
init()
pushInteractions(_:token:)
downloadDataNotification(_:callback:)
BackgroundManager
backgroundOperations()
DeviceInfoManager
getModel()
getVersion()
getId()
ShareManager
showToast(message:viewController:)
copyLink(_:)
shareSMS(_:)
shareEmail(_:)
shareFacebook(_:)
shareTwitter(_:)
shareWhatsapp(_:)
AlertManager
getActionSheet(title:message:)
getAlert(title:message:)
getAlertConfirmation(title:message:confirmAction:)


UITableViewController
BluetoothTableViewController
beacons
viewDidLoad()
numberOfSections(in:)
tableView(_:numberOfRowsInSection:)
tableView(_:cellForRowAt:)
tableView(_:titleForHeaderInSection:)
UIApplicationDelegate
UIResponder
AppDelegate
window : UIWindow?
registerListeners()
application(_:didFinishLaunchingWithOptions:)
scheduledPushInteractions()
handlePushInteractions(_:)
startup()
application(_:performFetchWithCompletionHandler:)
application(_:willFinishLaunchingWithOptions:)
applicationWillResignActive(_:)
applicationDidEnterBackground(_:)
applicationWillEnterForeground(_:)
applicationDidBecomeActive(_:)
applicationWillTerminate(_:)
updatePushId(token:)
application(_:didRegisterForRemoteNotificationsWithDeviceToken:)
application(_:didFailToRegisterForRemoteNotificationsWithError:)
application(_:didReceiveRemoteNotification:)
application(_:didReceiveRemoteNotification:fetchCompletionHandler:)
handleRemoteContent(_:)
MFMessageComposeViewControllerDelegate
MFMailComposeViewControllerDelegate
UIViewController
StatusBarViewController
preferredStatusBarStyle : UIStatusBarStyle
messageComposeViewController(_:didFinishWith:)
mailComposeController(_:didFinishWith:error:)
HowitWorksViewController
scrollView : UIScrollView!
viewDidLoad()
viewDidAppear(_:)
backButton(_:)
BluetoothOffViewController
viewDidLoad()
viewDidAppear(_:)
run()
changedBluetoothStatus(notification:)
handleBluetoothStatus(_:)
openBluetoothAction(_:)
goNext()
bluetoothOff()
bluetoothDeniedOrUnauthorized()
bluetoothNotAvailable()
howCanHelpMoreAction(_:)
skipNext(_:)
LaunchViewController
viewDidLoad()
viewDidAppear(_:)
ViewController
getStartedButton : RoundButton!
spinnerLoadingIndicator : UIActivityIndicatorView!
recaptcha
viewDidLoad()
viewDidAppear(_:)
handleBluetoothChangeStatus(notification:)
letsGetStarted(_:)
continueNavigation()
howItWorks(_:)
inactiveViewController
bluetoothButton : UIButton!
bluetoothLabelButton : UIButton!
bluetoothSeparator : UIView!
locationButton : UIButton!
LocationLabelButton : UIButton!
locationSeparator : UIView!
notificationButton : UIButton!
notificationLabelButton : UIButton!
scrollView : UIScrollView!
colorGreen : UIColor
colorRed : UIColor
viewDidLoad()
viewWillAppear(_:)
viewDidDisappear(_:)
viewWillDisappear(_:)
viewDidAppear(_:)
statusChanged()
refresh()
openMain()
bluetoothAction(_:)
locationAction(_:)
notificationAction(_:)
howItWorks(_:)
shareFacebook(_:)
shareTwitter(_:)
shareWhatsapp(_:)
shareSMS(_:)
shareEmail(_:)
copyLink(_:)
inherits


StatusBarViewController
NotificationViewController
viewDidLoad()
viewDidAppear(_:)
run()
manageStatus(_:)
handleStatus()
goNext()
giveNotification(_:)
skipNext(_:)
MainViewController
qrCodeImage : UIImageView!
scrollView : UIScrollView!
counterHidden : Int
debugButton : UIButton!
viewBackgroundTitle : UIView!
titleLabel : UILabel!
messageLabel : UILabel!
messageButton : UILabel!
viewDidLoad()
viewWillDisappear(_:)
viewWillAppear(_:)
viewDidAppear(_:)
viewDidDisappear(_:)
run()
statusChanged()
updatePatientStatus()
updateStatus()
howItWorks(_:)
shareFacebook(_:)
shareTwitter(_:)
shareWhatsapp(_:)
shareSMS(_:)
shareEmail(_:)
shareCopyLink(_:)
debugInteractions(_:)
messageActionMoreInfo(_:)
showID(_:)
LocationAlwaysViewController
viewDidLoad()
viewDidAppear(_:)
changedLocationAuthorization(notification:)
goNext()
handleChangeAuthorizationStatus(_:)
enableLocationAction(_:)
run()
skipNext(_:)
ViewController
getStartedButton : RoundButton!
spinnerLoadingIndicator : UIActivityIndicatorView!
recaptcha
viewDidLoad()
viewDidAppear(_:)
handleBluetoothChangeStatus(notification:)
letsGetStarted(_:)
continueNavigation()
howItWorks(_:)
ShareLocationViewController
viewDidLoad()
viewDidAppear(_:)
shareLocation(_:)
skipStep(_:)
goNext()
LocationViewController
viewDidLoad()
viewDidAppear(_:)
changedLocationAuthorization(notification:)
goNext()
handleChangeAuthorizationStatus(_:)
switchAlwaysPermission()
enableLocationAction(_:)
run()
skipNext(_:)
inherits
inherits
inherits
inherits
inherits
inherits