Aplicación para proyecto EDIGA

Viviana Luongo
Lia Colombo

Tutoras: Regina Motz
Libertad Tansini

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Montevideo
2021

Resumen

El proyecto EDIGA (Entornos Digitales e Identidades de Género en la Adolescencia) tiene como objetivo analizar el papel que los entornos digitales tienen en la construcción de las identidades de género en la adolescencia. Para realizar este proyecto, resulta de interés conocer el uso que hacen los adolescentes de las redes sociales.
Con este objetivo, se planteó la idea de desarrollar una aplicación móvil que permitiera recolectar información sobre dicho uso, y guardarla en una base de datos para que los investigadores tengan acceso luego. La aplicación creada incluye cuestionarios para obtener información sobre el estado de ánimo de los usuarios (información cualitativa), además de obtener el tiempo de uso que se hace de la red social Instagram. Esta aplicación móvil se desarrolló para Android usando Java.
Se consideró implementar un diario para que los adolescentes pudieran reflejar el uso de Instagram en su dispositivo. Además, se estudió la posibilidad de obtener datos con menos sesgo de forma automática desde el propio dispositivo. Se logró obtener un prototipo de una herramienta para que los adolescentes participantes aporten voluntariamente información acerca de sus interacciones en las redes sociales. En un futuro se podría realizar el trabajo de extracción de actividades en la red Instagram en los marcos éticos y legales acordes para obtener una mayor cantidad de información, se considera que lo investigado en este proyecto puede servir para ello.

Palabras clave— App Android, extracción de datos de Instagram

Índice general

Capítulo 1

Introducción

1.1. Motivación

El proyecto busca obtener información acerca de la interacción entre los adolescentes y su entorno digital. Se investigó en el contexto del proyecto EDIGA, que estudia cómo los entornos digitales influyen en la identidad de género de los adolescentes en diferentes contextos socioculturales: España, México y Uruguay. [67].
Para conseguir este objetivo, la observación directa de las interacciones de los adolescentes en las redes aportaría muchos datos. La recolección manual de esta información es compleja por la cantidad de recursos que se necesitarían, pues requiere observar los perfiles de redes sociales de los sujetos y registrar todo lo que se ve, además de que no serían datos objetivos sino subjetivos de quien observa. Por lo tanto, la motivación principal se centra en la investigación sobre las posibilidades de obtener datos automáticamente y datos que pueda aportar el adolescente mismo, haciendo uso de una aplicación para este fin.

1.2. Problema

El problema consiste en encontrar una forma de automatizar la obtención de información sobre el uso que una persona hace de las redes sociales. Para esto se plantea el uso de una aplicación móvil que realice la recolección de los datos, con un correspondiente servidor y base de datos que recojan lo obtenido. Esta recolección debe hacerse teniendo en cuenta la privacidad de los adolescentes, ya que la información obtenida no debe ser utilizada con fines no investigativos. Una vez obtenidos los datos, se pueden procesar para anonimizarlos, ya que en esta etapa están asociados a los adolescentes que participan. Además, se pueden procesar con alguna herramienta de inteligencia artificial en alguna etapa posterior a este proyecto.

1.3. Objetivos

Los objetivos del proyecto EDIGA, de acuerdo con el sitio [67], son:

- Obtener una radiografía, desde una perspectiva de género, de los usos que los adolescentes hacen de los entornos digitales y elaborar informes con orientaciones para los centros participantes que les permitan conocer la situación de su alumnado.
- Identificar patrones de uso en los entornos digitales asociados a conductas que pueden desencadenar situaciones de grooming, sexting, cyberbullying.
- Comprender la influencia de los entornos sociofamiliares y de los espacios de afinidad de los y las adolescentes en diferentes contextos socioculturales y económicos en la construcción de las identidades de género en cada país participante.
- Identificar, analizar y comprender las manifestaciones identitarias y prácticas socioculturales relacionadas con el género de las y los adolescentes en los entornos digitales: (a) prácticas más habituales, (b) prácticas menos habituales y (c) prácticas de rechazo.
- Identificar y analizar diferencias y similitudes en diferentes contextos de Iberoamérica (México, Uruguay y España).
- Analizar la estructura discursiva de los entornos digitales utilizados por las y los adolescentes.

Los objetivos específicos de este Proyecto de Grado son:

- Estudiar las posibilidades de extracción automática de información cuantitativa del uso de redes sociales a través de dispositivos.
- Desarrollar un prototipo de herramienta para la obtención y el manejo de los datos de uso y exposición de redes sociales de adolescentes.
- Permitir que los adolescentes participen de forma activa compartiendo la información que deseen haciendo uso de la herramienta como un diario de uso.

1.4. Resultados esperados

Se espera obtener un prototipo de una herramienta para dispositivos móviles que permita recolectar información de forma automática sobre el uso que los adolescentes realizan de las redes sociales en sus dispositivos, y que además funcione para los sujetos como un diario en el que puedan dar información de forma voluntaria. Esto se debe hacer teniendo en cuenta siempre la importancia de preservar la privacidad de los adolescentes que consienten participar en el estudio (o cuyos padres o tutores dieron su consentimiento). Se espera que la información obtenida sea de utilidad para los investigadores del proyecto EDIGA, permitiéndoles conocer mejor los hábitos de los adolescentes en el uso de sus redes.

1.5. Conclusiones

Las redes sociales establecen fuertes restricciones al acceso a la información, lo cual dificulta la obtención de datos en tiempo real. Esto llevó a que el resultado obtenido no permita

automatizar la extracción de datos de uso. Obtener una herramienta que les permita a los investigadores tener acceso a lo que hacen los usuarios en todo momento requeriría trabajo en un marco legal, no sólo técnico, y también implicaría consideraciones éticas. Se logró obtener un prototipo de una herramienta para que los adolescentes participantes aporten voluntariamente información acerca de sus interacciones en las redes sociales. En un futuro se podría realizar el trabajo en los marcos mencionados para obtener una mayor cantidad de información, se considera que lo investigado en este proyecto puede servir para ello.

1.6. Organización general de este documento

Este documento se organiza en los siguientes capítulos:

- Esta introducción, donde se presenta el panorama general del proyecto
- La revisión de antecedentes, donde se habla de aplicaciones de control parental, se presentan algunos conceptos necesarios para entender pruebas realizadas y se expone la investigación realizada sobre APIs de distintas redes sociales
- El desarrollo del proyecto, donde se explica el relevamiento de requerimientos realizado, el análisis de posibilidades técnicas y las decisiones de diseño e implementación tomadas en base a los dos puntos anteriores
- Las pruebas realizadas sobre la aplicación móvil, el servidor web y la integración entre ambas partes
- Conclusiones sobre el trabajo realizado y potenciales mejoras a futuro
- Referencias bibliográficas consultadas
- Apéndice A, con un pequeño manual de usuario
- Apéndice B, donde se describe la implementación de la aplicación móvil
- Apéndice C, donde se describe la implementación del servidor web y la base de datos
- Apéndice D, que contiene un borrador con ideas para refinar la página web para el uso de los investigadores

Capítulo 2

Revisión de antecedentes

2.1. Aplicaciones similares

Existen aplicaciones que calculan el tiempo de uso de aplicaciones en dispositivos móviles Android, por ejemplo App Usage [1]. Además, Android provee las clases para obtener el tiempo de uso de las aplicaciones: un administrador de estadísticas de uso que devuelve la información de todas las aplicaciones [2], y una clase para obtener datos de una aplicación específica [3]. También existen explicaciones del código necesario para realizar el cálculo de tiempo de uso de una aplicación en particular [7] [3]. En iOS, el propio sistema operativo le provee al usuario el tiempo de uso de las aplicaciones en pantalla, igualmente en la documentación para desarrolladores no se menciona una API que se pueda consumir para obtener esta información y darle uso [4].

Para extraer de forma automática toda la información posible sobre el uso de las redes sociales, se buscaron aplicaciones de control parental. Sin embargo, muchas de ellas no monitorean directamente el uso de las redes, sino que revisan el tiempo de uso o bloquean el uso de ciertas aplicaciones, como es el caso de KidLogger para Android [4] o Boomerang [5]. Si bien existen algunas aplicaciones como Qustodio [6] o Bark [7], estas no son de código abierto, por lo cual no es posible reusar su código. Además, esta última sólo permite obtener datos como mensajería privada en Android, no en iOS [59].

Lo que sí se logró inferir de Bark es que para poder obtener la información buscada sería necesario obtener de alguna forma el tráfico HTTP/HTTPS y analizarlo, como hace Bark que tiene su propia VPN [58]. Es por esto que se analizó entre las opciones la posibilidad de desarrollar o utilizar una VPN propia con este fin.

[1] `https://play.google.com/store/apps/details?id=com.a0soft.gphone.uninstaller`
[2] `https://developer.android.com/reference/android/app/usage/UsageStatsManager`
[3] `https://developer.android.com/reference/android/app/usage/UsageStats`
[4] `https://kidlogger.net/kidlogger-android-features.html`
[5] `https://useboomerang.com/`
[6] `https://www.qustodio.com/es/`
[7] `https://www.bark.us/?ref=W6GPBZR`

2.1.1. Conceptos previos

Sobre el tema de captura de tráfico se entrará más en detalle en la sección Alternativas para capturar tráfico. Previo a ello, sin embargo, es necesario explicar algunos conceptos que se usarán en dicha sección.

Proxy

Un servidor proxy es un intermediario entre equipos. Es común que una computadora personal se comunique con un proxy en lugar de comunicarse directamente con los sitios web deseados. El proxy recibe y redirige todo el tráfico entre ambas partes [8].

Certificado

Para poder entender el concepto de certificado digital, es necesario entender los sistemas de encriptación usados en comunicación. Para encriptar información que se envía entre dos partes, se suele usar un sistema que requiere dos claves: una privada y una pública, usando una de ellas para encriptar y la otra para desencriptar. Estas claves se generan juntas, y es computacionalmente imposible obtener la clave privada a partir de la pública.

Si se recibe información encriptada, una forma de verificar la identidad del emisor es si éste hace pública la clave de desencriptación, manteniendo privada la clave con la que se hizo la encriptación. De esta forma, cualquiera puede usar la clave de desencriptación para descifrar la información. Como dicha clave pública sólo puede descifrar información encriptada con la clave privada correspondiente, se confirma entonces que el emisor es quien dice ser, pues posee dicha clave privada. Esto se conoce también como "firma digital".

Un certificado digital, entonces, es un documento firmado por una autoridad certificadora, que contiene varios datos, entre los cuales destacan la identificación de una entidad y su clave pública. De esta forma, se puede verificar la clave pública y la identidad de su dueño [10].

Autoridad certificadora (CA)

Es una entidad que tiene la potestad de emitir certificados. Tanto el emisor como el receptor de la información deben confiar en ella. La CA confirma que la clave pública pertenece al emisor. Sólo se puede validar certificados de CAs en las cuales se confíe [10].

Certificate pinning

Es una técnica en la cual un sitio o aplicación web sólo confía en los certificados emitidos por las CA que los administradores de dicho sitio determinen. Un cliente que se conecte a un servidor que aplique esta técnica no considerará ningún certificado firmado por otra CA y lo tratará como inválido, impidiendo realizar una conexión HTTPS [63].

2.2. Recolección de datos de redes sociales

Dentro del proyecto EDIGA, se estudió el contexto social actual para saber qué redes sociales se estaban utilizando por los adolescentes en este momento. Se llegó a la conclusión

de que las aplicaciones más utilizadas son: Discord, Tik Tok e Instagram. Con esta información se estudiaron las funcionalidades que ofrecen las APIs de estas plataformas para extracción de datos.

2.2.1. API de Discord

Para poder hacer uso de la API de Discord, es necesario autenticarse. El proceso incluye la utilización de OAuth2, lo cual requiere una URI de redirección válida para obtener un código que luego se envía en una solicitud al servidor para obtener un token [18]. Se siguió un tutorial [56] y no se logró que se tomara localhost como una URI válida, por lo que no se logró obtener el token. Se intentó realizar una solicitud sin el token correspondiente y se obtuvo un error 403.

La investigación de qué información se puede obtener de la API de Discord llevó a la conclusión de que no era lo que se necesitaba. Respecto al usuario, los datos obtenidos son muy básicos [20]: nombre de usuario, mail, entre otros que no aportaban acerca del uso y las interacciones en la red. La única información de importancia que se lograría obtener es a qué "guilds" pertenece el usuario.

De acuerdo con la referencia de Discord [19], las guilds son colecciones de usuarios y canales, comúnmente llamados "servidores" en la interfaz. La referencia de la API muestra una gran cantidad de métodos que pueden ser invocados con respecto a estas "guilds", pero en general se puede observar que estos están pensados para administrar el "guild" (manejo de todos los mensajes, canales, y usuarios), y no para obtener información de un usuario en particular. Además, al investigar en profundidad se encontró que esta parte de la API está pensada para crear un "bot".

Los bots son software para realizar ciertas tareas de forma automática [47]. En el caso de Discord, se utilizan para ayudar en la administración de un "guild", pudiendo realizar tareas como enviar invitaciones, programar actividades, y muchas más [21]. Si bien esto concuerda con lo previamente observado sobre la API y sus métodos para "guilds", también resulta claro que no está pensado para dar información sobre un usuario particular. Por lo tanto, no resulta de gran utilidad para lo que se pretende de la aplicación a realizar.

2.2.2. API de Tik tok

El material oficial de Tik tok para desarrolladores [8] está hecho para embeber videos o compartir enlaces, no para obtener información de usuarios. Al investigar se encontró la existencia de varias APIs no oficiales, pero todas estas presentan dificultades para su uso. Una de ellas requiere que el desarrollador implemente una solución para firmar la URL con parámetros anti-spam [9], lo cual excede el alcance de este proyecto. Otra está desarrollada para python [10], por lo cual no puede ser utilizada directamente en un proyecto de aplicación móvil, sino que hay que usar alguna herramienta extra [31]. Hay otra que requiere pagar para ser utilizada [11], y una URI de redirección válida para OAuth2, lo cual, como se vio en la

[8] `https://developers.tiktok.com/`
[9] `https://github.com/szdc/tiktok-api`
[10] `https://github.com/davidteather/TikTok-Api`
[11] `https://tikapi.io/documentation/`

sección de la API de Discord presenta una complejidad adicional.

Además, usar APIs no oficiales puede ser riesgoso. Si Tik tok realiza cambios sobre su estructura de software, es muy probable que estas APIs dejen de funcionar [46]. Esta es una consideración muy importante, pues si se usara una de estas APIs no oficiales en la aplicación y empezara a fallar, traería dos problemas. Primero, durante un tiempo no se podría obtener información de Tik Tok, mientras se soluciona el problema. Segundo, posiblemente la API utilizada tenga cambios, lo cual a su vez implicaría cambios en la aplicación para poder seguir haciendo uso de ella.

2.2.3. API de Instagram

Esta red social tiene en realidad dos APIs: la API Graph, para cuentas de empresas o creadores y la API de visualización básica para usuarios comunes [12]. Las cuentas de las personas que usarían la aplicación no iban a ser de empresas, por lo cual aplicaba la segunda API. Esta API otorga información restringida sobre el usuario, solamente se pueden obtener algunos datos sobre los archivos multimedia que el usuario tiene publicados [26]. La información de interés para el proyecto, como cantidad de seguidores, likes, entre otros, es únicamente accesible desde la API Graph [22], la cual no aplica para este caso por lo ya mencionado. Otra alternativa considerada es la de realizar scraping directamente en la web de Instagram. Esta es una técnica que consiste en analizar el contenido HTML de una página para obtener información [45]. Sin embargo, hay un problema legal y es que el scraping viola los términos de servicio de Instagram, como se ve en el punto 10 de dichos términos [13].

Se realizaron pruebas para lograr el login OAuth con la API. Se descubrió que la API de Instagram no permite loguearse, sino que se debe usar el login con Facebook si se desea autenticarse [14]. Se implementó entonces el inicio de sesión con Facebook, aunque esto implica algunas limitaciones al proyecto: primero, es necesario tener la aplicación de Facebook en el dispositivo [24]. Segundo, dado que se pretende usar la información de Facebook para obtener una sesión de Instagram, es necesario que el adolescente vincule sus cuentas de Instagram y Facebook. Estas dos restricciones implican que los usuarios realicen ciertas acciones para participar en el proyecto, y es un riesgo porque puede no suceder, y por lo tanto no se podrían obtener los datos esperados.

Para poder obtener la información mencionada de la API, es necesario tener permiso de Instagram para que la aplicación a desarrollar le pueda solicitar los permisos relevantes al usuario (revisión de la aplicación) [25]. Este es un proceso que puede llegar a ser complejo, ya que requiere que el sistema se encuentre en un estado avanzado, pues se necesita subir la aplicación que se va a utilizar y crear un video de captura de pantalla que muestre exactamente cómo se va a hacer uso de los datos, entre otros elementos [23]. Dadas las características del proyecto, esto implicaría no sólo tener una versión casi final de la implementación de la aplicación móvil, sino también del servidor con su base de datos, pues los datos no se muestran en la aplicación sino que se almacenan allí. Recién culminado este proceso se podría empezar a realizar las pruebas con la API de Instagram, por lo que esta parte de la aplicación

[12] `https://developers.facebook.com/docs/instagram`

[13] `https://www.instagram.com/about/legal/terms/before-january-19-2013/`

[14] `https://developers.facebook.com/docs/instagram-basic-display-api`

sólo se podría completar cuando no quedara nada más que desarrollar. Como se mencionó anteriormente el uso que se le dará a los datos almacenados, no está definido en el alcance de este proyecto por lo que es una restricción para hacer la solicitud de permisos a Instagram.

2.2.4. Conclusiones

Gracias a lo observado, se puede concluir que Instagram es la única red que tiene disponible una API para consultar algunos de los datos que los usuarios suben a su cuenta. Tik tok no tiene disponible una API propia y Discord ofrece datos irrelevantes para saber qué uso le dan los adolescentes a la red social.

Capítulo 3

Relevamiento de requerimientos

El requerimiento principal es conseguir que el prototipo de la aplicación móvil obtenga la mayor cantidad de datos de uso de Instagram de los adolescentes. En el proceso de refinamiento de los requerimientos en conjunto con el equipo de EDIGA, se discutieron los problemas éticos de "espiar" a los adolescentes, pues se podría considerar la extracción automática de sus datos como una violación a su privacidad.

Además, se discutió en el proceso de refinamiento que la aplicación fuera lo más usable y simple posible sin enviar demasiadas notificaciones al usuario para que éste no pierda el interés en el uso.

Se decidió en conjunto a través de lo investigado restringir la aplicación únicamente a Android, pues se notó que para desarrollar para dispositivos Apple existían restricciones monetarias y técnicas.

Se seleccionó la red Instagram en base al estudio comparativo de la Revisión de antecedentes y a la interacción con los interlocutores que habían investigado de antemano qué redes son las más utilizadas por el grupo de estudio.

Para obtener más datos sobre las interacciones dentro de Instagram y debido a que no se puede descifrar el tráfico en caso de que se obtuviera automáticamente (ver Alternativas para capturar tráfico), se plantearon dos posibles soluciones. Una de ellas era tener un sitio web en el que los investigadores pudieran subir manualmente observaciones de las cuentas de Instagram que participan del estudio. Para realizar esta observación, los investigadores usarían "cuentas amigas", que serían cuentas de Instagram que los adolescentes deberían aceptar la solicitud de seguimiento en Instagram, pues si tienen el perfil privado no se puede visualizar la actividad de otra forma.

La otra posibilidad era que se realizaran cuestionarios al adolescente a través de una aplicación móvil durante el período de desarrollo de la investigación.

Debido a que hubo dificultades en el proceso de refinamiento de requisitos, por la interdisciplinariedad de los participantes, se planteó una encuesta al equipo de EDIGA con preguntas sobre su conocimiento de redes sociales y herramientas de trabajo informáticas. El objetivo es sondear el conocimiento y la utilidad de los requerimientos que se plantean en este proyecto para el uso de su investigación. Los resultados, que se ven en la Figura 3.1 y Figura 3.2, mostraron que, si bien la mayoría tiene Instagram, el nivel de familiaridad varía, lo cual podría

explicar la dificultad para relevar, pues al desconocer la red, se desconoce también qué datos se pueden obtener de ella. Además, se nota que se prefiere trabajar en una página web y no en una aplicación móvil para investigadores.

Partiendo de lo expuesto anteriormente y en el contexto del uso de Instagram, los requisitos principales son los siguientes.

- Obtener el país de origen del usuario
- Realizar un cuestionario al inicio
- Permitirle cada día al usuario enviar fotos
- Hacer que el usuario conteste preguntas diferentes cada día configuradas manualmente en la base de datos
- Obtener el tiempo de uso diario de Instagram
- Realizar un cuestionario a la mitad del período de uso de la aplicación
- Realizar un cuestionario al final
- Implementar una página web para uso de los investigadores

El sitio web para que los investigadores puedan subir la información que observaran desde las cuentas amigas tendría alguna forma de inicio de sesión y les permitiría subir capturas de pantalla y comentarios de éstas. Los wireframes diseñados se pueden ver en Wireframes de aplicación Web. Éste último requerimiento no se llegó a refinar por lo tanto quedó con menor prioridad que el resto.

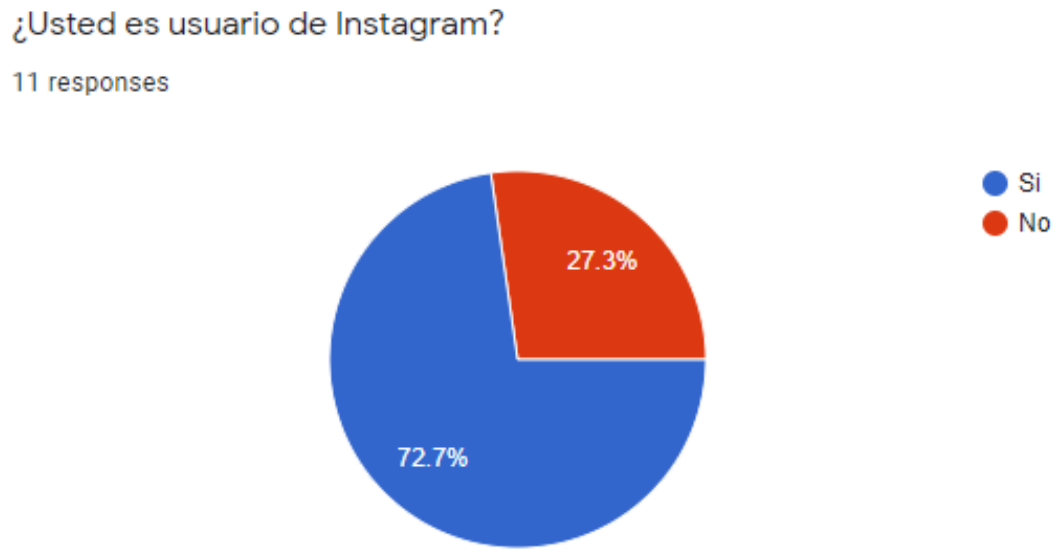

Figura 3.1: Resultado de preguntar si el equipo tiene o no Instagram

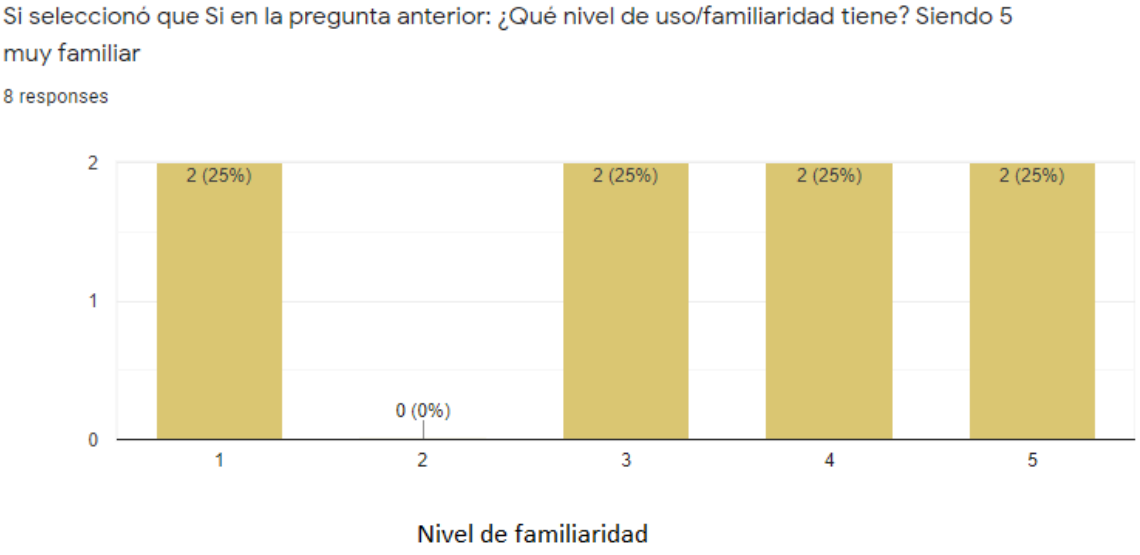

Figura 3.2: Familiaridad con Instagram

Capítulo 4

Diseño

4.1. Alternativas para capturar tráfico

Se estudió la factibilidad técnica de obtener las interacciones de los usuarios en Instagram de forma automática. Una de las formas de hacerlo, dado que no se puede acceder a lo que hacen otras aplicaciones (Android, por ejemplo, sólo permite invocar otras aplicaciones específicas como la cámara para obtener un resultado en la aplicación propia [39]), es mediante captura automática de tráfico. Dado lo mencionado sobre Bark y su VPN, la primera alternativa que se estudió fue la de tener una VPN propia, para lo cual se requería tener un servidor VPN en la nube [1], y una forma de conectarse desde el dispositivo móvil. Sin embargo, teniendo en cuenta que conceptualmente las VPN están pensadas para otorgar seguridad, más que para filtrar el tráfico, se buscaron otras alternativas.

Con este fin el equipo se reunió con el profesor Eduardo Grampín, del Departamento de Arquitectura del Instituto de Computación. Grampín planteó dos posibilidades: una era tener un servidor proxy en la nube, el cual interceptaría el tráfico del dispositivo móvil (mediante configuración previa de este dispositivo) para luego poder analizarlo donde fuera conveniente. Advirtió la necesidad de tener una buena infraestructura en la nube para poder llevarlo a cabo. La otra posibilidad planteada fue capturar el tráfico directamente en el dispositivo y reenviarlo a un servidor para su procesamiento. En este caso, la dificultad estaría en enviar y luego borrar los datos antes de saturar el dispositivo del usuario.

4.1.1. Captura en dispositivo

Android provee una clase que se debe implementar para poder capturar el tráfico, llamada VPNService [2]. Lo que ésta hace es permitir configurar una interfaz de red virtual, por la cual se puede hacer pasar (y leer) el tráfico. Se buscaron soluciones de código abierto que hicieran esta captura, para poder incorporarlas al proyecto.

Se encontró una solución que captura correctamente los paquetes [3]. El problema fue que, si bien permite el guardado de un archivo pcap (archivo que contiene la captura de tráfico [48]),

[1] `https://www.digitalocean.com/solutions/vpn/`

[2] `https://developer.android.com/reference/android/net/VpnService`

[3] `https://github.com/SummerOak/NetCloud_android`

este archivo no capturó correctamente el tráfico.

Otra alternativa que se encontró fue PCAPdroid [4], que también permite guardar archivos PCAP, pero además menciona en su guía de usuario [30] la capacidad de desencriptar tráfico HTTPS/TLS usando un cierto proxy. De acuerdo con esta documentación, usar esta utilidad en la propia aplicación requiere que el dispositivo y el servidor proxy estén en la misma red, por lo cual se descartó la opción de usarlo directamente con la aplicación. Este proxy se investigó más en profundidad, lo cual se verá en la próxima sección.

Más allá de esto, PCAPdroid tenía una importante característica, y es que permite filtrar para que sólo se tome el tráfico de una aplicación [29]. Esto permitiría tomar sólo el tráfico correspondiente a Instagram. Sin embargo, la propia documentación advierte que los archivos PCAP generados no son completamente confiables. De todas formas, se consideró que sería una buena guía sobre cómo realizar la captura de paquetes en Android.

Otra documentación sobre el tema que se encontró fue:

- Una guía oficial de Android sobre cómo manejar la clase VPNService [42]
- Algunos ejemplos de código, incluyendo una aplicación que usa este servicio para bloquear acceso a Internet por app, y que podría servir para saber cómo capturar sólo los paquetes de una app particular [5]
- Información sobre cómo Android hace esta distinción específicamente (UIDs asignados a cada app [50])

4.1.2. Proxy

Pruebas realizadas

Interceptación de tráfico y descifrado de HTTPS
Como se mencionó anteriormente, la idea de usar un proxy que además permitiera desencriptar el tráfico HTTPS resultaba conveniente, pues resolvería dos problemas con una sola herramienta. Para poder realizar las pruebas en Windows se usó Fiddler, un proxy que permite el desencriptado de tráfico HTTPS [6].

Se siguieron las instrucciones para instalar y configurar el proxy en una PC local [15], y luego se siguieron las instrucciones para conectarse desde un dispositivo Android [16], en este caso un Xiaomi Redmi 6A. Estas últimas instrucciones explican cómo se debe descargar e instalar un certificado generado por Fiddler, el cual no fue emitido por ninguna CA. Un detalle es que, si el usuario del celular no tiene configurado ningún bloqueo de pantalla (como fue el caso de la prueba), Android exige como medida de seguridad establecer algún tipo de bloqueo para poder instalar el certificado.

Se configuró el proxy en el dispositivo como explican las instrucciones y se realizó la conexión. El resultado al que se llegó no era el deseado: el celular se conectaba efectivamente al proxy levantado en el PC, y parte del tráfico HTTPS captado (por ejemplo la navegación a algunos sitios web) se podía desencriptar, pero no todo. En el caso de varias aplicaciones,

[4] `https://github.com/emanuele-f/PCAPdroid`
[5] `https://github.com/M66B/NetGuard`
[6] `https://www.telerik.com/fiddler`

incluyendo Instagram, no se veía tráfico HTTPS y además dejaban de funcionar (no cargaban nuevo contenido).

Es posible que esto se deba a que estas aplicaciones utilizan "certificate pinning". Por esto, el certificado de Fiddler no sirve para las aplicaciones que usen esta técnica, lo cual incluye Instagram. Entonces, se buscó una forma de sortear este obstáculo.

Se descubrió que Facebook implementó hace poco una configuración para testers que permite deshabilitar el "certificate pinning" tanto para Facebook como para Instagram [69] [27]. Esta configuración parecía ser lo que se necesitaba, por lo que se procedió a activarla. Sin embargo, los resultados no fueron los esperados. Si bien en el proxy se podían ver paquetes HTTPS y desencriptarlos, en la aplicación de Facebook, la mayor parte del contenido no cargaba, y en la de Instagram, al refrescar, se mostraba un error de que no se podía actualizar el feed. Por lo tanto, esta solución no funcionaba, pues impedía el uso correcto de la aplicación.

Por lo tanto, se optó por intentar encontrar soluciones que permitieran romper el "certificate pinning". De esto se hablará en la sección siguiente. Antes de pasar a ese tema, sin embargo, el equipo considera importante plantear dos cuestiones que se desprenden de las pruebas realizadas hasta el momento.

La primera es que, aún suponiendo que se logre romper el "certificate pinning" de Instagram, seguiría existiendo el problema de que otras aplicaciones que también hacen uso de esta técnica dejan de funcionar. Intentar romper el "certificate pinning" de todas las aplicaciones del usuario no es una opción viable, dado que se desconoce qué aplicaciones pueda tener el usuario, y tampoco parece muy ético ni legal romper un mecanismo de seguridad en todas las aplicaciones. Pero que no funcionen ciertas aplicaciones está fuera de la cuestión, sobre todo considerando que se pretende que el adolescente pueda usar su dispositivo con la mayor normalidad posible.

Se habló de la clase VPNService, y cómo permite implementar una interfaz de red virtual. La documentación de Google [43] menciona que se puede establecer qué aplicaciones pueden hacer uso de la interfaz virtual. El equipo consideró que, haciendo uso de esta característica, podría ser posible enviar sólo las conexiones de Instagram hacia el proxy, y que el resto de las conexiones usaran la interfaz normal. Esto permitiría que el resto de aplicaciones se conectaran con el servidor que esperan y usaran su certificado usual. Existe una aplicación, TunProxy [7], que lo que hace es configurar una conexión a un proxy y usarla sólo para ciertas aplicaciones. Dado que es de código abierto, se podía tomar dicho código y usarlo en la aplicación a desarrollar.

La segunda cuestión que se desprende de las pruebas, es la necesidad de usar un certificado autofirmado. Los navegadores web no suelen confiar en estos certificados, mostrando una advertencia al usuario cuando se intenta usar uno [60], y como se menciona en la referencia, hacer que el usuario confíe en un certificado no emitido por una CA puede hacerlo vulnerable a ataques, si confía en un servidor falso. Además, si el certificado es comprometido (es decir, un atacante roba la clave privada), no puede ser revocado, como sí ocurre con un certificado emitido por una CA.

Es cierto que, dado el tiempo reducido en el que realmente sería necesario usar el certificado, el riesgo es menor. Pero igualmente sería prudente eliminar el certificado en cuanto se

[7] `https://github.com/raise-isayan/TunProxy`

termine el estudio, para quitar todo riesgo. Como no es posible eliminarlo desde la aplicación a desarrollar [11], hay que confiar en que el adolescente lo haga cuando se le solicite, lo cual no es una garantía real.

Quebrado de "certificate pinning" y riesgos
Se realizó una búsqueda de métodos para romper el "certificate pinning" de Instagram específicamente. Un resultado encontrado fue una versión de Instagram configurada para este fin [8]. Al intentar instalar el certificado del proxy (Burp suite, no Fiddler que fue el usado anteriormente) siguiendo las instrucciones [13], se encontraron dos problemas que se repetirían en varias de las soluciones encontradas.

Uno es que se hace uso del comando "adb", correspondiente a una herramienta de Android para conectarse desde una PC a un dispositivo [35]. Esta documentación aclara que la forma de realizar la conexión puede ser mediante cable USB o WiFi, requiriendo que el dispositivo y la PC estén en la misma red. Dado que el caso de estudio requeriría conectar el dispositivo del cliente a un servidor remoto, esta no es una posibilidad. Por lo tanto, no es aplicable ninguna solución que implique esta conexión.

El otro problema es que algunos de los comandos usados para instalar el certificado en el dispositivo [13] requieren tener permiso de "root" en el dispositivo Android. Dado que Android se basa en Linux, obtener permiso de "root" significa tener permisos de superusuario, y tener acceso a los directorios del sistema operativo, además de poder hacer cambios en éste [71]. Obtener este permiso no es trivial, y por lo general involucra explotar alguna vulnerabilidad [66]. Pero además de esta dificultad, obtener permiso de "root" para usuarios que probablemente no tengan conocimiento tecnológico es peligroso. Algunos de los riesgos mencionados [71] [66] son:

- Se deshabilitan mecanismos de seguridad que vienen con el dispositivo, dejando al sistema y la información del dispositivo más vulnerable
- El dispositivo deja de buscar actualizaciones, o no se pueden instalar porque el sistema está comprometido, lo que implica dejar al dispositivo sin actualizaciones de seguridad
- Algunas aplicaciones que usan información importante, como aplicaciones de bancos, pueden realizar chequeos para asegurarse de no ejecutarse en un dispositivo con permiso de "root", por lo cual dejarían de funcionar
- Ciertos tipos de malware pueden aprovecharse de tener mayores permisos para hacer más daño
- El proceso para obtener permiso de "root", al ser complejo, podría llegar a dañar el sistema y hacer que el dispositivo deje de funcionar

Por esto, obtener permisos de "root" en el dispositivo de los participantes del proyecto no es una opción segura. Por lo tanto, cualquier método para romper el "certificate pinning" que tenga este requerimiento no es viable para el proyecto.

[8] `https://github.com/itsMoji/Instagram_SSL_Pinning`

Además de la ya mencionada, varias otras soluciones que se explican en diferentes sitios web, como haciendo modificaciones de la aplicación de Instagram [34] o tutoriales de YouTube, suelen hacer uso de "adb" o requerir permisos de "root". Incluso en muchos casos, se utiliza un emulador para probar, en lugar de un dispositivo real. Esto permite mayor libertad para probar técnicas, pues si se hace algún daño irreparable en el sistema operativo del dispositivo, simplemente se puede crear otro emulador. Esto no es una opción para este proyecto, dado que funcionará en dispositivos reales, además de que dichas técnicas no son aplicables.

Una herramienta encontrada es Frida [9], que en teoría permitiría conseguir lo requerido. Al estudiar cómo utilizarla, el primer problema encontrado es que muchos de los ejemplos y tutoriales requieren que el dispositivo tenga permisos de "root" [49], ya que hacen uso de un archivo binario que necesita dichos permisos. Esto descartó la posibilidad de usar muchos de estos ejemplos.

Se investigaron opciones que no requirieran permisos de "root". Se encontraron algunas opciones que requieren hacer uso de ciertas herramientas de frida [49] [28], como frida-trace [10], que únicamente mencionan la posibilidad de conectar un dispositivo por USB. Si se revisa la documentación de Frida para Android [33], se puede ver que hace uso de "adb" para la conexión. Como ya se vio en esta sección, esto significa que la comunicación no puede realizarse con dispositivos remotos, por lo tanto tampoco sirve.

Finalmente, se concluyó que Frida no es una alternativa viable. La única solución que parecía posible, mencionada en un tutorial [64] y que de hecho es parte de los pasos en las otras opciones ya mencionadas [49] [28], es partir del archivo apk de Instagram, es decir, el archivo al cual se compila una aplicación Android [32] (el cual se puede encontrar fácilmente en Google [11]). Utilizando una herramienta tal como Apktools [12] para obtener el código de la aplicación a partir de dicho apk, se puede inyectar el código necesario para hacer que la aplicación acepte el certificado del usuario como válido. Sin embargo, esto implicaría tener que pedirles a los participantes del proyecto que desinstalen su aplicación de Instagram, para usar la versión modificada que se haya creado. Esto puede resultar en un problema legal, ya que se estaría utilizando una versión de Instagram modificada que no es oficial y que está "espiando" al usuario.

En definitiva, no existe una solución viable al proyecto que permita descifrar el tráfico. La única posibilidad es obtener un certificado válido de Instagram, para realizar esta tarea de forma "legal". Se desconoce si sería posible contactar al equipo de Instagram, o qué clase de acuerdo legal se debería lograr.

Captura del tráfico

Más allá de poder observar el tráfico, era importante almacenarlo para procesarlo en etapas siguientes al proyecto. Como se explicó anteriormente, esto se puede hacer con archivos pcap. Sin embargo, Fiddler no permite exportar el tráfico en este formato [17], por lo que era necesaria otra herramienta. Con este propósito se utilizó la herramienta Wireshark [13].

[9] `https://frida.re/`
[10] `https://frida.re/docs/frida-trace/`
[11] `https://instagram.uptodown.com/android`
[12] `https://ibotpeaches.github.io/Apktool/`
[13] `https://www.wireshark.org/`

Si bien dicha herramienta, en forma gráfica, permite guardar archivos PCAP [62], sólo se permite guardar cuando se termina de capturar, lo cual no servía para lograr el objetivo. Sin embargo, Wireshark incluye una versión por línea de comandos, TShark, la cual tiene opciones para escribir los paquetes en un archivo a medida que se van capturando [14]. Incluso tiene opciones para separar la captura en varios archivos, moviéndose a archivos nuevos según algún criterio (por ejemplo tamaño del archivo). Esto permitiría obtener toda la información de captura y dividirla en archivos de menor tamaño.

Por otro lado, el almacenamiento de archivos pcap de varios clientes podría llegar a resultar muy costoso, y más si se considera que, al conectarse los clientes al proxy, se obtendría todo su tráfico, no sólo el de Instagram. Entonces se buscó una solución que permitiera capturar sólo el tráfico de Instagram, que es lo relevante en el contexto de este proyecto.

Wireshark permite filtrar por host http [5] [51], pero sólo para mostrar los paquetes en la aplicación, no para su captura. Al investigar, se encontró que capturar sólo los paquetes para el dominio de Instagram no es trivial [12], pues se requiere escribir algún programa en el cual ir almacenando las direcciones IP correspondientes a los hosts de Instagram [57].

No se encontró una herramienta que hiciera esto, pero sí una aplicación de Android que permite que sólo el tráfico de Instagram pase por el proxy. Dicha aplicación (TunProxy) ya fue mencionada en una anterior sección.

4.1.3. Comparación entre los dos métodos

A priori, ninguno de los dos métodos parece más ventajoso que el otro. Usar un servidor proxy requeriría que esté disponible para todos los usuarios de la aplicación, lo cual implica pagar un servicio en la nube para levantar al menos una instancia de este servidor. Además, dependiendo de la cantidad de conexiones que deba manejar el proxy, es muy probable que se necesite más de un servidor y un balanceador de carga, para asegurar que el tráfico circule con la menor demora posible. Finalmente, de implementarse esta opción, se necesitaría también agregar código en la aplicación para que sólo el tráfico de Instagram vaya hacia el proxy.

Al capturar tráfico en Android, existe el riesgo de que se llene el almacenamiento de los dispositivos con la información capturada. Para evitar esto, sería necesario asegurarse de que las capturas se envíen cada cierto tiempo a un servidor que las procese. Dependiendo de la frecuencia de estos envíos, podría llegar a ser costoso en el uso de datos del dispositivo móvil. En caso de no lograr enviar las capturas y que éstas se acumulen, el uso habitual del dispositivo podría verse afectado si su memoria se llena.

El almacenamiento ocupado en el caso de utilizar el proxy también puede llegar a ser un riesgo, habría que realizar algunas pruebas de carga para identificar qué infraestructura se necesita de forma que el servicio no se vea afectado por la falta de espacio.

Más allá de que existen estas posibilidades, la realidad es que el tráfico encriptado no es útil para el proyecto, pues no se puede acceder a la información que contiene y que es de interés. Podría utilizarse únicamente la información de en qué horario hay mayor flujo de datos.

Si se consiguiera el certificado de Instagram, entonces se podría descifrar la información. En este caso, la única alternativa sería usar el proxy, pues no es posible obtener el tráfico

[14] `https://www.wireshark.org/docs/man-pages/tshark.html`

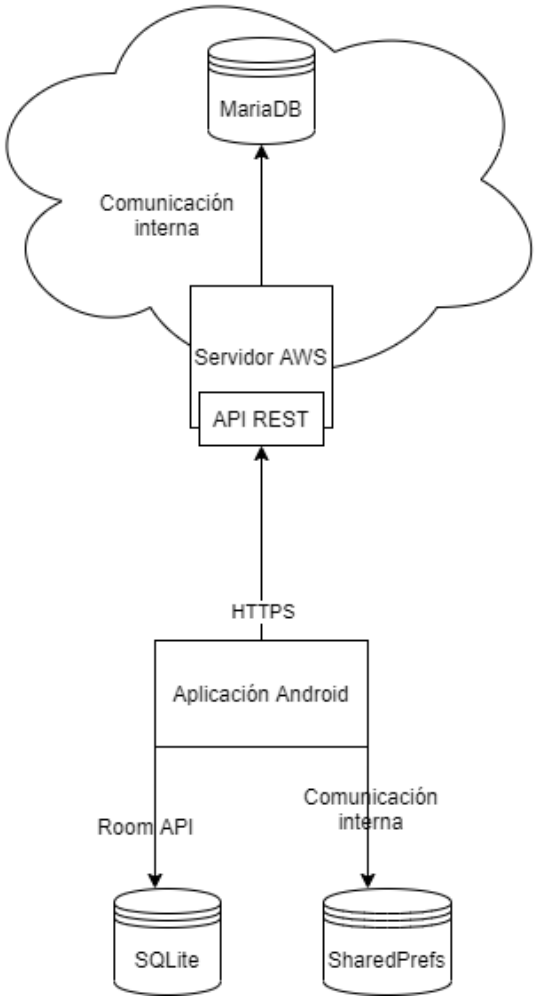

Figura 4.1: Diseño de arquitectura final

desencriptado desde Android (ya se captura cifrado, dado que no se está interviniendo en el funcionamiento de HTTPS con un certificado).

4.2. Diseño de la solución

La arquitectura que se definió fue la que se ve en la Figura 4.1.

El servidor, que está hosteado en AWS, publica una API REST y se conecta internamente con una base de datos relacional MariaDB. Tiene un certificado firmado por una CA reconocida [40] que le permite exponer la API como HTTPS, y de ese modo la aplicación móvil puede invocarla de forma segura (sabiendo que el tráfico viajará encriptado). La aplicación móvil, a su vez, se comunica internamente con las preferencias compartidas (de forma directa) y con una base SQLite (mediante la Room API, de la cual se habla en detalle más adelante).

4.2.1. Alternativas para servicios en la nube

Dado que la aplicación se va a usar desde tres países distintos, la alternativa más conveniente para el servidor y la base de datos sería colocarlos en algún servicio de nube. Se decidió investigar Azure y Amazon Web Services gracias a la disponibilidad de servicios que tienen y que están bien documentados.

Azure

Para el caso de una base SQL Server, Azure ofrece diversas opciones respecto a los distintos aspectos, como el hardware o los recursos asignados. Luego de estudiar la documentación, se logró llegar a algunas conclusiones sobre qué servicio sería mejor.

Primero, existen dos modelos para asignar y medir el consumo (modelos de compra): el modelo de núcleo virtual (vCore) o DTU [52]. De acuerdo con la documentación, el primero permite mayor personalización del hardware que el segundo, el cual viene con algunas opciones preconfiguradas. Dado el desconocimiento de Azure del equipo, y el hecho de que la demanda sobre la base no iba a ser muy grande, se consideró que DTU era más adecuado.

Dentro de DTU, existen distintos planes [53]. Se consideró que tal vez el standard sería suficiente para las necesidades del proyecto. Además, se puede contratar el servicio de base de datos única (cada base tiene recursos fijos) o grupo elástico, el cual tiene una cantidad de recursos fijos asignados para un grupo de bases, y cada base usa los recursos cuando los necesita [52]. La naturaleza del proyecto implica que probablemente no se haga un gran uso de las bases de datos en todo momento, por lo que se consideró más adecuado el grupo elástico.

Con respecto a la aplicación web, Azure ofrece App Service para publicar APIs [15]. Se consideró que el nivel básico B2 o B3 podía ser suficiente para las necesidades del proyecto. En un principio, no se pensó necesario contratar un servicio estándar ya que no sería necesario autoescalar.

Finalmente, para la comunicación entre la base de datos y el servidor, Azure ofrece una red privada [54]. Esto es importante, ya que la información que se va a guardar en la base debe estar segura no sólo en la base, sino también cuando se envía desde el servidor.

AWS

Amazon Web Services tiene varios servicios disponibles, desde buckets para mantener datos estáticos hasta servicios de inteligencia artificial. Para subir una API rest existen varias opciones según las necesidades, se notó que para subir una API ya implementada en .NET core lo más usable es una máquina virtual. También, se estudiaron los costos en la calculadora [16] y se hizo hincapié en la cantidad de documentación brindada para poder llevar a cabo la implementación. Se notó que que la consola de administración es intuitiva y fácil de usar, se pueden limitar los costos y además existe basta documentación de cada servicio prestado.

[15] `https://azure.microsoft.com/es-es/pricing/details/app-service/windows/#overview`
[16] `https://calculator.aws`

Conclusiones

Se consideró que Azure es complejo y poco intuitivo. Además, el equipo contaba con experiencia previa configurando AWS, que es lo que se decidió utilizar para este prototipo. Se utilizan entonces, los servicios gratuitos por un año de Amazon Web Services, en concreto se levantó una maquina virtual con linux [17] y un servicio de bases de datos relacionales RDS [18]. Se optó por MariaDb porque no tiene costos asociados por un año y la capacidad brindada es suficiente para presentar el prototipo de la aplicación. Una vez pasado el año de actividad gratis en AWS, se debería evaluar el uso que se hace de estos recursos para ver si sigue siendo la mejor opción o si se debería migrar a otro tipo de servidor.

4.2.2. Anonimización de datos

Este es un proceso que se realiza sobre los datos personales para evitar que puedan ser usados para identificar al dueño de éstos, pudiéndose por ejemplo eliminar o generalizar ciertos tipos de datos [14]. En el caso de esta base de datos, el dato que identifica a los participantes del estudio es el usuario de perfil de Instagram. Se decidió generar un ID de usuario Guid (identificador único global) [19] de forma que en las siguientes etapas del proyecto se pueda borrar el dato de la cuenta de Instagram para desasociarlo a los datos recopilados. El Id Guid se genera en el sistema cuando completan el formulario al inicio, y se maneja siempre a los usuarios con esos Ids generados. En la base de datos se almacena una tabla con la correspondencia entre ID de usuario y usuario de Instagram encriptado con una clave simétrica AES [20]. El motivo por el cual se almacena la cuenta de Instagram del usuario es que de lo contrario no sería posible que se cargaran datos de forma manual porque no se podrían relacionar.

Otro paso importante para la anonimización sería eliminar las fotos que se almacenen, pues se pueden usar para identificar a una persona, y almacenar descripciones en su lugar.

Las etapas pensadas para este proyecto se pueden ver en la Figura 4.2, este proyecto alcanza hasta la 3, pero a partir de la 4 se podría realizar la anonimización tal como se comentó anteriormente. Por ejemplo, se podrá hacer uso de inteligencia artificial para cambiar las imágenes por descripciones.

[17] `https://aws.amazon.com/es//?-whats-new.sort-by=item.additionalFields.postDateTime&-whats-new.sort-order=desc`

[18] `https://aws.amazon.com/es/rds/`

[19] `https://techlib.net/definition/guid.htmll`

[20] `https://es.wikipedia.org/wiki/Advanced_Encryption_Standard`

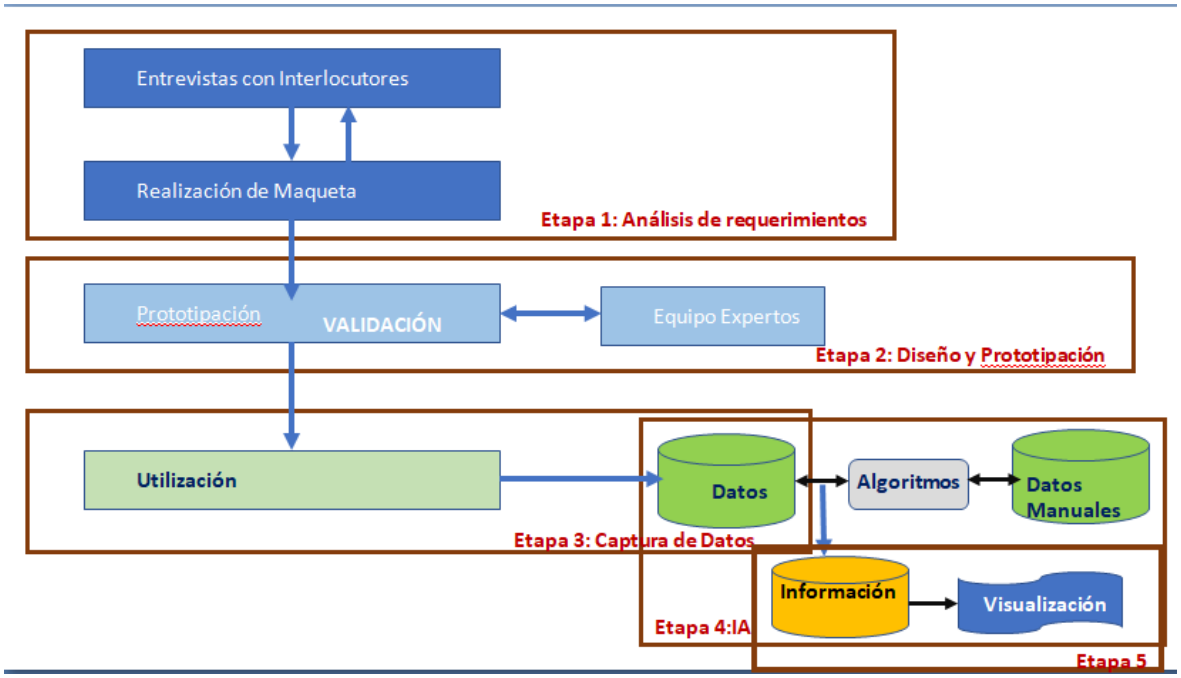

Figura 4.2: Etapas del proyecto completo de la aplicación móvil; este proyecto va hasta la etapa 3

Capítulo 5

Implementación

5.1. Tecnología usada para aplicación móvil

Para crear una aplicación móvil, existen diferentes soluciones [1]. La primera alternativa es escribir código nativo para las plataformas para las que se vaya a desarrollar (Java/Kotlin para Android, Swift para iOS).

Se investigaron frameworks que permitían desarrollar una aplicación para Android e iOs simultáneamente. Se estudió React Native, el cual implica escribir código en lenguaje JSX (similar a Javascript) que luego se compila a código nativo en cada plataforma [1], y Flutter, que usa su propio lenguaje, Dart [2].

React Native fue lanzado en 2013, mientras que Flutter se lanzó en 2018 (según [6]); como consecuencia, React Native se considera mucho más maduro que Flutter, y tiene mayor actividad en la comunidad de desarrolladores. Esto significa una mejor probabilidad de encontrar soluciones a problemas comunes, bibliotecas para distintas necesidades y mejor documentación.

Otra ventaja que menciona el artículo en [6] es la mayor facilidad para incorporar módulos nativos a React Native en comparación con Flutter, dada la forma de funcionamiento. Uno de los requerimientos planteados es obtener el tiempo de uso de las distintas aplicaciones. Esto no es realizable con un framework, por lo cual se necesitarían módulos nativos.

Dados los requerimientos del proyecto se requiere utilizar código nativo de Android, ya que iOs está fuera del alcance. Como es obligatorio tener un proyecto de Android ya creado para poder importar el módulo en React Native [44], se iba a requerir algo de conocimiento de Android nativo para el desarrollo. Bajo estas circunstancias, se optó por usar código Android nativo [9] (es decir, desarrollar Java para Android).

5.2. Diseño de la aplicación móvil

El diseño de la aplicación móvil se basa en el modelo básico MVC que implementa Android, basándose en Actividades. Si bien es cierto que esta implementación viola el principio de

[1] `https://reactnative.dev/`
[2] `https://flutter.dev/`

responsabilidad única [2], es la más sencilla de implementar.

El detalle de las clases se especifica en Apéndice B. A grandes rasgos, la aplicación se basa en una preferencia que indica el estado de la aplicación para saber qué hacer. Al abrir la aplicación por primera vez, se debe completar un formulario y luego se va a una página donde se muestran las actividades pendientes (estas son las que se reciben del servidor). Las actividades consisten en una pregunta, donde se debe escribir un comentario y/o enviar una foto. Independientemente de estas actividades, siempre se puede subir una foto con un comentario opcional. Al llegar a la mitad del tiempo del proyecto, se hará otro cuestionario, y luego un cuestionario final pasado el tiempo completo ("tiempo del proyecto" es el tiempo durante el cual se pretende que el adolescente use la app; en principio son 14 días). Cuando se completa el cuestionario final, ya no se puede volver a la sección de actividades pendientes (como sí sucede cuando se completa el cuestionario del medio).

5.3. Realización de tareas periódicas en segundo plano

Una de las principales características de la aplicación era que gran parte del trabajo realizado (recopilación de datos de uso de Instagram) sucedería en segundo plano. Para esto se investigaron diferentes alternativas. La primera encontrada, planteada en el libro de Android [9] fue usar un servicio cuya ejecución fuera planificada por una herramienta llamada AlarmManager, la cual, de acuerdo al libro, permite definir horarios para ejecuciones. Sin embargo, reconociendo que este manual no es muy reciente, y que el propio libro advierte que crear alarmas puede ser costoso para los recursos del celular, se investiga la herramienta recomendada por Google, que es WorkManager [3]. Esta permite ejecutar un trabajo de forma confiable aunque el dispositivo se reinicie o se cierre la aplicación.

De acuerdo con la documentación de Google [36], se pueden programar trabajos periódicos o de una sola vez, y si el trabajo falla se puede reintentar. Estos reintentos tienen, en caso de fallas, una política de backoff [36]. Dado que los trabajos del proyecto deben enviar información a un servidor y la conexión puede fallar, se consideró que tener el backoff ya implementado sería bueno para evitar sobrecargar al servidor en futuros reintentos.

Se hicieron pruebas con trabajos periódicos y con reintentos de trabajo que fueron exitosas. Se encontró que no funcionan si la aplicación es cerrada completamente (es decir, acceder al menú de aplicaciones recientes y eliminarla de allí). Esto, de acuerdo con [55], sucede debido al sistema operativo (las primeras pruebas fueron realizadas en un dispositivo Xiaomi), que fuerza a las aplicaciones cerradas a detener todos sus procesos, no permitiendo que el WorkManager continúe. En el dispositivo probado, fue necesario aplicar ciertas instrucciones [4] para que los trabajos periódicos funcionaran. Se probó también en un emulador de un Nexus One (dispositivo de Google), y en ese caso el trabajo periódico funcionó estando la aplicación cerrada y sin tener que realizar configuraciones adicionales. Otra prueba realizada fue en un Samsung A01, con el mismo resultado.

En los últimos celulares Samsung también existe la tendencia de no permitir la ejecu-

[3] `https://developer.android.com/topic/libraries/architecture/workmanager`

[4] `https://dontkillmyapp.com/xiaomi`

ción de trabajos en segundo plano para ahorrar batería [5]. El sitio da instrucciones sobre cómo desactivar esta prohibición en distintas versiones de Android, según el fabricante. Esta tendencia representa un riesgo para el correcto funcionamiento de la aplicación a realizar, pues necesita ejecutar trabajos en segundo plano. Además, significa requerir que el usuario realice las configuraciones pertinentes él mismo, y es posible incluso que se aborten trabajos necesarios.

Durante las pruebas de obtener el uso diario de Instagram, se detectó que se debe estar usando la aplicación para lograrlo, además de que el celular tiene que estar desbloqueado [41]. Por lo tanto, una vez por día se le solicitará al usuario que abra la aplicación para que se pueda realizar la recolección y envío de datos. Esto conlleva mostrarle diariamente al usuario una notificación en la pantalla.

La notificación es enviada con un trabajo periódico. Además del WorkManager, se exploraron otras opciones. Las alternativas manejadas fueron:

- **Enviar la notificación usando Firebase:** esto implicaba agregar trabajo del lado del servidor para enviar la notificación [6], aunque se comprobó que la notificación llegaba aún si la aplicación estaba cerrada. De todas formas, se prefirió evitar esta opción en lo posible, pues no siempre llegan las notificaciones.
- **Servicio con AlarmManager:** no se consideró esta opción, ya que todas las referencias que se encontraron sobre el tema tenían varios años de antigüedad.
- **Servicio en primer plano:** este es un tipo de servicio que corre en todo momento, aún si la aplicación está cerrada, mostrando una notificación al usuario para informarle de esto [38]. Si bien la prueba fue exitosa y el servicio se ejecutaba aún con la aplicación cerrada, estaba corriendo todo el tiempo, lo cual resulta poco eficiente considerando que el envío de datos no debería tomar más de unos minutos por día.

Luego de realizadas varias pruebas, se decidió usar el WorkManager, aunque esto implica que el usuario no pueda detener la ejecución de la aplicación.

5.4. Almacenamiento de datos internos

Se investigó SQLite, la base de datos disponible en Android [7], y se hicieron pruebas para comprobar su funcionamiento. Se consideró también usar las preferencias compartidas [8]. Estas preferencias permiten guardar pares clave-valor y que sólo la aplicación que los guarda pueda acceder a ellos. Dado que los datos que se necesitan guardar son únicos, se consideró una alternativa más sencilla a la base de datos (al seguir el capítulo sobre el tema en el libro de Android [9] se encontró que se requiere implementar varias clases para que funcione), además de ser más adecuada a las necesidades de la aplicación. Para versiones de Android cuya API sea 23 o mayor, también existe la opción de usar las preferencias compartidas

[5] `https://dontkillmyapp.com/samsung`
[6] `https://firebase.google.com/docs/cloud-messaging`
[7] `https://developer.android.com/training/data-storage/sqlite?hl=es`
[8] `https://developer.android.com/training/data-storage/shared-preferences`

encriptadas, en caso de que se requiera almacenar información sensible [37]. Esto último se usó para almacenar un identificador de usuario que se usa para identificar al dispositivo en las solicitudes que realiza al servidor.

Para el listado de preguntas diarias para que el usuario conteste, sí es necesaria la utilización de SQLite. Se encontró que existe una API llamada Room [9] que permite usar SQLite sin tener que escribir código de bajo nivel.

5.5. Servidor Web

Se desarrolló una API Rest, para cuya implementación se tomó la decisión de utilizar .NET core 5, ya que es un framework informático administrado, gratuito y de código abierto para los sistemas operativos Windows, Linux y macOS. [10]

5.6. Arquitectura de la API Rest

La arquitectura elegida para trabajar es Clean Architecture [68]. Es una arquitectura fácil de testear, que tiene la lógica de la comunicación con la base de datos independiente y además, se puede utilizar con cualquier tecnología. La arquitectura se compone de 4 capas: presentación, aplicación, dominio e infraestructura.

La capa de dominio es la que incluye parte de la lógica de negocio y las entidades, la capa de aplicación también tiene parte de la lógica de negocio. La capa de infraestructura es la encargada de generar la comunicación con la base de datos, en este caso se está utilizando Entity Framework [11], la ventaja mayor de utilizar este framework es que independientemente de las entidades y relaciones se puede aplicar a cualquier servicio de base de datos relacional, por ejemplo MySql, MariaDb, SqlServer, etc.

5.7. Autenticación en API Rest

Para garantizar que solamente la aplicación móvil hiciera pedidos al servidor, se dispuso la política de que en el header de autorización http[12] se enviara el guid del usuario registrado en base64, de esta forma solo un usuario registrado puede realizar solicitudes.

5.8. Modelo de base de datos

Dentro de la capa de Dominio se definieron las entidades como se muestra en el modelo de entidad relación en la Figura 5.1, en base a este modelo es que se construye la base de datos con las siguientes tablas.

[9] `https://developer.android.com/training/data-storage/room`
[10] `https://es.wikipedia.org/wiki/.NET_Core`
[11] `https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview`
[12] `https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Authorization`

- Activity(Id, UserId, Question_id, Answer, Original_question_date, Date_of_answer, Shared_photo)
- ActivityQuestion(Id, Question, Ask_on_day_from_project_start, Is_text_mandatory, Is_photo_mandatory)
- DailyUsage(Id, UserId, Date, Usage_time)
- EndFormAnswers(UserId, Date_of_completion, Would_upload_same_pics, Regrets_post, Got_expected_feedback_in_regretted_post, Effect_of_regretted_photo, Reason_for_not_uploading_planned_content, Content_seen_influences_self_perception, Content_published_that_defines_them_most_in_project_time, Favorite_content_published_in_project_time, Thought_carefully_about_published_content, Photos_shared_in_project_time_compared_to_normal, Care_in_content_shared_in_project_time_compared_to_normal, Uploads_more_or_less_than_in_project_time)
- MiddleFormAnswers(UserId, Date_of_completion, Favorite_uploaded_picture, Reason_for_liking, Reason_for_uploading, Defines_identity, Helps_others_see_their_real_appearance, Helps_others_see_their_desired_appearance)
- Photo(Id, UserId, Date_of_answer, Shared_photo, Photo_comment)
- User(Id, Ig_User, Country, Age)
- UserRegisterInfo(UserId, Date_of_completion, Real_name_in_network, Followers, Other_networks, Most_used_network, Weekly_ig_uploaded_content, Profile_type, Preferred_contents, Would_change_nick, Profile_pic_defines_identity, Profile_pic_helps_others_see_their_real_appearance, Profile_pic_helps_others_see_their_desired_appearance)

5.9. Implementación en AWS

Tal como se mencionó anteriormente se levantó una instancia del servicio EC2, que es una máquina virtual para la que se utilizó Linux ami 2 [13]. Dentro de la máquina se instalaron los servicios de .NET, y se siguió un tutorial para la implementación [65], dentro de los pasos a seguir se levanta la API con Supervisor [14] y se realiza el deploy. Tal como se comentó anteriormente dentro del servicio de bases de datos relacionales de Amazon, RDS, se levantó una instancia de MariaDb con 20Gb de almacenamiento que estimamos es más que suficiente para este prototipo. Por último se registró el dominio `www.edigawebapi.com` y junto con el servicio Route53[15] se configuró el ruteo por https para que la Web Api contara con mayor seguridad. Dentro de la máquina virtual se instaló el certificado utilizando Let's Encrypt[16].

[13] `https://aws.amazon.com/es/amazon-linux-2/`
[14] `http://supervisord.org/`
[15] `https://aws.amazon.com/es/route53/`
[16] `https://letsencrypt.org/`

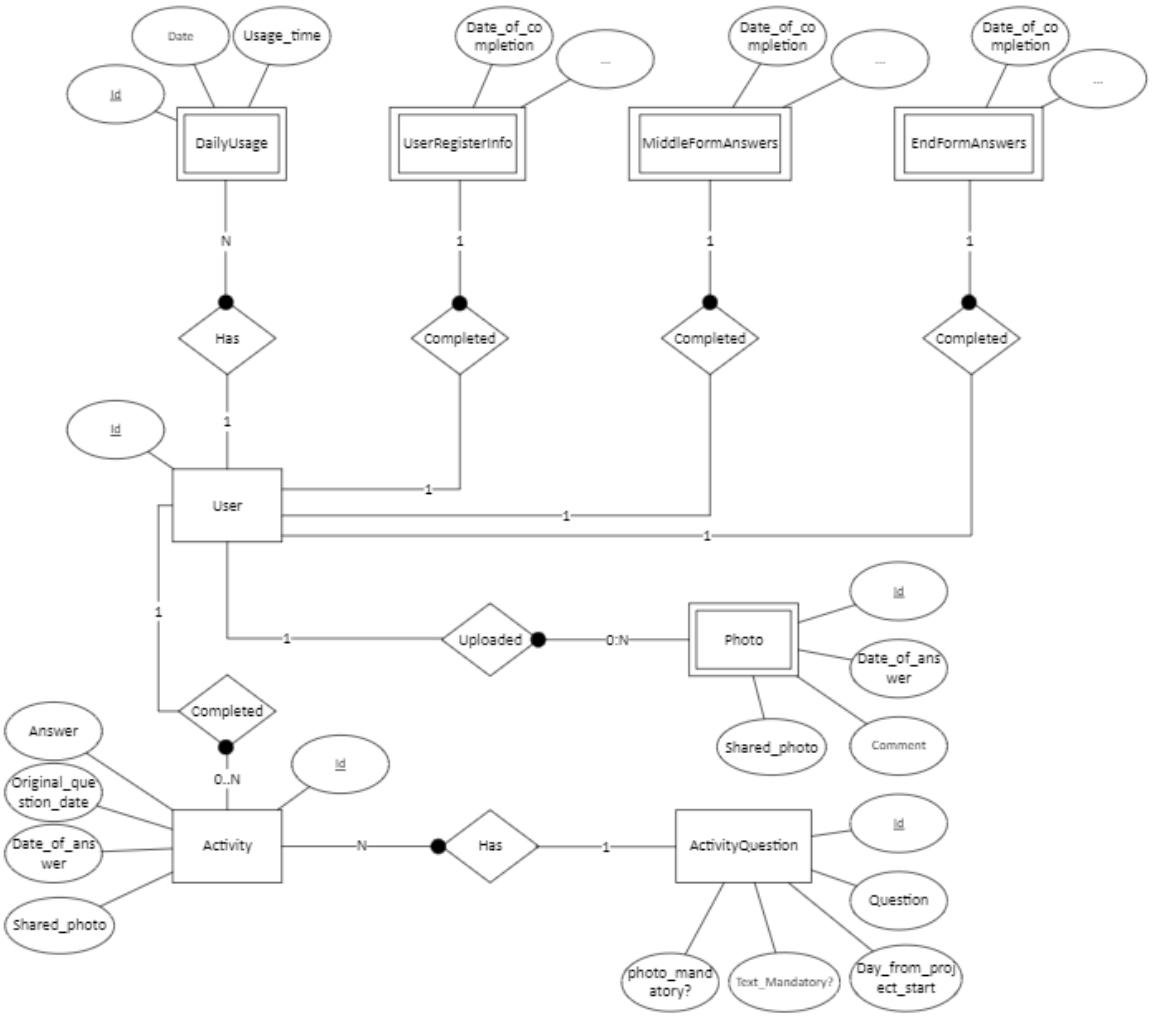

Figura 5.1: Modelo entidad relación de la base de datos

5.10. Validación del proceso de implementación

Durante el proceso de implementación se fueron encontrando errores que se fueron corrigiendo. Muchos se encontraron al implementar y probar cada parte de forma individual, pero otros se descubrieron durante la integración.
Primero, al probar un flujo completo en la aplicación (sin el servidor), es decir, registrarse, recibir notificaciones y actividades, y completar los formularios, se descubrió que, al no haberse implementado una forma de comprobar que se había completado el formulario que aparece a la mitad del tiempo del estudio, cada vez que se abría la aplicación pasado este tiempo, siempre se abría en este formulario (pues se medía que hubiera pasado el tiempo necesario, y esta condición siempre era verdadera). Fue necesario entonces agregar un mecanismo para comprobar que se hubiera completado.
Por otro lado, en la integración con el servidor se detectaron y corrigieron otros problemas. Del lado de la aplicación, se comprobó que no se había agregado el encabezado http "Authentication" (ejemplo: Device MDhkOTlhN2EtMGVjYy00NjhkLTgxMWUtM2QwYzg2N2FiMTJh) que se agregó como medida de seguridad en la solicitud de las actividades (ver Autenticación en API Rest). Además, el método que recibe el resultado no estaba preparado para el caso de que la respuesta fuera vacía. Se corrigieron estos detalles, así como que el código de respuesta de éxito podía ser del 200 al 204 (no sólo el 200). Finalmente, se corrigió un pequeño detalle que hacía que la fecha original de una actividad se enviara en un formato incorrecto, lo que resultaba en un error.
Del lado del servidor, los errores corregidos fueron mayoritariamente tipos de datos en los objetos de la solicitud que no eran los correctos. Además se agregó una URL para el caso del reintento, como se explica en Servidor web. Espera un id de usuario fijo que en la aplicación se guarda en una constante. En un futuro, esto se podría por ejemplo guardar en un archivo de configuración, sobre todo si el código se hace público, ya que cualquiera podría enviar una solicitud con ese id.

5.11. Tecnología usada para aplicación de carga manual de datos

Se investigaron distintas alternativas con el fin de que los investigadores pudieran cargar datos y observaciones manualmente.

5.11.1. Progressive Web Apps

Una aplicación nativa es una aplicación desarrollada específicamente para el sistema operativo en el que corre, mientras que las aplicaciones web corren en un navegador. Las aplicaciones web progresivas son aplicaciones que no dependen completamente del navegador, y pueden ser instaladas como aplicaciones nativas en dispositivos móviles o incluso en Windows a través del navegador [61]. Esto implicaría que, si se desarrollara la aplicación de carga de datos en esta tecnología, funcionaría tanto para dispositivos móviles como PC y se podría descargar en ambas.

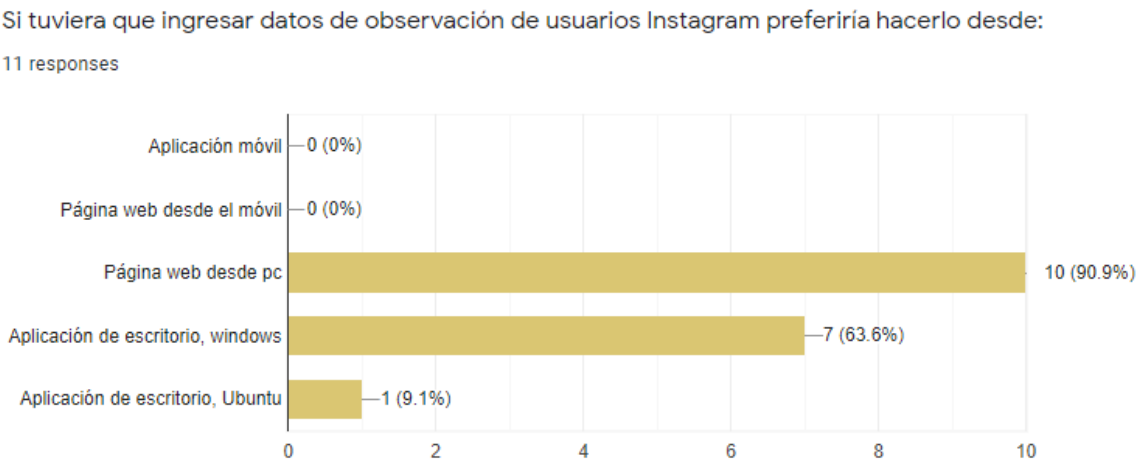

Figura 5.2: Preferencia para ingresar datos de Instagram

Las herramientas usadas para este tipo de apps son, por ejemplo, React o Angular, además de Service Workers [70]. Esta solución fue descartada por restricciones de tiempo del proyecto, ya que requería estudiar tecnologías nuevas y además no fue identificado como prioritario por el equipo de interlocutores de EDIGA.

5.11.2. Incorporación a la aplicación para adolescentes

Otra alternativa es la de la incorporación de la carga manual de datos a la aplicación móvil, usando la misma aplicación para dos tipos de usuario: adolescente e investigador. Requiriendo un login para cada tipo de usuario.

La encuesta que se mencionó en la sección de Requerimientos, mostró que los miembros del equipo EDIGA no tenían interés en usar una aplicación móvil para realizar su trabajo, como se ve en la Figura 5.2 y Figura 5.3 Por lo tanto, se descartó también esta idea.

5.11.3. Conclusión

Se llegó por lo tanto a la conclusión de que lo más adecuado es implementar un sitio web, e incluso se tuvieron en cuenta diferentes tecnologías que se podían usar para éste. Sin embargo, finalmente no se realizó la implementación de la herramienta de carga manual de datos por restricciones de tiempo y falta de refinamiento de requisitos.

5.12. Descripción de la aplicación creada

Finalmente, a grandes rasgos, la aplicación implementada almacena información del usuario, le permite completar actividades diarias (además de enviar la información de uso de

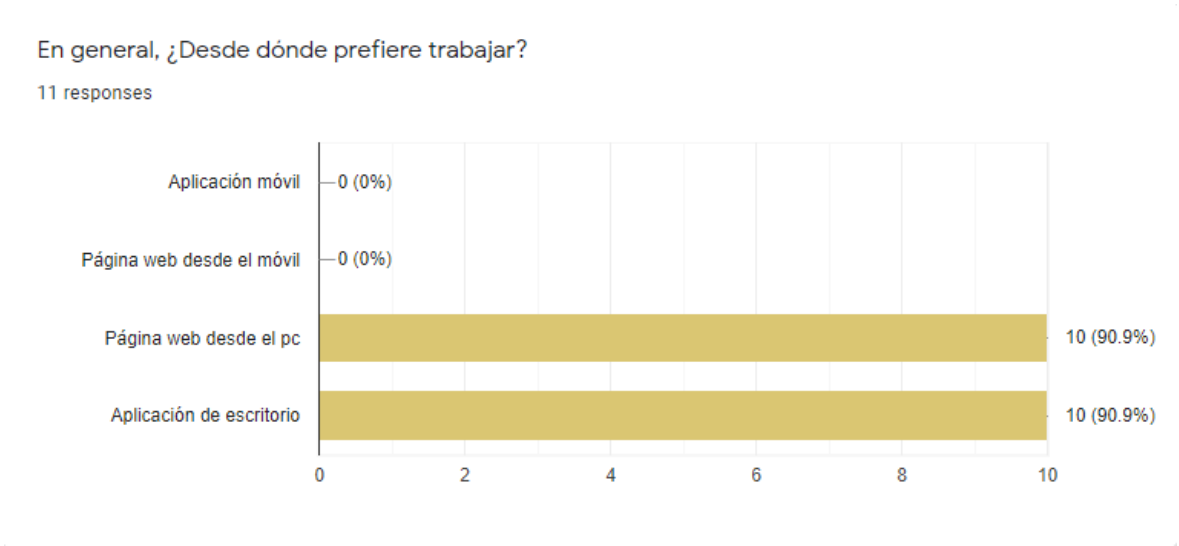

Figura 5.3: Preferencia general de herramientas

Instagram) y dos formularios más, uno al medio y otro al final del estudio de 14 días. En el prototipo, este valor no es parametrizable; para modificarlo, se debe cambiar una constante del código.

Las funcionalidades de la aplicación son las siguientes:

5.12.1. Formulario de registro

Este formulario se completa al inicio y contiene algunas preguntas sobre cómo usa el adolescente generalmente las redes. Primero se le pregunta a qué país pertenece (Figura 5.4), y luego se procede al formulario mencionado (Figura 5.5). Para el prototipo, se definieron preguntas de ejemplo que deberán ser refinadas a futuro.

5.12.2. Actividades

Cuando el usuario se registra, la aplicación obtiene del servidor un listado de actividades para que el usuario complete a lo largo de la duración del proyecto. Estas actividades consisten en una pregunta que el usuario puede responder con un comentario, una foto o ambas.
Una vez completo el registro, el usuario accederá a un listado de estas actividades que aún no haya completado (Figura 5.6). Dichas actividades tendrán un día asignado: hasta que no llegue ese día, no estarán disponibles para el usuario y no las verá en el listado. Además, una vez por día el usuario recibirá una notificación recordándole que revise sus actividades pendientes.

Si toca cualquier actividad, accederá a la pantalla que le permite completarla (Figura 5.7). El comentario y la pregunta pueden ser opcionales u obligatorios, según cómo se configuró la actividad.

En el caso de que no hayan actividades (ya sea porque el usuario las completó todas o

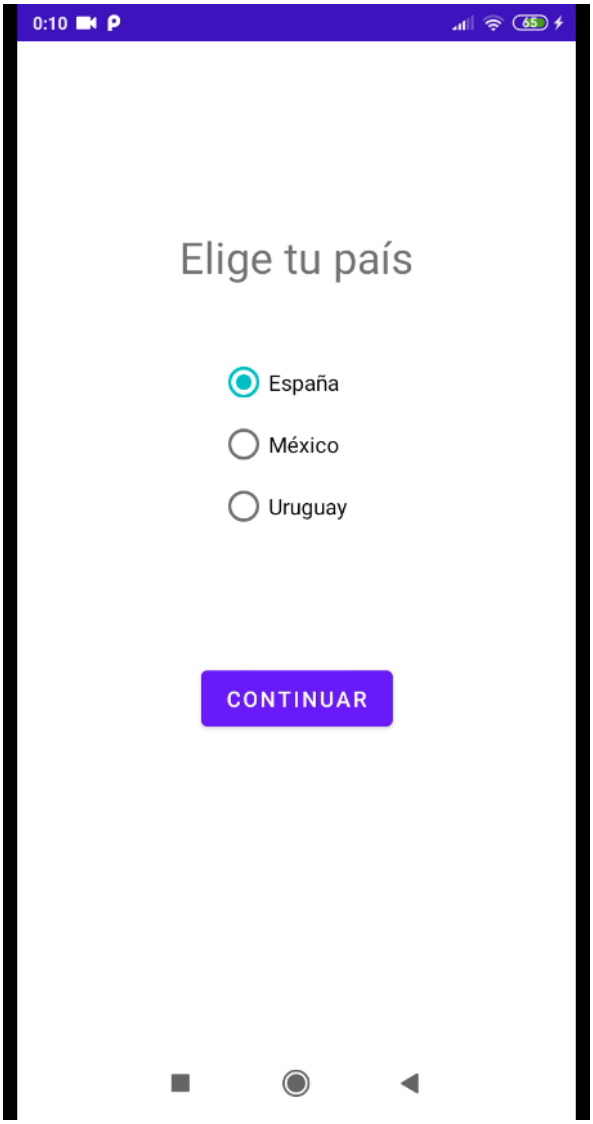

Figura 5.4: Selección de país

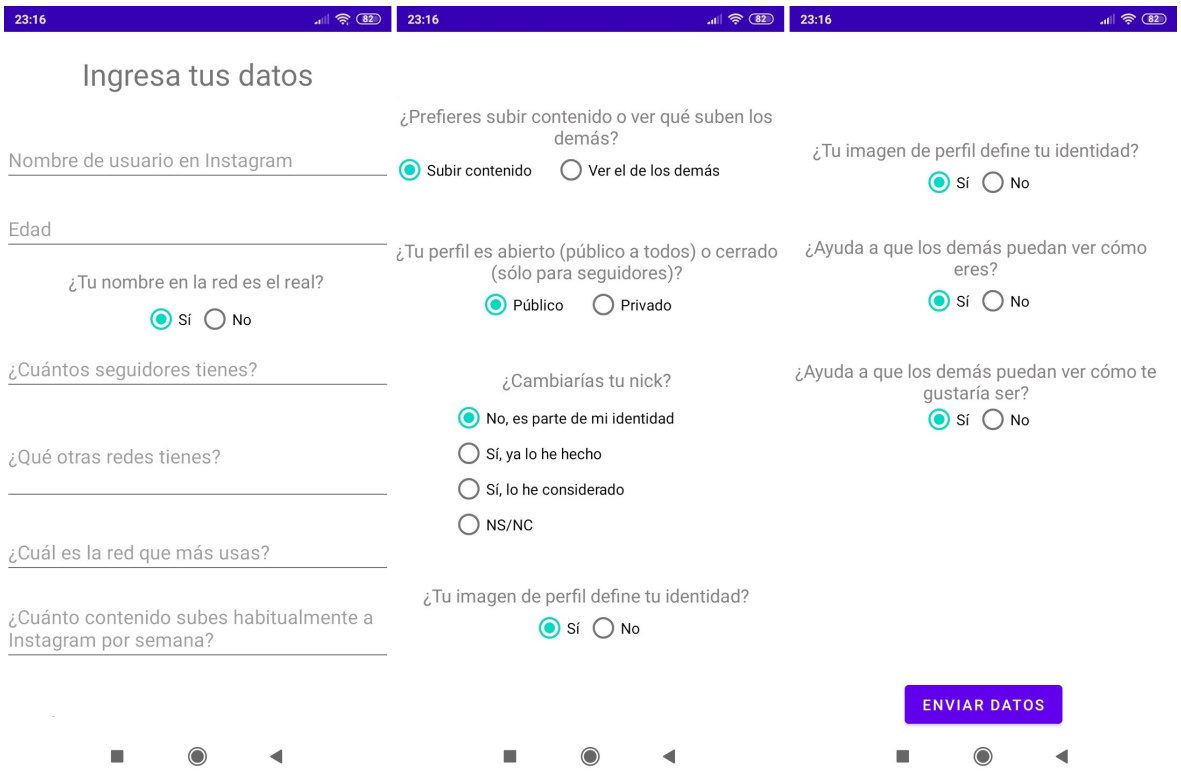

Figura 5.5: Formulario de inicio

Figura 5.6: Listado de actividades. Abajo a la derecha se encuentra el botón para subir una foto.

Figura 5.7: Actividad a completar. Más abajo se encuentra un botón para enviar los datos.

porque no se designaron actividades), se mostrará un mensaje al usuario indicando que no tiene actividades pendientes (Figura 5.8). En este caso podrá realizar el envío de foto, como se explica a continuación.

5.12.3. Envío de foto

El listado de actividades pendientes muestra también un botón que sirve para enviar una foto. Esto se puede hacer en cualquier momento independientemente de las actividades pendientes (Figura 5.9). De este modo, el adolescente puede voluntariamente compartir información si desea hacerlo.

5.12.4. Formulario del medio

Cuando haya pasado la mitad del tiempo del proyecto, en lugar de la página de actividades pendientes, el usuario será llevado a un nuevo formulario que deberá completar. Este formulario tiene preguntas diferentes al de registro (Figura 5.10). Una vez completado el formulario, el usuario será dirigido nuevamente al listado de actividades pendientes.

5.12.5. Formulario de fin

Completado el tiempo del proyecto, el usuario será redirigido a un formulario final con preguntas sobre su experiencia (Figura 5.11). Una vez completado ese formulario, culminará su actividad en la aplicación y no podrá volver al listado de actividades, aún si quedaron actividades sin completar.

5.12.6. Otras características

La primera pantalla que ve el usuario le pide que le otorgue a la aplicación los permisos necesarios: este es el permiso para acceder al uso de datos, que permitirá obtener el tiempo de uso de Instagram. El usuario tiene que acceder a una pantalla de configuración de Android para poder otorgar ese permiso. La información de uso de Instagram se envía junto con las actividades y las fotos.
En caso de que suceda un error que impida el envío de datos, la aplicación lleva al usuario a una página donde se le informa de esto (Figura 5.12). Mientras el usuario hace otras cosas, la aplicación en segundo plano reintenta el envío. Una vez logrado, cuando el usuario retome la aplicación se le redirigirá a la página correcta.

Figura 5.8: Listado de actividades vacío

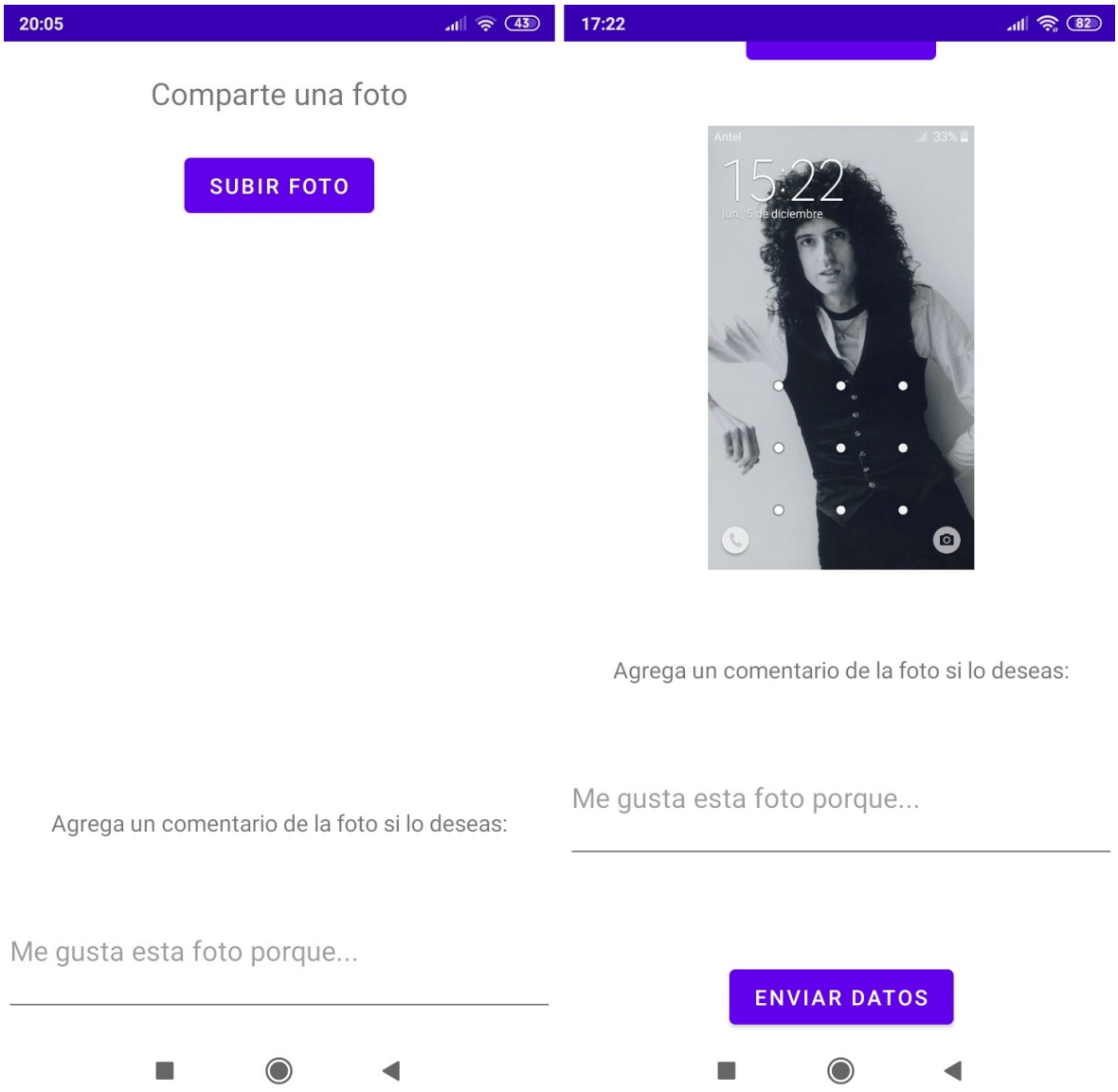

Figura 5.9: Pantalla para enviar una foto. Se puede agregar un comentario también.

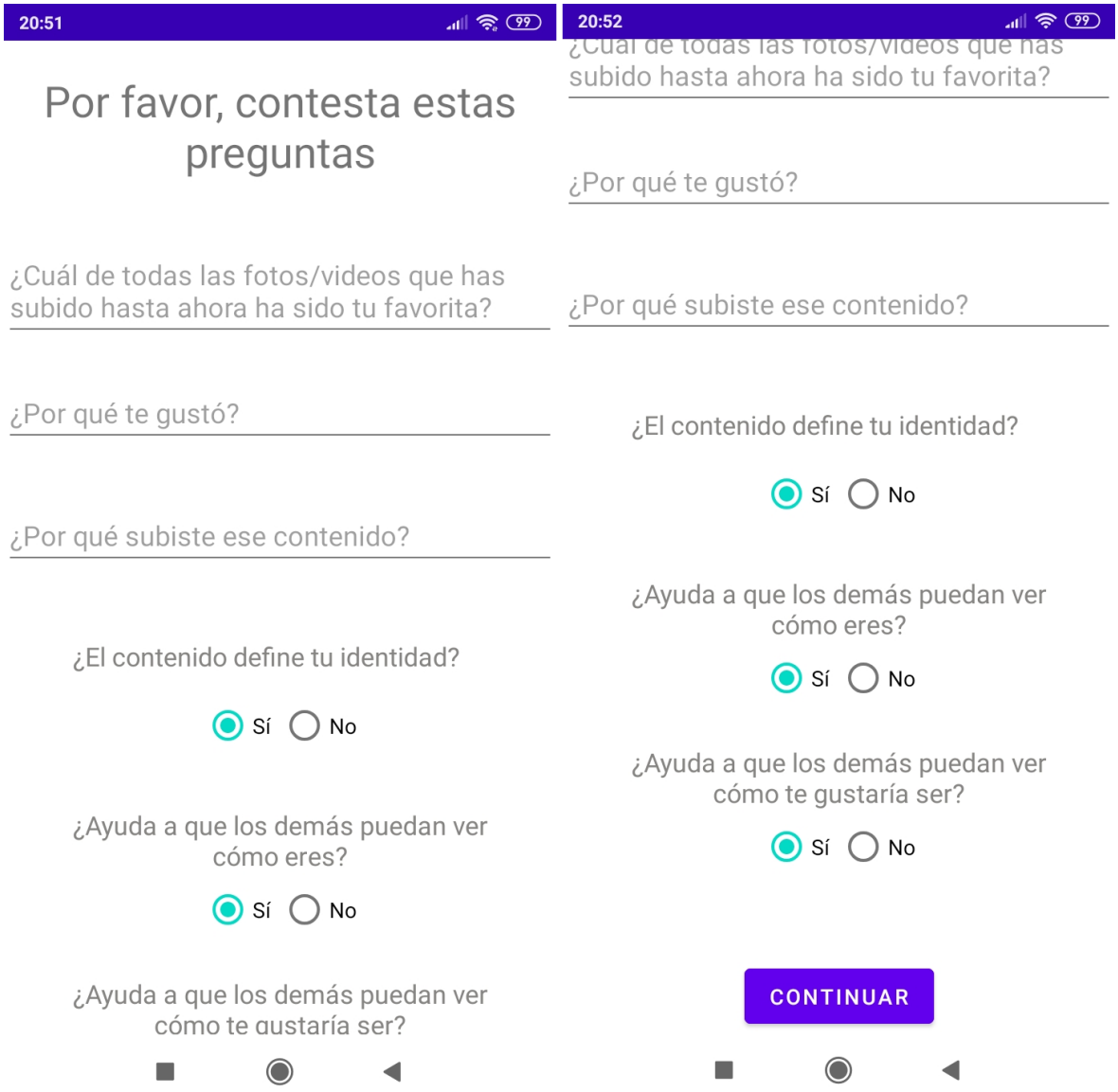

Figura 5.10: Formulario a mitad del tiempo

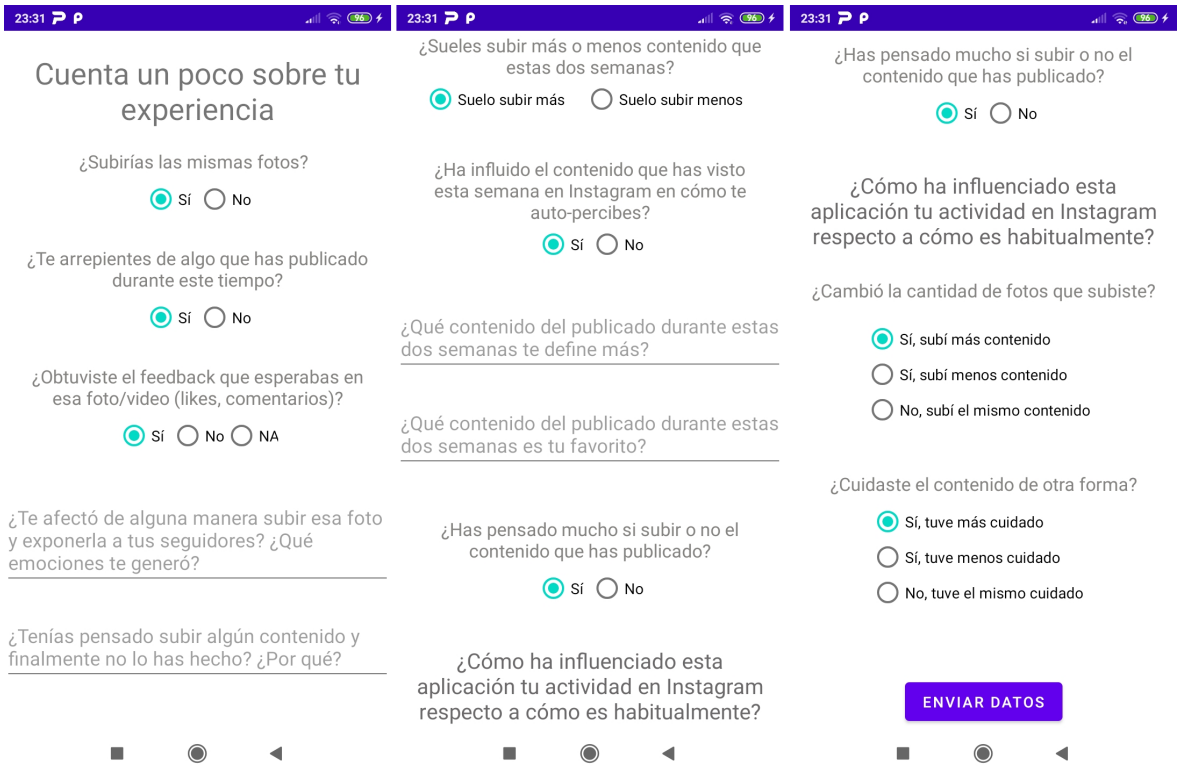

Figura 5.11: Formulario final

Figura 5.12: Pantalla de error. Se muestra si falla el envío de datos.

Capítulo 6

Experimentación

Se realizaron diferentes pruebas sobre la aplicación móvil, el servidor web y la comunicación entre ambos.

6.1. Pruebas sobre la aplicación móvil

Las pruebas sobre la aplicación móvil se realizaron con Beeceptor, una herramienta que permite hacer rápidamente una mock API [1] para alterar el resultado que se desea obtener. A no ser que se especifique lo contrario, esta API estará configurada para devolver un resultado exitoso. El objetivo es comprobar que la aplicación tiene el comportamiento esperado en todos los casos (datos sin completar, errores en el envío).

6.1.1. Envío de formulario inicial

Objetivo: realizar todos los pasos para el registro correctamente, sin fallas en la comunicación con la API.

Precondiciones: la instalación es nueva (no hay información previa almacenada).

Pasos: luego de dar el permiso de uso de datos, completar y enviar el formulario (incluyendo el paso de seleccionar el país).

Resultado: se accede a la pantalla de actividades pendientes correctamente.

6.1.2. Registro sin permiso sobre el uso de datos

Objetivo: mostrar que no se puede hacer el registro sin haber permitido el acceso a la aplicación sobre el uso de datos.

Precondiciones: no se ha abierto aún la aplicación.

Pasos: abrir la aplicación y presionar "Continuar" para avanzar.

Resultado: se ve un mensaje de error y no se puede acceder a la siguiente pantalla.

[1] `https://beeceptor.com/`

6.1.3. Registro sin completar

Objetivo: mostrar que no se permite hacer el registro sin haber completado todos los campos del formulario de registro.

Precondiciones: ya se dio permiso a la aplicación y se seleccionó el país.

Pasos: sin completar todos los campos de texto, presionar "Enviar datos".

Resultado: se ve un mensaje de error y no se puede acceder a la siguiente pantalla.

6.1.4. Fallo al enviar el formulario de registro

Objetivo: llevar a una pantalla de error cuando se intenta hacer el registro y la comunicación falla.

Precondiciones: ya se dio permiso a la aplicación y se seleccionó el país. La mock API está configurada para retornar error a la solicitud de registro.

Pasos: completar todos los campos del formulario y presionar "Enviar datos". Esperar un minuto y luego configurar la mock API para que devuelva resultado exitoso nuevamente.

Resultado: la aplicación lleva a una pantalla de error. En la mock API se pueden observar las solicitudes que llegan con la información, cada vez más espaciadas en el tiempo (recordar que se implementa una política de back-off). Luego de reconfigurada la mock API, cuando el envío se hace correctamente, la aplicación lleva a la pantalla de actividades pendientes.

6.1.5. Completar actividad pendiente

Objetivo: completar y enviar una actividad pendiente correctamente.

Precondiciones: se tienen actividades pendientes.

Pasos: seleccionar una actividad pendiente del listado, y en la pantalla de la actividad completar los datos que sean obligatorios. Presionar "Enviar datos".

Resultado: se accede nuevamente a la pantalla de actividades pendientes y la actividad completada ya no está.

6.1.6. Envío de actividad pendiente no completa

Objetivo: mostrar que no se permite enviar una actividad que no está completa.

Precondiciones: se seleccionó una actividad para completar.

Pasos: no rellenar ninguno de los campos y presionar "Enviar datos".

Resultado: se ve un mensaje de error y no se puede acceder a la siguiente pantalla.

6.1.7. Fallo de envío de actividad pendiente

Objetivo: llevar a una pantalla de error cuando se intenta completar la actividad y la comunicación falla.

Precondiciones: se seleccionó una actividad para completar. La mock API está configurada para retornar error a la solicitud de actividad.

Pasos: completar los datos de la actividad y presionar "Enviar datos".

Resultado: la aplicación lleva a una pantalla de error. En la mock API se pueden observar las solicitudes que llegan con la información, cada vez más espaciadas en el tiempo (recordar

que se implementa una política de back-off). Luego de reconfigurada la mock API, cuando el envío se hace correctamente, la aplicación lleva a la pantalla de actividades pendientes.

6.1.8. Enviar foto

Objetivo: enviar una foto correctamente.

Precondiciones: se completó el registro exitosamente.

Pasos: presionar el botón con el ícono de la foto en el listado de actividades pendientes. En la pantalla que se muestra, cargar una foto y presionar "Enviar datos".

Resultado: se accede nuevamente a la pantalla de actividades pendientes.

6.1.9. Enviar foto sin completar campos

Objetivo: mostrar que no se permite enviar una foto sin haber completado el campo de la foto.

Precondiciones: se accedió a la pantalla de envío de foto.

Pasos: no cargar ninguna foto y presionar "Enviar datos".

Resultado: se ve un mensaje de error y no se puede acceder a la siguiente pantalla.

6.1.10. Fallo de envío de foto

Objetivo: llevar a una pantalla de error cuando se intenta enviar una foto y la comunicación falla.

Precondiciones: se accedió a la pantalla de envío de foto. La mock API está configurada para retornar error a la solicitud de actividad.

Pasos: cargar una foto y presionar "Enviar datos".

Resultado: la aplicación lleva a una pantalla de error. En la mock API se pueden observar las solicitudes que llegan con la información, cada vez más espaciadas en el tiempo (recordar que se implementa una política de back-off). Luego de reconfigurada la mock API, cuando el envío se hace correctamente, la aplicación lleva a la pantalla de actividades pendientes.

6.1.11. Envío de notificaciones

Objetivo: recibir notificaciones de la aplicación correctamente.

Precondiciones: se accedió a la pantalla de tareas pendientes. La configuración de las notificaciones se cambió para que se envíen cada 15 minutos (y no 24 horas).

Pasos: esperar a que pase el tiempo tras el cual debería llegar una notificación con la aplicación en segundo plano. Tocar la notificación entrante y volver a dejar la aplicación en segundo plano.

Resultado: se recibe una notificación que indica que se pueden tener actividades pendientes. Tocar la notificación lleva a la pantalla de actividades pendientes. Cada vez que transcurre el período de tiempo especificado, llega otra notificación con el mismo texto.

6.1.12. Solicitud de de formulario a la mitad del tiempo de estudio

Objetivo: mostrar cómo se accede al formulario a la mitad del tiempo de estudio correctamente.

Precondiciones: se accedió a la pantalla de tareas pendientes en algún momento y ya pasó la mitad del tiempo de la aplicación.

Pasos: tocar la notificación recibida en el día para acceder a la aplicación. Alternativamente, abrir la aplicación.

Resultado: se accede al formulario a la mitad del tiempo de estudio en lugar de la pantalla de actividades pendientes.

6.1.13. Enviar formulario a la mitad del tiempo de estudio

Objetivo: completar y enviar el formulario a la mitad del tiempo de estudio.

Precondiciones: pasó la mitad del tiempo total de la aplicación.

Pasos: abrir la aplicación, completar los datos del formulario y presionar "Enviar datos".

Resultado: se accede nuevamente a la pantalla de actividades pendientes.

6.1.14. Enviar formulario a la mitad del tiempo de estudio sin completar campos

Objetivo: mostrar que no se permite enviar el formulario sin completar todos los campos.

Precondiciones: se accedió al formulario a la mitad del tiempo de estudio.

Pasos: sin haber completado los campos de texto, presionar "Enviar datos".

Resultado: se ve un mensaje de error y no se puede acceder a la siguiente pantalla.

6.1.15. Fallo de envío de formulario a la mitad del tiempo de estudio

Objetivo: llevar a una pantalla de error cuando se intenta enviar el formulario a la mitad del tiempo de estudio y la comunicación falla.

Precondiciones: se accedió al formulario a la mitad del tiempo de estudio. La mock API está configurada para retornar error al envío del formulario.

Pasos: completar el formulario y presionar el botón de "Enviar datos".

Resultado: la aplicación lleva a una pantalla de error. En la mock API se pueden observar las solicitudes que llegan con la información, cada vez más espaciadas en el tiempo (recordar que se implementa una política de back-off). Luego de reconfigurada la mock API, cuando el envío se hace correctamente, la aplicación lleva a la pantalla de actividades pendientes.

6.1.16. Solicitud de de formulario al final del tiempo de estudio

Objetivo: mostrar cómo se accede al formulario al final del tiempo de estudio correctamente.

Precondiciones: se accedió a la pantalla de tareas pendientes en algún momento y ya pasó el tiempo total de la aplicación.

Pasos: tocar la notificación recibida en el día para acceder a la aplicación. Alternativamente, abrir la aplicación.

Resultado: se accede al formulario al final del tiempo de estudio en lugar de la pantalla de actividades pendientes.

6.1.17. Enviar formulario al final del tiempo de estudio

Objetivo: completar y enviar el formulario al final del tiempo de estudio.
Precondiciones: pasó el tiempo total de la aplicación.
Pasos: abrir la aplicación, completar los datos del formulario y presionar "Enviar datos".
Resultado: se accede a la pantalla al final del tiempo de estudio, desde la cual ya no se puede acceder a ninguna otra pantalla.

6.1.18. Enviar formulario al final del tiempo de estudio sin completar campos

Objetivo: mostrar que no se permite enviar el formulario sin completar todos los campos.
Precondiciones: se accedió al formulario al final del tiempo de estudio.
Pasos: sin haber completado los campos de texto, presionar "Enviar datos".
Resultado: se ve un mensaje de error y no se puede acceder a la siguiente pantalla.

6.1.19. Fallo de envío de formulario al final del tiempo de estudio

Objetivo: llevar a una pantalla de error cuando se intenta enviar el formulario al final del tiempo de estudio y la comunicación falla.
Precondiciones: se accedió al formulario al final del tiempo de estudio. La mock API está configurada para retornar error al envío del formulario.
Pasos: completar el formulario y presionar el botón de "Enviar datos".
Resultado: el usuario es llevado a una pantalla de error. En la mock API se pueden observar las solicitudes que llegan con la información, cada vez más espaciadas en el tiempo (recordar que se implementa una política de back-off). Luego de reconfigurada la mock API, cuando el envío se hace correctamente, la aplicación lleva a la pantalla al final del tiempo de estudio.

6.2. Pruebas realizadas sobre el servidor web

Para testear el servidor Web se hicieron varias llamadas a cada endpoint descrito en el Apéndice C, utilizando Postman [2]. Se probaron casos borde y genéricos de llamadas exitosas para detectar errores, igualmente las pruebas de mayor profundidad se realizaron en conjunto con la integración con la Aplicación móvil.

6.3. Pruebas de integración

El objetivo es realizar pruebas pruebas integrando los dos componentes.

[2] `https://www.postman.com/`

6.3.1. Registro de usuario

Objetivo: permitir que un usuario se registre correctamente.

Precondiciones: la instalación es nueva (no hay información previa almacenada).

Pasos: completar y enviar el formulario.

Resultado: se accede a la pantalla de actividades pendientes, y los datos del usuario quedan registrados en el servidor. Además, la aplicación recibe del servidor el id del usuario.

6.3.2. Envío y recepción de listado de actividades pendientes

Objetivo: mostrar que la aplicación recibe correctamente el listado de actividades pendientes.

Precondiciones: el usuario se registró por primera vez.

Pasos: luego del registro esperar a que se cargue la siguiente pantalla.

Resultado: la aplicación recibe el listado de actividades pendientes que existe en el servidor, y muestra aquellas con fecha menor o igual a la actual.

6.3.3. Envío y recepción de listado vacío de actividades pendientes

Objetivo: mostrar que la aplicación recibe correctamente el listado vacío de actividades pendientes.

Precondiciones: el usuario se registró por primera vez, y en el servidor web no hay actividades cargadas.

Pasos: luego del registro esperar a que se cargue la siguiente pantalla.

Resultado: la aplicación recibe el listado vacío, por lo que muestra un mensaje de que no se tienen actividades pendientes.

6.3.4. Envío de actividad

Objetivo: enviar correctamente una actividad.

Precondiciones: se completó una actividad pendiente.

Pasos: presionar "Enviar datos".

Resultado: la actividad es recibida y almacenada en el servidor, junto a los datos de uso de Instagram (si aún no se enviaron datos de uso ese día) e identificada por el id del usuario que la completó, y desaparece del listado de actividades pendientes en la aplicación. Sólo hay una entrada de uso de Instagram por día en el servidor.

6.3.5. Envío de foto

Objetivo: enviar correctamente una foto.

Precondiciones: se cargó una foto con un comentario.

Pasos: presionar "Enviar datos".

Resultado: la información es recibida y almacenada en el servidor, junto a los datos de uso de Instagram (si aún no se enviaron datos de uso ese día) e identificada por el id del usuario que la completó. Sólo hay una entrada de uso de Instagram por día.

6.3.6. Envío de formulario a la mitad del estudio

Objetivo: enviar correctamente el formulario a la mitad del estudio.
Precondiciones: se completó el formulario a la mitad del estudio.
Pasos: presionar "Enviar datos".
Resultado: la información es recibida y almacenada en el servidor, identificada por el id del usuario que la completó.

6.3.7. Envío de formulario al final del estudio

Objetivo: enviar correctamente el formulario al final del estudio.
Precondiciones: se completó el formulario al final del estudio.
Pasos: presionar "Enviar datos".
Resultado: la información es recibida y almacenada en el servidor, identificada por el id del usuario que la completó.

6.3.8. Fallo en registro inicial

Objetivo: mostrar el reintento de envío del formulario inicial si el primer envío falla.
Precondiciones: se accedió al formulario inicial. El servidor está configurado para, si recibe el valor "failure" en el campo de otras redes del usuario, retornar un error.
Pasos: completar el formulario colocando el valor "failure" en el campo de otras redes del usuario y presionar "Enviar datos".
Resultado: se recibe un error y la aplicación pasa a la pantalla de error. Eventualmente la aplicación reintenta el envío, el cual sí funciona. La aplicación recibe el id de usuario y sigue su curso en el listado de actividades pendientes normalmente.

6.4. Pruebas planificadas

El objetivo de estas pruebas es realizar una carga de datos mayor a las pruebas funcionales ya realizadas, para esto se planificaron pruebas para la aplicación en las cuales participan 5 personas. El objetivo general es probar la usabilidad y el entendimiento de la aplicación, emulando el caso de uso real en el proyecto EDIGA.

6.4.1. Configuración inicial

- El tiempo del "estudio" es de 5 días
- En la base de datos hay actividades para los días 1, 2, y 4

6.4.2. Pasos iniciales en común

Pre requisitos: tener Instagram instalado en el teléfono.

1. Instalar aplicación (archivo apk)

Id	UserId	Date_of_answer	Shared_photo	Photo_comment
14	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	libélula
15	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	selfie piscina
16	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	en Instagram me gustó la receta de cata de palleja d...
17	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	prueba de subir muchas fotos
18	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	prueba de subir muchas fotos
19	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	
20	08d9b077-5543-4168-80fb-ffe2e3012b53	2021-11-27 00:00:00.000000	/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDABALDA...	prueba de subir muchas fotos

Figura 6.1: Registro en base de datos de las fotos subidas por un usuario

2. Seguir todos los pasos de configuración
3. Registrarse
4. Verificar que todos los días se recibe una notificación para entrar a la aplicación

6.4.3. Observaciones

- Se registran en una planilla las acciones realizadas durante el proceso para poder ver si la base de datos almacenó todos los cambios.
- Si se tiene dispositivo Xiaomi, seguir las instrucciones explicadas en Aplicación móvil para permitir el funcionamiento de los trabajos en segundo plano.

6.4.4. Prueba 1

Objetivo: prueba de carga
Pasos: subir muchas fotos, una tras otra (al menos 5)
Resultado: varios usuarios subieron de a 5 o 6 fotos de corrido y no tuvieron problemas de demora. Se puede ver en la Figura 6.1 los resultados de una de estas pruebas de carga, con varias fotos subidas por el mismo usuario.

6.4.5. Prueba 2

Objetivo: comprobar que es obligatorio completar el formulario a la mitad del tiempo de estudio
Pasos: no responder el formulario a la mitad del tiempo de estudio. Cerrar y volver a ingresar a la aplicación de vez en cuando.
Resultado: varios usuarios observaron que al no completar el formulario, no podían acceder a las actividades pendientes ni subir fotos nuevas.

6.4.6. Prueba 3

Objetivo: comprobar que las actividades en el listado no desaparecen hasta que no se completen.
Pasos: dejar acumular actividades sin completar y responderlas al día siguiente o al otro.
Resultado: varios usuarios dejaron actividades sin responder y observaron que cada día que pasaba iban apareciendo nuevas en la lista, estando también las anteriores presentes.

6.4.7. Resultados obtenidos

Se encontraron algunos detalles al probar la aplicación en dispositivos diferentes de los que se usaron para desarrollar la aplicación. Los formularios no se visualizaban correctamente en algunos de estos, por temas de tamaño de pantallas o de fuente. Se corrigió el problema probando con un dispositivo angosto con la letra al tamaño máximo, para asegurarse del correcto funcionamiento en el caso con menos espacio en pantalla. También se consideró que una pregunta del formulario de inicio podría tener opciones. No se realizó ningún cambio, porque son preguntas de prueba con el fin de probar la aplicación, deben ser refinadas y ajustadas según la necesidad del proyecto EDIGA.

Otro detalle encontrado fue sobre la validación de los datos del lado del servidor, esto impedía que se recibieran los datos de los formularios correctamente. Se corrigió dicha validación y se observó en la base de datos que los formularios fueron correctamente almacenados. Una sugerencia realizada por uno de los usuarios y que queda como trabajo a futuro es que las imágenes y actividades realizadas se puedan visualizar.

Capítulo 7

Conclusiones y trabajo futuro

El prototipo de aplicación que se creó, resultó más limitado que lo que se había propuesto originalmente como objetivo del proyecto. Esto se debe principalmente a dos razones. Por un lado, las dificultades para obtener información del uso de las redes fueron grandes, pues se comprobó que las redes protegen la confidencialidad de sus usuarios y no permiten obtener mucha información sobre ellos. Para obtener la información que realmente se pedía al principio de este proyecto, sería necesario alcanzar algún tipo de acuerdo, por ejemplo, con Facebook para poder visualizar esta información.

La otra dificultad fue el proceso de relevamiento de requerimientos. La falta de decisiones concretas llevó a que esta etapa se prolongara más de lo planificado, dejando al equipo con poco tiempo de desarrollo, pues dada la heterogeneidad de los interlocutores se dificultó y demoró la definición de los requerimientos. Por esto se creó una aplicación que otorgara la información que al equipo le pareció que podía ser útil para el proyecto EDIGA, en base a lo hablado y las posibilidades tanto técnicas como de tiempo. Por lo que se discutió en las reuniones, datos como el tiempo real de uso de Instagram o imágenes que el adolescente envíe de forma voluntaria son contribuciones que pueden aportar valor al proyecto.

Por estos motivos, también, es que este prototipo tiene muchas posibilidades de mejora. Se podría, por ejemplo, incorporar solicitudes a la API de Instagram, lo cual se investigó pero no se llegó a realizar. Esto se debió más que nada a la certeza de que la aplicación realizada es una primera versión que requeriría cambios para ser utilizada por el equipo de EDIGA como lo requieren, y como ya se vio, lo ideal sería tener la versión final para pedir el permiso de la API a Facebook. Sin embargo, como se planteó también, la API no da realmente mucha información, por lo que podría buscarse el acuerdo mencionado para tener un certificado que permita descifrar la información de las redes. Por supuesto, esto implicaría también tener un proxy para capturar este tráfico, y tener algún programa que analice este tráfico para obtener la información importante. Esto requeriría recursos de almacenamiento y procesamiento mucho mayores a los actuales, aunque tendría la ventaja de permitir conocer todas las interacciones que interesan al equipo de EDIGA.

Una mejora a futuro sugerida es permitir la personalización de las preguntas que se realizan en los formularios de la aplicación. Esta propuesta surge a partir de la dificultad del equipo de interlocutores para definir las preguntas a realizar en estos formularios. Actualmente

estas son estáticas, pero se podría implementar un método que obtenga preguntas del servidor, como se hace con las actividades. Esto permitiría a los interlocutores modificar los formularios en un futuro. La razón por la cual no se hizo esto es que tal nivel de dinamismo implicaría un trabajo mucho mayor en los formularios, y el equipo consideró más prioritario finalizar otras características de la aplicación y dejar un prototipo funcional. Además, el equipo considera que al utilizar una base de datos relacional, almacenar respuestas a preguntas dinámicas no es trivial, quizás en ese caso amerite utilizar algún tipo de base de datos no relacional. Lo que sí se cree que es una buena mejora es poder tener recursos para la personalización del lenguaje de cada pregunta según el país. Esto llevaría una reestructura en el código de la aplicación móvil y definiciones por parte de los interlocutores que en definitiva son quienes conocen cómo se deben hacer las preguntas en cada región.

Otra tarea a realizar a futuro es aplicar la anonimización de los datos una vez que se hayan obtenido. En el caso de este proyecto, consistiría en eliminar la columna que contiene los nombres de usuario de Instagram encriptados de la tabla de usuarios.

Con respecto a las pruebas, a futuro cuando se tenga una infraestructura fija y no provisoria para la aplicación, se deberían realizar pruebas de estrés y de carga para evaluar la eficiencia del modelo y de la infraestructura.

La página web es un requerimiento que no se pudo completar por restricciones de tiempo y que tampoco pudo ser debidamente refinado. Dada la diversidad de opiniones en el equipo EDIGA sobre la herramienta de carga manual, en el futuro se podría trabajar esta idea un poco más con ellos para refinar el requerimiento y lograr una alternativa que les sea útil para la forma de trabajo esperada.

Bibliografía

[1] Academind: React native vs flutter vs ionic vs nativescript vs pwa (2019), https://academind.com/tutorials/react-native-vs-flutter-vs-ionic-vs-nativescript-vs-pwa, accedido en abril 2021

[2] Adilovic, A.: A guide to google's recommended architecture for android apps (2021), https://www.scalablepath.com/blog/recommended-architecture-for-android-apps/, accedido en julio 2021

[3] Akhtar, W.: Using usagestats.gettotaltimeinforeground() to get the time every application in a device spent in foreground (2015), http://shorturl.at/eoHN9, accedido en agosto 2021

[4] Apple: ios screentime (2021), https://developer.apple.com/documentation/screentime, accedido en mayo 2021

[5] Aragon, J.: 5.3. saving captured packets (2015), https://osqa-ask.wireshark.org/questions/42310/catch-all-the-http-requests-to-a-certain-domain/, accedido en julio 2021

[6] Bartosz Skuza, Agnieszka Mroczkowska, D.W.: Flutter vs. react native – what to choose in 2021? (2021), https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021, accedido en abril 2021

[7] chand becse: How to get the time spent on an application in android programmatically (2019), https://stackoverflow.com/questions/32825854/how-to-get-the-time-spent-on-an-application-in-android-programmatically, accedido en agosto 2021

[8] Belcic, I.: ¿qué es un servidor proxy y cómo funciona? (2021), https://www.avast.com/es-es/c-what-is-a-proxy-server, accedido en setiembre 2021

[9] Bill Phillips, Chris Stewart, K.M.: Android Programming: the Big Nerd Ranch guide. Pearson Technology Group (2017), accedido en julio 2017

[10] Blanco, A.: Curso de fundamentos de seguridad informática - diapositivas sobre criptografía (2020)

[11] Buddy: How to delete user installed certificate programmatically? (2015), https://stackoverflow.com/questions/30327023/how-to-delete-user-installed-certificate-programmatically, accedido en junio 2021

[12] carson: Tool to track bandwidth by domain name? (2011), https://serverfault.com/questions/223106/tool-to-track-bandwidth-by-domain-name, accedido en julio 2021

[13] Compute, D.: Tech note: Installing burp certificate on android 9 (2019), https://distributedcompute.com/2019/08/15/tech-note-installing-burp-certificate-on-android-9/, accedido en junio 2021

[14] de Gobierno Electrónico y Sociedad de la Información y del Conocimiento, A.: Guía sobre anonimización de datos (2020), https://www.gub.uy/agencia-gobierno-electronico-sociedad-informacion-conocimiento/sites/agencia-gobierno-electronico-sociedad-informacion-conocimiento/files/documentos/publicaciones/Gu%C3%ADa%20sobre%20Anonimizaci%C3%B3n%20de%20Datos%20vf.pdf, accedido en mayo 2021

[15] Corporation, P.S.: Configure fiddler classic to decrypt https traffic (2021), https://docs.telerik.com/fiddler/configure-fiddler/tasks/decrypthttps, accedido en junio 2021

[16] Corporation, P.S.: Configure fiddler for android / google nexus 7 (2021), https://docs.telerik.com/fiddler/configure-fiddler/tasks/configureforandroid, accedido en junio 2021

[17] Corporation, P.S.: Export to default formats (2021), https://docs.telerik.com/fiddler/save-and-load-traffic/tasks/importexportdefault, accedido en julio 2021

[18] Discord: Discord developer portal — documentation — reference - authentication (2021), https://discord.com/developers/docs/reference#authentication, accedido en mayo 2021

[19] Discord: Discord developer portal — documentation — reference - guild object (2021), https://discord.com/developers/docs/resources/guild#guild-object, accedido en mayo 2021

[20] Discord: Discord developer portal — documentation — reference - user object (2021), https://discord.com/developers/docs/resources/user#user-object, accedido en mayo 2021

[21] Estapé, J.A.P.: Discord bots: qué son, cómo funcionan y los mejores que puedes añadir a tu servidor (2021), https://computerhoy.com/reportajes/tecnologia/discord-bots-como-instalar-mejores-bots-776589, accedido en mayo 2021

[22] Facebook: Contenido multimedia de instagram - plataforma de instagram (2021), https://developers.facebook.com/docs/instagram-api/reference/ig-media, accedido en julio 2021

[23] Facebook: Enviar para revisión (2021), https://developers.facebook.com/docs/app-review/submission-guide, accedido en julio 2021

[24] Facebook: Inicio de sesión con facebook para android: inicio rápido (2021), https://developers.facebook.com/docs/facebook-login/android, accedido en julio 2021

[25] Facebook: Permissions - plataforma de instagram (2021), https://developers.facebook.com/docs/instagram-basic-display-api/overview/permissions#instagram-graph-user-profile, accedido en julio 2021

[26] Facebook: Referencia - plataforma de instagram (2021), https://developers.facebook.com/docs/instagram-basic-display-api/reference, accedido en julio 2021

[27] Facebook: Security testing for mobile apps made easy (2021), https://www.facebook.com/notes/977148692766914/, accedido en junio 2021

[28] Fadeev, A.: Frida's gadget injection on android: No root, 2 methods (2020), https://fadeevab.com/frida-gadget-injection-on-android-no-root-2-methods/, accedido en julio 2021

[29] Faranda, E.: 1.3 filters - pcapdroid (2021), https://emanuele-f.github.io/PCAPdroid/quick_start#13-app-filter, accedido en mayo 2021

[30] Faranda, E.: 3.1 decrypt https/tls (2021), https://emanuele-f.github.io/PCAPdroid/tls_decryption, accedido en mayo 2021

[31] Farooq, U.: Tools to run python on android (2018), https://medium.com/@umerfarooq_26378/tools-to-run-python-on-android-9060663972b4, accedido en setiembre 2021

[32] FileInfo: .apk file extension (2021), https://fileinfo.com/extension/apk, accedido en julio 2021

[33] Frida: Android - frida, https://frida.re/docs/android/, accedido en julio 2021

[34] Genovese, M.: How to bypass instagram ssl pinning on android (v78) (2019), https://plainsec.org/how-to-bypass-instagram-ssl-pinning-on-android-v78/, accedido en julio 2021

[35] Google: Android debug bridge (adb) (2021), https://developer.android.com/studio/command-line/adb?hl=es-419, accedido en junio 2021

[36] Google: Define work requests (2021), https://developer.android.com/topic/libraries/architecture/workmanager/how-to/define-work, accedido en agosto 2021

[37] Google: Edit shared preferences (2021), https://developer.android.com/topic/security/data#edit-shared-preferences, accedido en agosto 2021

[38] Google: Foreground services (2021), https://developer.android.com/guide/components/foreground-services, accedido en agosto 2021

[39] Google: Getting a result from an activity (2021), https://developer.android.com/training/basics/intents/result, accedido en junio 2021

[40] Google: Proteger sitios con el protocolo https (2021), https://developers.google.com/search/docs/advanced/security/https?hl=es, accedido en octubre 2021

[41] Google: Usagestatsmanager - queryeventstats (2021), https://developer.android.com/reference/android/app/usage/UsageStatsManager#queryEventStats(int,%20long,%20long), accedido en agosto 2021

[42] Google: Vpn - android developers (2021), https://developer.android.com/guide/topics/connectivity/vpn, accedido en mayo 2021

[43] Google: Vpn - per-app vpn (2021), https://developer.android.com/guide/topics/connectivity/vpn#per-app, accedido en junio 2021

[44] Inc., F.: Android native modules (2021), https://reactnative.dev/docs/native-modules-android, accedido en junio 2021

[45] Ionos: Web scraping: ¿qué es y para qué se utiliza esta técnica? (2020), https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-el-web-scraping/, accedido en julio 2021

[46] Irani, R.: The curious case of the unofficial apis (2011), https://www.programmableweb.com/news/curious-case-unofficial-apis/2011/11/15, accedido en junio 2021

[47] Kaspersky: What are bots? – definition and explanation (2021), https://www.kaspersky.com/resource-center/definitions/what-are-bots, accedido en junio 2021

[48] Keary, T.: Pcap: Packet capture, what it is and what you need to know (2021), https://www.comparitech.com/net-admin/pcap-guide/, accedido en junio 2021

[49] Kozyrakis, J.: Using frida on android without root (2017), https://koz.io/using-frida-on-android-without-root/, accedido en julio 2021

[50] Latif, I.: How to view network traffic requested by a specific app? (2021), https://android.stackexchange.com/questions/203868/how-to-view-network-traffic-requested-by-a-specific-app, accedido en junio 2021

[51] Lekensteyn: How do i get wireshark to filter for a specific web host? (2014), https://superuser.com/questions/840656/how-do-i-get-wireshark-to-filter-for-a-specific-web-host, accedido en julio 2021

[52] Microsoft: Elección entre los modelos de compra de núcleo virtual y dtu: Azure sql database y sql managed instance (2021), https://docs.microsoft.com/es-es/azure/azure-sql/database/purchasing-models, accedido en agosto 2021

[53] Microsoft: Niveles de servicio en el modelo de compra basado en dtu (2021), https://docs.microsoft.com/es-es/azure/azure-sql/database/service-tiers-dtu, accedido en agosto 2021

[54] Microsoft: Web app private connectivity to azure sql database (2021), https://docs.microsoft.com/en-us/azure/architecture/example-scenario/private-web-app/private-web-app, accedido en agosto 2021

[55] mikepenz: Android. is workmanager running when app is closed? (2018), https://stackoverflow.com/questions/50682061/android-is-workmanager-running-when-app-is-closed, accedido en agosto 2021

[56] Nylander, J.: Oauth: how to test with local urls? (2020), https://stackoverflow.com/questions/10456174/oauth-how-to-test-with-local-urls, accedido en mayo 2021

[57] packetie: Capture packet dump from a specific domain (2015), https://stackoverflow.com/questions/30049186/capture-packet-dump-from-a-specific-domain, accedido en julio 2021

[58] Paola: Troubleshooting screen time and web filtering – bark (2021), https://support.bark.us/hc/en-us/articles/360050415891-Troubleshooting-Screen-Time-Web-Filtering#h_01EPW8NTH3EEBQ0N63Q2VRA3BD, accedido en junio 2021

[59] Paola: What bark monitors on different platforms – bark (2021), https://support.bark.us/hc/en-us/articles/360049955772-What-Bark-Monitors-on-Different-Platforms#h_01EP5EP37Z6F06500T59FHF0NW, accedido en junio 2021

[60] Pornin, T.: What are the risks of self signing a certificate for ssl (2011), https://security.stackexchange.com/questions/8110/what-are-the-risks-of-self-signing-a-certificate-for-ssl, accedido en junio 2021

[61] Ramírez, I.: ¿qué es una aplicación web progresiva o pwa? (2018), https://www.xataka.com/basics/que-es-una-aplicacion-web-progresiva-o-pwa, accedido en octubre 2021

[62] Richard Sharpe, Ed Warnicke, U.L.: 5.3. saving captured packets (2021), https://www.wireshark.org/docs/wsug_html_chunked/ChIOSaveSection.html, accedido en julio 2021

[63] Rowley, J.: Stop certificate pinning (2020), https://www.digicert.com/blog/certificate-pinning-what-is-certificate-pinning, accedido en junio 2021

[64] Sapariya, N.: How to bypass ssl pinning in android (2017), https://nileshsapariya.blogspot.com/2017/02/how-to-bypass-ssl-pinning.html, accedido en julio 2021

[65] servicestack.net: Deploying .net core apps to amazon linux 2 ami (2021), https://docs.servicestack.net/deploy-netcore-to-amazon-linux-2-ami, accedido en octubre 2021

[66] Snyder, J.: What are the security risks of rooting your smartphone? (2021), https://insights.samsung.com/2021/04/15/what-are-the-security-risks-of-rooting-your-smartphone-2/, accedido en julio 2021

[67] Stellae: Ediga - entornos digitales e identidad de género en la adolescencia (2021), http://stellae.usc.es/ediga/, accedido en setiembre 2021

[68] Taylor, J.: Clean architecture (2020), https://jasontaylor.dev/clean-architecture-getting-started/, accedido en octubre 2021

[69] TechMonitor: Facebook rolls out special feature for whitehats (2021), https://techmonitor.ai/techonology/cybersecurity/facebook-whitehat-settings, accedido en junio 2021

[70] Warren, J.: Pwa - pros, cons, cost and technologies (2020), http://shorturl.at/fvHSW, accedido en octubre 2021

[71] Whitwam, R.: Why you should (or shouldn't) root your android device (2021), http://shorturl.at/zI679, accedido en julio 2021

[72] Wikipedia: Iso_3166-2 (2020), https://es.wikipedia.org/wiki/ISO_3166-2, accedido en octubre 2021

Apéndice A

Manual de usuario

¡Gracias por ser parte del proyecto EDIGA! Una vez que obtengas la aplicación, puedes instalarla. Cuando la abras por primera vez, te encontrarás con el cartel de la Figura A.1.

Aquí tienes que darle permisos a la aplicación para que pueda acceder a tu tiempo de uso de Instagram. Para ello pulsa el botón "Dar permisos". Se te llevará a una pantalla que listará tus aplicaciones, como se ve en la Figura A.2. Busca "Proyecto Ediga" y púlsala para poder acceder a la pantalla para otorgar permisos.

Una vez hagas esto, vuelve atrás hasta llegar nuevamente a la aplicación y presiona "Continuar". Completa los datos que se te soliciten. Cuando los hayas enviado, se mostrará una pantalla que mostrará tus actividades pendientes, si es que tienes. Las actividades corresponden a cierto día y consisten en que contestes una pregunta o envíes una foto. Puede ser necesario que hagas ambas cosas.

Más allá de las actividades, si así lo deseas, en cualquier momento puedes compartir una foto y opcionalmente un comentario, presionando el botón que aparece abajo a la derecha en el listado.

A la mitad del tiempo del proyecto, verás al abrir la aplicación que se te pedirá que contestes ciertos datos. Una vez que lo hagas, volverás a ver la pantalla de siempre. Al final del tiempo, tendrás que contestar un último formulario, tras el cual podrás desinstalar la aplicación.

A.1. IMPORTANTE: configuración de trabajos en segundo plano

Esta aplicación envía notificaciones para recordarte que revises tus actividades (si las abres, la aplicación te llevará a estas actividades). En dispositivos Xiaomi, para que se envíen estas notificaciones deberás hacer una configuración especial. Típicamente, dicha configuración consiste en acceder a Configuración, luego Aplicaciones, luego Administrar aplicaciones y allí seleccionar Proyecto Ediga. Dentro de la configuración de la aplicación, accede a "Ahorrador de batería" y selecciona "Sin restricciones". Esto se ve en la Figura A.3.

Además de esto, para que funcione la aplicación como corresponde, no podrás cerrarla completamente en ningún momento, pues esto hace que dejen de funcionar los trabajos en

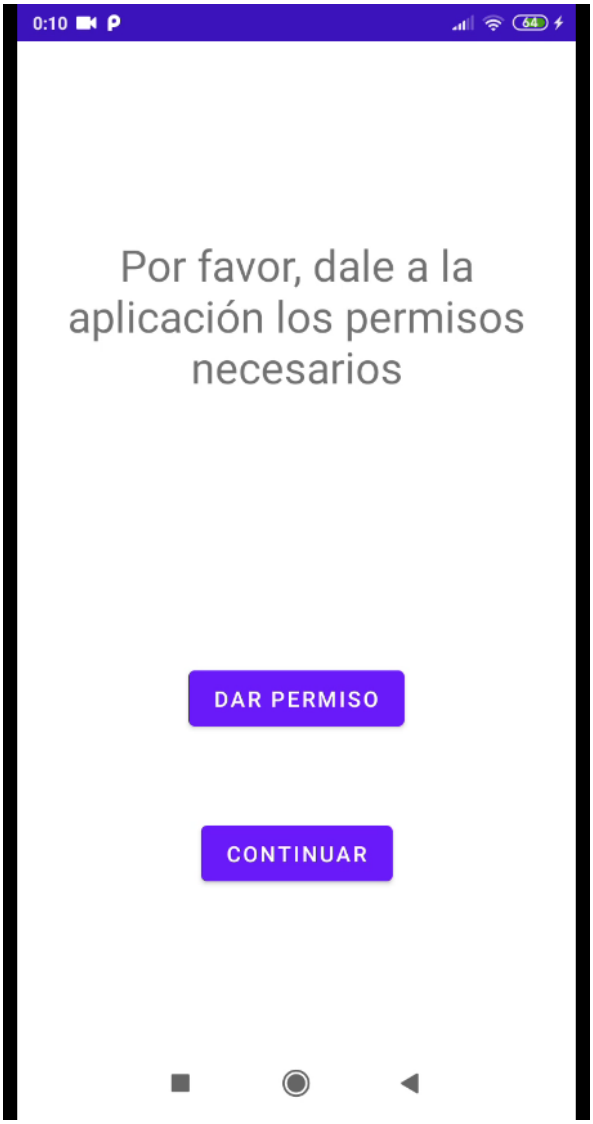

Figura A.1: Primera pantalla de la aplicación

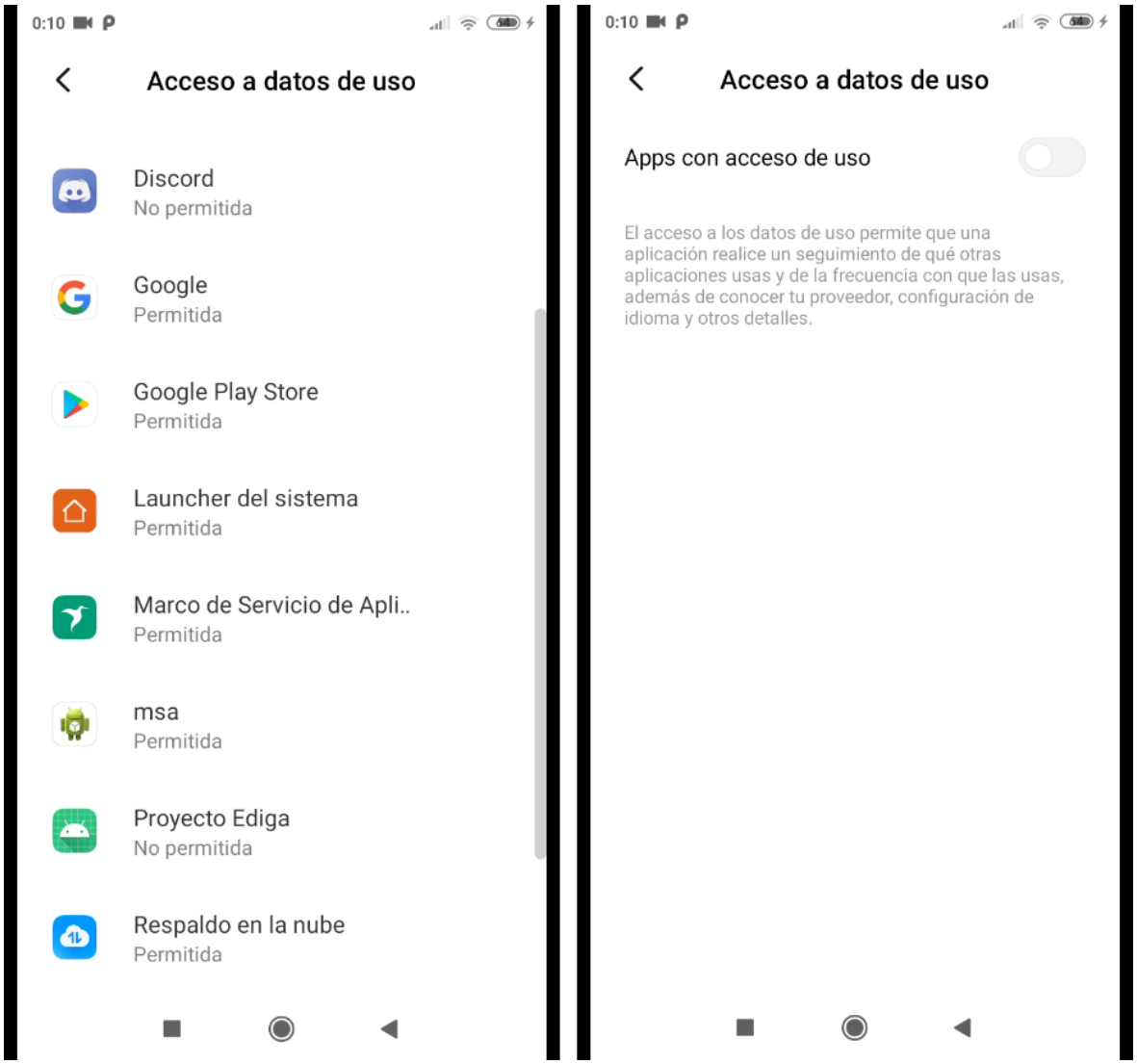

Figura A.2: Listado de aplicaciones. Al seleccionar Proyecto Ediga accederás a la pantalla de la derecha, donde debes tocar el interruptor para activar los permisos.

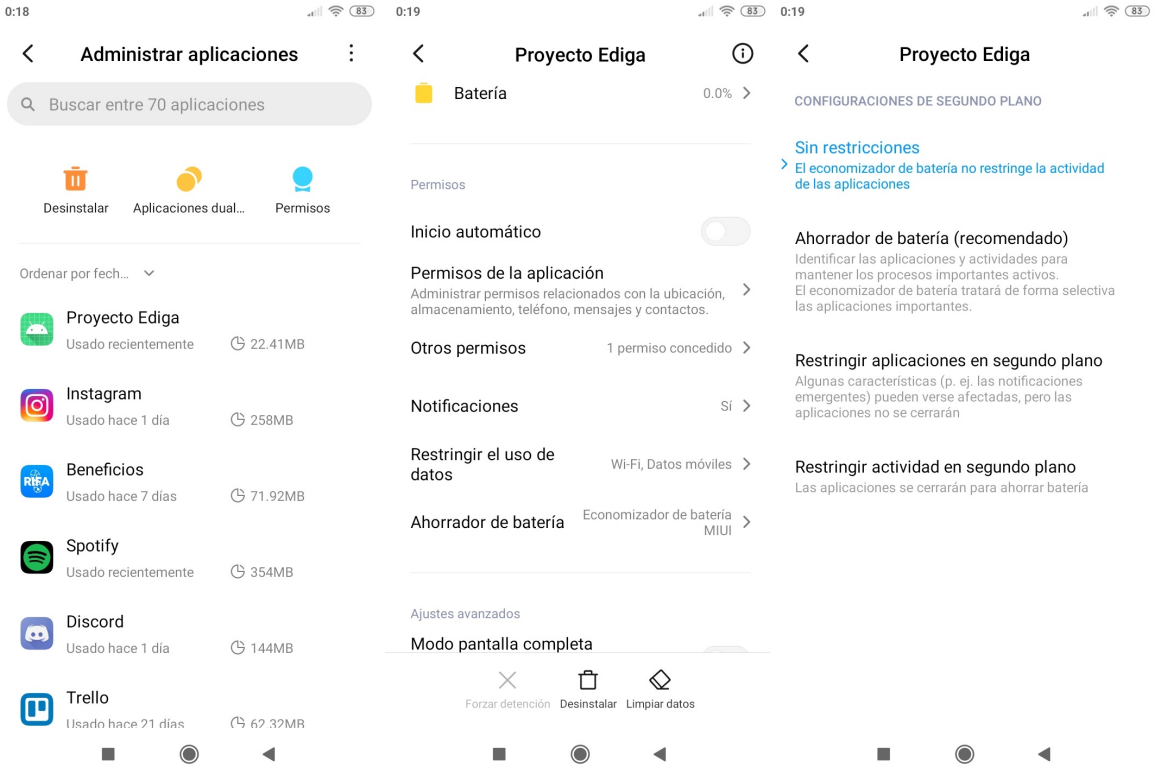

Figura A.3: En el listado de aplicaciones a administrar, al entrar a esta aplicación, debes acceder a “Ahorrador de batería” y seleccionar “Sin restricciones”.

segundo plano de los que depende la aplicación. Si falla algún envío de datos, también se usarán estos trabajos, por lo que en caso de falla es aún más importante que no cierres la aplicación (como se menciona en la Figura A.4).

Si la configuración de tu dispositivo Xiaomi se ve diferente a ésta, o si tu dispositivo no es Xiaomi pero has notado que no te funcionan las notificaciones (pues es posible que estos problemas sucedan también en dispositivos Samsung, por ejemplo), dirígete a `https://dontkillmyapp.com/` y busca la marca de tu dispositivo para obtener instrucciones específicas del sistema operativo que tengas.

¡Que disfrutes!

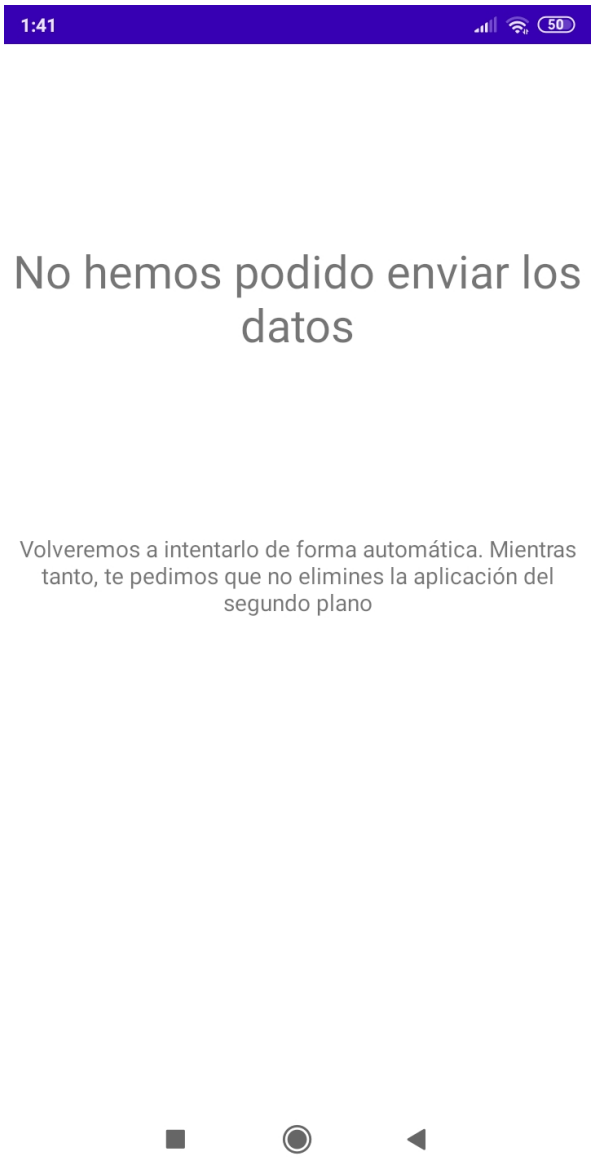

Figura A.4: Si ves este cartel, es que el envío ha fallado. No cierres la aplicación; déjala en segundo plano mientras haces otras cosas. Eventualmente esta pantalla cambiará cuando el envío tenga éxito.

Apéndice B

Aplicación móvil

La aplicación se maneja con estados que permiten definir a dónde se debe llevar al usuario. Estos estados están definidos en un enumerado. Hay varias actividades, cada una con su correspondiente archivo de layout (el cual define qué es lo que se muestra en la pantalla), como Android Studio las genera. La actividad principal lee el estado actual de la aplicación (almacenado en una preferencia compartida) y define a qué actividad llevar.

El otro componente principal de la aplicación son los workers. Ya se mencionó que se usa WorkManager; la forma en que funciona es creando clases que hereden de Worker, que hacen el trabajo. En esta aplicación se usan para reintentar el envío de datos, a excepción de un worker periódico que se encarga de enviar las notificaciones.

También existen algunas otras clases, tales como enumerados para definir algunas respuestas de los formularios, y clases estáticas que devuelven los nombres de las propiedades que hay que enviar a los endpoints. A continuación se describen las clases existentes en el proyecto, separadas por las carpetas en que se encuentran.

B.1. Activities

Contiene las actividades. Hay dos que no se usan en el proyecto: GetFBTokenActivity y TermsAndConditionsActivity. La primera es la que tiene el botón de login con Facebook y se usaría para obtener el token de Facebook para poder usar la API de Instagram. La segunda debería mostrar los términos y condiciones para usar la aplicación. Ambas actividades se dejaron en el proyecto para su uso en una futura versión de la aplicación.

B.1.1. MainActivity

Es llamada cada vez que se abre la aplicación de cero. Obtiene la preferencia compartida del estado de la aplicación y define a qué actividad llamar en base a dicho estado.

B.1.2. BasePostRequestActivity

Su función principal es proveer funcionalidades para que hereden todas aquellas actividades que tengan que enviar datos al servidor (no tiene un layout asociado, pues no muestra

nada). Su método principal es el que realiza la solicitud, obteniendo el objeto JSON a enviar y la URL de sus hijos. También agrega el id de usuario a las solicitudes, desde la preferencia compartida encriptada (en el caso del formulario de inicio, donde no existe este id aún, se envía vacío). Si la solicitud falla, programa el worker para reintentar el envío, obteniendo la información necesaria (clase del worker y datos que debe usar) también de sus hijos. Además, tiene métodos que sus hijos pueden implementar si hay que realizar alguna tarea cuando el envío tiene éxito o falla.

B.1.3. BasePhotoAndIGUseActivity

Hereda de la anterior, y tampoco muestra nada. Su objetivo es centralizar los métodos para abrir el selector de imágenes, cargar una imagen en la pantalla y obtener el tiempo de uso de Instagram.

B.1.4. RequestErrorActivity

Es la pantalla de error que se muestra si falla alguno de los envíos. Cuando es creada, obtiene de la actividad que la llamó qué trabajo debe revisar (y si no lo encuentra porque fue iniciada de cero, obtiene esa información a partir del estado de la aplicación). Cada vez que la aplicación retoma su actividad (por ejemplo, si estaba abierta pero en segundo plano y es llevada al frente), verifica los trabajos en curso para comprobar si el trabajo que está revisando tuvo éxito. De ser así, realiza las tareas que tenga que hacer luego del envío de datos exitoso, si es que hay alguna (esto depende del lugar desde donde haya sido invocada, cosa que se puede deducir del nombre del trabajo a revisar).

- Si fue invocada desde el formulario de inicio, debe obtener el id de usuario (recibido del servidor) del worker (que lo carga en sus datos de salida) y poner su valor en una preferencia compartida encriptada.
- Si fue invocada desde el envío de una actividad, debe obtener el id de la actividad completada para pasárselo al listado de actividades pendientes (que lo usa para determinar que se completó una actividad).
- En los otros casos no necesita hacer nada.

Terminadas las tareas, actualiza el estado de la aplicación y la redirige a la actividad que corresponda.

B.1.5. SettingsCheckActivity

Muestra el botón para dar permisos de uso de datos a la aplicación. Contiene la lógica para llamar a la configuración de Android correspondiente y para verificar que efectivamente se haya otorgado el permiso.

B.1.6. GetCountryActivity

Permite al usuario seleccionar a qué país pertenece, y le pone el valor seleccionado a una preferencia de la aplicación. Esto se hace para, a futuro, personalizar los textos de la aplicación según el país. También cambia el estado de la actividad para que, si por algún motivo el usuario cierra la aplicación, al volver a abrirla no tenga que completar la revisión de permisos de nuevo.

B.1.7. StartFormActivity

Hereda de BasePostRequestActivity. Carga el formulario de inicio, lee los datos ingresados (controlando que no falte ninguno) y los carga en el JSON que su clase padre enviará en la solicitud. También le envía a la clase padre la URL correcta, la clase del worker que se usa para reenviar su formulario y carga el objeto de datos que usa el worker. Cuando el envío es exitoso, obtiene el id de usuario de la respuesta del servidor y lo carga como preferencia compartida encriptada. Luego redirige a la aplicación al listado de actividades pendientes. Si el envío no es exitoso, redirige a la pantalla de error. En ambos casos se actualiza el estado de la aplicación.

B.1.8. PendingActivitiesActivity

Esta actividad hace varias tareas, dado que es la "pantalla principal". Primero, revisa si se completó el formulario del medio (mediante una preferencia compartida).

- En caso negativo, calcula cuántos días pasaron desde que se completó el formulario de registro. Si ya pasó la mitad del tiempo, redirige la aplicación al formulario del medio.
- En caso positivo, calcula si ya pasó todo el tiempo de la aplicación, en cuyo caso redirige al formulario final.

Luego revisa si la actividad que la invocó incluyó un id de actividad al hacerlo. De ser así, significa que se completó alguna actividad. Finalmente obtiene el listado de preguntas del servidor. Para esto, primero revisa una preferencia compartida para verificar si ya las obtuvo o no.

- Si no las obtuvo, realiza la solicitud al servidor para tenerlas. Si esta solicitud falla, se considera que no hay preguntas. Si tiene éxito, las preguntas se cargan en una base de datos SQLite. En ambos casos, se configuran las notificaciones una vez al día para advertirle al usuario que revise la aplicación.
- Si las obtuvo, revisa el id de actividad mencionado antes y marca esa actividad como completa en la base de datos.

Finalmente se obtiene de la base SQLite las actividades que no se hayan completado y cuya fecha sea menor o igual a la actual. Estas son las actividades pendientes que se mostrarán en el listado.
Hecho todo eso, se carga la pantalla en sí, mostrando un texto si no hay actividades.

B.1.9. UserActivityActivity

Hereda de BasePhotoAndIGUseActivity. Muestra la pregunta seleccionada en la actividad anterior, junto a un espacio para contestarla, y permite subir una foto (usando los métodos heredados de la clase padre). Cuando se presiona "Enviar datos", controla que se hayan llenado los campos obligatorios de la actividad y luego obtiene el tiempo de uso de Instagram del método implementado por su padre. Los datos de la actividad los obtiene de la invocación realizada para llamar a UserActivityActivity. El envío de la foto se hace como string en base64. También implementa los métodos para devolver la URL, la clase de worker, el objeto JSON (incluyendo el uso de Instagram) y los datos del worker que requiere BasePostRequestActivity. En este último caso, como el uso de Instagram es una lista y el string de la imagen es pesado, no se pueden enviar como dato al worker, por lo que se usa otra técnica. El string se carga como preferencia compartida (a la cual el worker tiene acceso) y la lista se carga en una tabla en SQLite para que el worker acceda a ella. Estas dos tareas se realizan en el método implementado cuando el envío falla. Además de esto, se redirige a la pantalla de error y se actualiza el estado.
Por otro lado, si el envío es exitoso, también actualiza el estado, pero redirige a la lista de actividades pendientes. En la invocación, además, agrega el id de la actividad que se acaba de completar, para que se marque como completada.

B.1.10. SendPhotoActivity

Hereda de BasePhotoAndIGUseActivity. Funciona de forma análoga a UserActivityActivity. Las únicas diferencias son que no se muestra una pregunta, el campo obligatorio siempre es la foto y nunca el comentario, y que no agrega ningún id al redirigir porque no se está completando una actividad.

B.1.11. MidTermFormActivity

Hereda de BasePostRequestActivity. Funciona de forma análoga a StartFormActivity. La única diferencia es que cuando el envío es exitoso, en lugar de cargar un id de usuario, carga la preferencia compartida que indica que ya se completó el formulario del medio. Además, actualiza el estado.

B.1.12. EndFormActivity

Hereda de BasePostRequestActivity. Funciona de forma análoga a MidTermFormActivity. La única diferencia es que no carga nada cuando el envío es exitoso.

B.1.13. FinalScreenActivity

Muestra una pantalla final agradeciendo por usar la aplicación. Como es la última, no permite al usuario ir hacia ninguna otra actividad. Actualiza el estado de la aplicación por última vez.

B.2. Fragments

Esta carpeta contiene una única clase, **ActivityRowFragment**. Se usa para cargar el listado de actividades pendientes. Para esto obtiene el listado de PendingActivitiesActivity y mapea los datos obtenidos a un listado de lo que se va a mostrar (fecha y texto de la pregunta). Además configura la acción a realizar cuando se toca una actividad, cargando los datos de la actividad seleccionada en la invocación que realiza.

B.3. Interfaces

Contiene las interfaces usadas para comunicarse con la base de datos SQLite.

B.3.1. DailyIGUsageDataDAO

Contiene los métodos necesarios para interactuar con la tabla que almacena el listado de uso de Instagram en caso de falla en el envío: insertar una lista, obtener la lista de datos y eliminar todos (se eliminan cuando el envío tiene éxito).

B.3.2. DailyQuestionDAO

Contiene los métodos necesarios para interactuar con la tabla que almacena las actividades:

- Obtener todas las preguntas sin contestar hasta el día actual. Estas preguntas serán las que se muestren en el listado de actividades pendientes.
- Obtener una pregunta por el id. Se usa para obtener la información de una actividad a completar.
- Insertar lista, que se usa cuando se obtienen del servidor la primera vez.
- Actualizar pregunta. Se usa para completar una actividad.

También hay disponibles métodos para obtener todos y eliminar todos, que si bien no se usan en la aplicación, sirven para hacer pruebas con la base de datos.

B.4. Models

Contiene modelos que usa la aplicación.

B.4.1. DailyIGUsageData

Es el modelo de la información de uso de Instagram, que se usa para el almacenamiento en base de datos. Contiene la fecha del dato y un número indicando el tiempo de uso en milisegundos.

B.4.2. DailyQuestion

Es el modelo de una actividad, que se usa para el almacenamiento en base de datos. Contiene el id de la pregunta, la fecha en la que debe ser preguntada, el texto de la pregunta, un booleano indicando si ha sido contestada, y dos booleanos que indican respectivamente si el texto y la foto son obligatorios.

B.4.3. QuestionResponse

Es el modelo de una pregunta que se recibe del servidor, y se usa para parsear la respuesta recibida de éste y luego convertir los datos a DailyQuestion. Contiene el id de la pregunta, un número que indica en qué día a partir de que se completó el registro debe ser respondida, el texto de la pregunta y dos booleanos que indican respectivamente si el texto y la foto son obligatorios.

B.5. Utils

Contiene clases estáticas y enumerados útiles para diferentes tareas.

B.5.1. ActivityDataFields

Clase estática que contiene los nombres de los campos que se envían en el objeto JSON de información de una actividad completada (excepto los nombres de uso de Instagram, que se encuentran en IGUsageDataFields).

B.5.2. ApiCallResult

Enumerado que representa los resultados de una solicitud HTTP. Los valores posibles son: no hay respuesta (aún no contestó el endpoint), fallo y éxito.

B.5.3. AppDatabase

Clase que se usa para crear la base de datos SQLite y poder invocar las interfaces creadas.

B.5.4. AppState

Enumerado que representa los estados de la aplicación. Los valores posibles son: sin iniciar, ingresar datos de inicio, standby, medio (formulario del medio), fin (formulario del fin), error en solicitud de inicio, error en solicitud de fin, error en solicitud estándar (solicitud de actividad), error en solicitud de medio y error en solicitud de foto.

B.5.5. Constants

Clase estática que contiene las constantes que se usan en la aplicación. Esto incluye claves para preferencias compartidas, códigos de solicitud, URLs y nombres de trabajos, entre otros.

B.5.6. Converters

Permite convertir una fecha a milisegundos y viceversa. Esta clase es usada por la base de datos para poder almacenar fechas (pues se almacenan en milisegundos; con esta clase se le puede pasar una fecha y hace automáticamente la conversión).

B.5.7. EndFormDataFields

Clase estática que contiene los nombres de los campos que se envían en el objeto JSON de información del formulario final.

B.5.8. IGUsageDataFields

Análogo a EndFormDataFields pero para la sección de uso de Instagram (que se envía en dos objetos diferentes pero es igual en ambos).

B.5.9. MiddleFormDataFields

Análogo a EndFormDataFields pero para la información del formulario del medio.

B.5.10. PhotoDataFields

Análogo a ActivityDataFields pero para la información de la foto.

B.5.11. ProfileType

Enumerado que representa los tipos de perfil de Instagram. Los valores posibles son público y privado.

B.5.12. ProjectTimeComparedToNormal

Enumerado que representa la actividad de un usuario durante el proyecto comparada a cómo es normalmente. Los valores posibles son más, menos o igual.

B.5.13. StartFormDataFields

Análogo a EndFormDataFields pero para la información del formulario inicial.

B.5.14. UploadOrSeeOthersContent

Enumerado que representa la preferencia del usuario en su actividad de Instagram. Los valores posibles son subir o ver contenido de otros.

B.5.15. UploadsComparedToProjectTime

Enumerado que representa la cantidad de contenido que sube el usuario normalmente comparado con el tiempo del proyecto. Los valores posibles son más o menos.

B.5.16. WouldChangeNick

Enumerado que representa si el usuario cambiaría su nick de Instagram. Los valores posibles son: no, ya lo ha hecho, lo ha considerado, o no contesta.

B.6. Workers

Los workers son las clases que se encargan de realizar los trabajos en segundo plano, sin importar si la aplicación está siendo usada o no. Un trabajo puede devolver tres resultados: éxito, fallo o reintentar. Si se devuelve "fallo", el trabajo no se reintenta (que sí se hace si se devuelve reintentar", como es lógico). Además, los workers trabajan con un objeto "Data"que les permite tomar datos de entrada y devolver datos de salida en el resultado, siempre que sean tipos primitivos.

B.6.1. BaseRequestWorker

Su función principal es realizar el trabajo de enviar datos al servidor, dejando únicamente algunos detalles específicos abstractos para que los otros workers que tengan que enviar datos (que son todos menos uno) los implementen. Usa un método abstracto para obtener el objeto JSON a partir del Data de entrada que recibió. Envía una solicitud al servidor, obteniendo la URL de sus hijos y agregando el id de usuario, y luego se queda esperando al resultado de la llamada.
Si la llamada es exitosa, tiene un método que pueden implementar sus hijos si hay que realizar alguna acción. En ese caso, el trabajo devuelve un resultado exitoso y dicho resultado incluye un objeto de datos devuelto por un método que es implementado por sus hijos. De ese modo, si se necesita obtener algún dato de la respuesta del servidor, se puede pasar a la actividad que lo necesite mediante el objeto de salida del worker.
Si la llamada, o el parseo de la respuesta, o cualquier otra parte del proceso falla, el worker devuelve "reintentar".

B.6.2. SendDailyNotificationWorker

Este worker se encarga de enviar una notificación avisándole al usuario que revise la aplicación por si tiene actividades pendientes. Como esa es su única tarea, siempre retorna un resultado exitoso.

B.6.3. SendEndDataWorker

Hereda de BaseRequestWorker. Implementa los métodos para pasar la URL correcta y el objeto JSON a partir del objeto Data que necesita la clase padre para funcionar. Reintenta el envío de los datos del formulario de fin.

B.6.4. SendMidDataWorker

Hereda de BaseRequestWorker. Es análogo a SendEndDataWorker, pero con el formulario de fin, y además implementa el método para el caso de llamada exitosa de la clase padre. Lo usa para poner en "true" la preferencia compartida que indica que se completó el formulario del medio.

B.6.5. SendPhotoWorker

Hereda de BaseRequestWorker. Es análogo a SendMidDataWorker, pero con los datos de una foto (no actividad), y la tarea que realiza en caso de éxito es eliminar los datos de uso de Instagram de la base de datos (para evitar problemas de inconsistencia). Además, en el método de cargar el objeto JSON a partir del Data, lee el string en base64 de la preferencia compartida correspondiente y los datos de uso de Instagram de la base de datos (recordar que no se pueden pasar en el Data).

B.6.6. SendStartDataWorker

Hereda de BaseRequestWorker. Es análogo a SendMidDataWorker, pero con el formulario inicial, y la tarea que realiza en caso de éxito es leer de la respuesta el id de usuario, el cual carga en el objeto de retorno del trabajo (mediante el método de la clase padre que sirve para ello).

B.6.7. SendUserActivityDataWorker

Hereda de BaseRequestWorker. Es análogo a SendPhotoWorker, pero con los datos de una actividad, y en caso de éxito, además de eliminar los datos de Instagram, también obtiene el id de la pregunta del objeto JSON de la solicitud, y lo devuelve como retorno del trabajo (este id se usará para marcar que la actividad está completa).

B.7. Resources

Android provee varios recursos para una aplicación: strings, layouts, temas e íconos. Se agregaron todos los textos necesarios en strings y se modificaron los layouts creados automáticamente. Además se agregó un ícono de Material Design para el botón de compartir foto.

B.8. Bibliotecas usadas

Se agregaron también varias bibliotecas, algunas de Google/Android y otras de terceros, para manejar diferentes aspectos de la aplicación, a saber:

- **Work-runtime:** es la biblioteca de Android que permite usar los workers.
- **OkHttp:** permite manejar solicitudes HTTP de forma más sencilla que el manejo nativo de Android.

- **Facebook-login:** permite realizar el login con Facebook. Si bien el prototipo no la usa en la práctica, varios de los elementos que provee se encuentran en la aplicación.
- **Gson:** es una biblioteca de Google que permite parsear objetos JSON.
- **Guava:** permite manejar encoding en base64.
- **Security-crypto:** es la biblioteca de Android que permite usar preferencias compartidas encriptadas.
- **Room-runtime:** es la API de Google que permite manejar SQLite de forma sencilla.

Apéndice C

Servidor web

Tal como se mencionó el servidor web es una API Rest implementada en .NET core 5 y entity Framework.

A continuación se describen los endpoints disponibles con sus respectivas entradas y salidas. El único de ellos que no requiere autenticación es /api/Forms/Start. Para el resto se tendrá que llamar con el header Authorization y valor "Device user_id"

- POST /api/Forms/Start

  Formulario inicial para registrar al usuario, retorna JSON conla propiedad "user_id" que es un Guid, en el caso de que este fomulario falle se puede reintentar la solicitud con el próximo endpoint. El campo fb_token por ahora no se está utilizando, se utilizaría en el caso que se integre el servicio con la API de Instagram.

  Entrada:

```
{
  "country": "string",
  "ig_User": "string",
  "fb_token": "string",
  "date_of_completion": "2021-11-02T23:50:43.274Z",
  "age": 0,
  "real_name_in_network": true,
  "followers": 0,
  "other_networks": "string",
  "most_used_network": "string",
  "weekly_ig_uploaded_content": "string",
  "profile_type": "Public",
  "preferred_contents": "Upload",
  "would_change_nick": "No",
  "profile_pic_defines_identity": true,
  "profile_pic_helps_others_see_their_real_appearance":
    true,
```

```
  "profile_pic_helps_others_see_their_desired_appearance":
    true
}
```

Validaciones:

- Country debe corresponder al ISO 31662 [72] del país
- Age mayor que cero
- Date_of_completion no puede tener la fecha por defecto
- Real_name_in_network no puede ser vacío
- Followers no puede ser vacío
- Other_networks no puede ser vacío
- Most_used_network no puede ser vacío
- Weekly_ig_uploaded_content no puede ser vacío
- Profile_type posbibles valores: "Public/"Private"
- Preferred_contents tiene como posibles valores: "Upload/"Other"
- Would_change_nick tiene como posibles valores:: "No"/"Done"/"Considered"/"NoAnswer"
- Profile_pic_defines_identity no puede ser vacío
- Profile_pic_helps_others_see_their_real_appearance no puede ser vacío
- Profile_pic_helps_others_see_their_desired_appearance no puede ser vacío

Salida:

```
{
  "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
}
```

- POST /api/Forms/RetryRegister

  Este formulario se utiliza cuando falla el de registro inicial, se envía un Guid dummy para la autenticación y el sistma retorna el Guid ya generado o genera uno nuevo.

  Entrada:

```
{
  "country": "string",
  "ig_User": "string",
  "fb_token": "string",
  "date_of_completion": "2021-11-02T23:50:43.274Z",
  "age": 0,
  "real_name_in_network": true,
  "followers": 0,
  "other_networks": "string",
```

```
  "most_used_network": "string",
  "weekly_ig_uploaded_content": "string",
  "profile_type": "Public",
  "preferred_contents": "Upload",
  "would_change_nick": "No",
  "profile_pic_defines_identity": true,
  "profile_pic_helps_others_see_their_real_appearance":
    true,
  "profile_pic_helps_others_see_their_desired_appearance":
    true
}
```

Validaciones:

- Country debe corresponder al ISO 31662 [72] del país
- Age mayor que cero
- Date_of_completion no puede tener la fecha por defecto
- Real_name_in_network no puede ser vacío
- Followers no puede ser vacío
- Other_networks no puede ser vacío
- Most_used_network no puede ser vacío
- Weekly_ig_uploaded_content no puede ser vacío
- Profile_type tiene como posibles valores: ”Public/”Private”
- Preferred_contents tiene como posibles valores: “Upload/“Other”
- Would_change_nick tiene como posibles valores: “No”/“Done”/“Considered”/“NoAnswer”
- Profile_pic_defines_identity no puede ser vacío
- Profile_pic_helps_others_see_their_real_appearance no puede ser vacío
- Profile_pic_helps_others_see_their_desired_appearance no puede ser vacío

Salida:

```
{
  "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
}
```

- POST /api/Forms/Middle

  Estas respuestas del usuario corresponden al formulario que se presenta en el medio del estudio.

  Entrada:

```
{
  "date_of_completion": "2021-11-02T23:52:13.395Z",
  "favorite_uploaded_picture": "string",
  "reason_for_liking": "string",
  "reason_for_uploading": "string",
  "defines_identity": true,
  "helps_others_see_their_real_appearance": true,
  "helps_others_see_their_desired_appearance": true
}
```

Validaciones:

- Date_of_completion no puede tener la fecha por defecto
- Favorite_uploaded_picture no puede ser vacío
- Reason_for_liking no puede ser vacío
- Reason_for_uploading no puede ser vacío
- Defines_identity no puede ser vacío
- Helps_others_see_their_real_appearance no puede ser vacío
- Helps_others_see_their_desired_appearance no puede ser vacío

Salida: 200 Success

- POST /api/Forms/End

  Estas respuestas del usuario corresponden al formulario final del estudio.

  Entrada:

```
{
  "date_of_completion": "2021-10-30T02:58:42.821Z",
  "would_upload_same_pics": true,
  "regrets_post": true,
  "got_expected_feedback_in_regretted_post": true,
  "effect_of_regretted_photo": "string",
  "reason_for_not_uploading_planned_content": "string",
  "uploads_more_or_less_than_in_project_time": "More",
  "content_seen_influences_self_perception": true,
  "content_published_that_defines_them_most_in_project_time
    ": "string",
  "favorite_content_published_in_project_time": "string",
  "thought_carefully_about_published_content": true,
  "photos_shared_in_project_time_compared_to_normal": "More
    ",
```

```
  "
      care_in_content_shared_in_project_time_compared_to_normal
      ": "More"
}
```

Validaciones:

- Would_upload_same_pics no puede ser vacío
- Regrets_post no puede ser vacío
- Reason_for_not_uploading_planned_content no puede ser vacío
- Uploads_more_or_less_than_in_project_time tiene como posibles valores: "More" / "Less" / "Same"
- Content_seen_influences_self_perception no puede ser vacío
- Favorite_content_published_in_project_time no puede ser vacío
- Thought_carefully_about_published_content no puede ser vacío
- Photos_shared_in_project_time_compared_to_normal tiene como posibles valores: "More" / "Less" / "Same"
- Care_in_content_shared_in_project_time_compared_to_normal tiene como posibles valores: "More" / "Less" / "Same"

Salida: 200 Success

- GET /api/Questions

  Una vez configuradas las preguntas diarias en la base de datos, la App las obtiene utilizando este endpoint para ir mostrando las notificaciones cuando corresponda. El campo "ask_on_day_from_project_start" indica dentro de qué día del estudio deberá aparecer la notificación.

  Salida:

```
[
  {
    "id": 0,
    "question": "string",
    "ask_on_day_from_project_start": 0,
    "is_text_mandatory": true,
    "is_photo_mandatory": true
  }
]
```

- POST /api/Activity

  Una vez completada la pregunta diaria propuesta se envía la información que puede contener una foto y/o comentarios. También se envía el uso diario de Instagram para

no llamar específicamente para eso, de esta forma se minimizan los posibles errores de conectividad entre la App y la Web Api.

Entrada:

```
{
  "question_id": 0,
  "answer": "string",
  "original_question_date": "2021-11-02T23:54:37.032Z",
  "date_of_answer": "2021-11-02T23:54:37.032Z",
  "shared_photo": "string",
  "daily_usage_list": [
    {
      "date": "2021-11-02T23:54:37.032Z",
      "usage_time": 0
    }
  ]
}
```

Validaciones:

- Date_of_answer no puede tener la fecha por defecto
- Original_question_date no puede tener la fecha por defecto

Validaciones sobre “daily_usage_list”

- Date no puede tener la fecha por defecto
- Usage_time no puede ser vacío, en ese caso se considera cero

Salida: 200 Success

- POST /api/Photos Cuando el usuario quiera puede compartir fotos con comentarios, este endpoint guarda la información y al igual que el endpoint anterior actualiza el uso diario de Instagram. Entrada:

```
{
  "date_of_answer": "2021-11-02T23:53:57.678Z",
  "shared_photo": "string",
  "photo_comment": "string",
  "daily_usage_list": [
    {
      "date": "2021-11-02T23:53:57.678Z",
      "usage_time": 0
    }
  ]
}
```

Validaciones:

- Date_of_answer no puede tener la fecha por defecto
- Shared_photo no puede ser vacío

Validaciones sobre "daily_usage_list"

- Date no puede tener la fecha por defecto
- Usage_time no puede ser vacío, en ese caso se considera cero

Salida: 200 Success

El tiempo diario de uso de Instagram se sobrescribe si este ya había sido ingresado, se asume que el último es el más actualizado del día.

Apéndice D

Wireframes de aplicación Web

Para el sitio Web propuesto se diseñaron las siguientes pantallas que fueron mostradas en la etapa de relevamiento de requisitos.

D.1. Login

En este punto los investigadores se loguean con usuario y contraseña (Figura D.1).

D.2. Observaciones

A medida que los investigadores fueran observando historias de Instagram u otros eventos utilizarían esta pantalla para subir comentarios e imágenes relevantes (Figura D.2).

D.2.1. Imágenes

En este punto se pueden subir imágenes complementarias a las observaciones de los investigadores sobre algún usuario de Instagram (Figura D.3).

D.3. Datos iniciales

Con el objetivo de cruzar datos de las encuestas y los relevados por la aplicación móvil se ofrece la posibilidad de subir un excel con los resultados de las encuestas (Figura D.4).

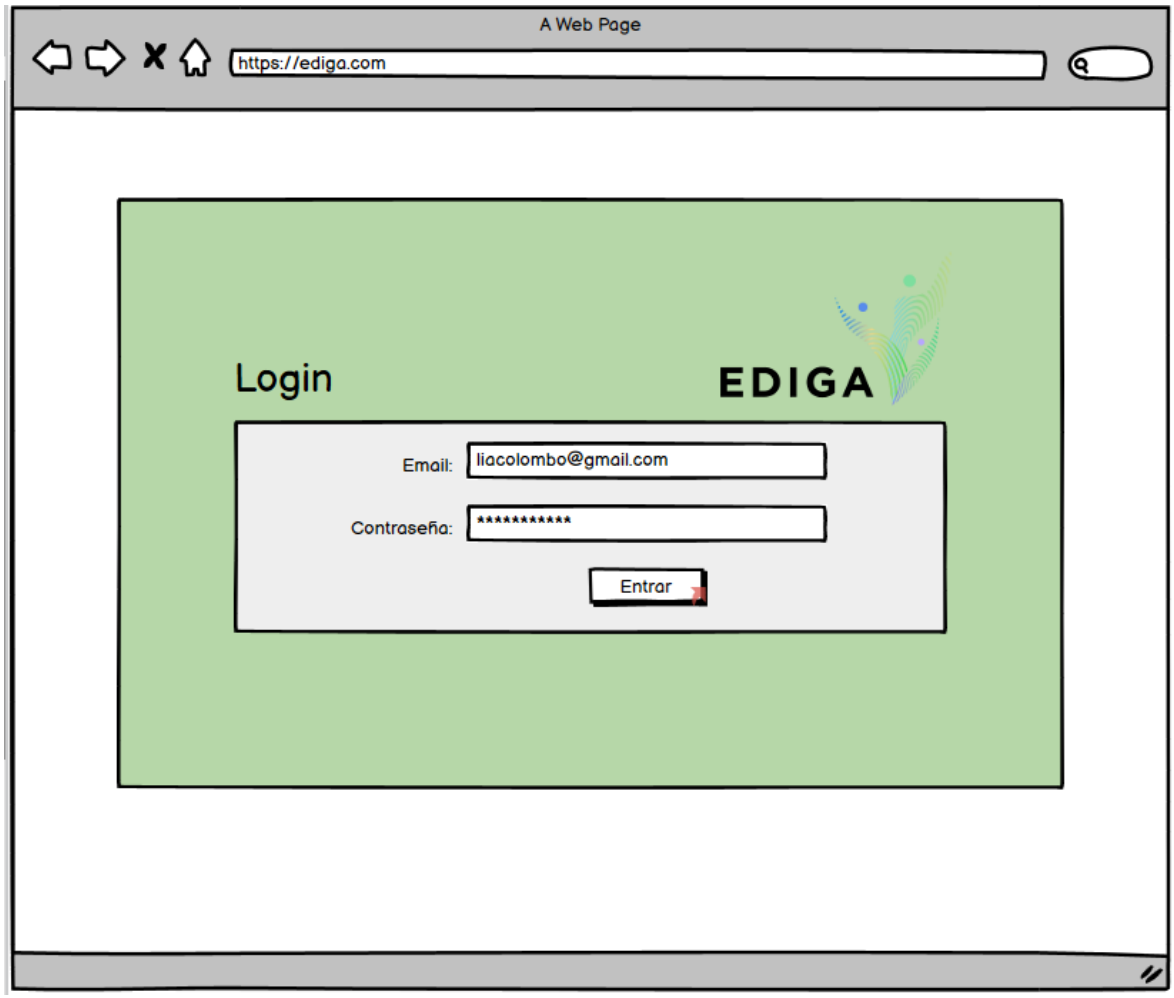

Figura D.1: Login del investigador

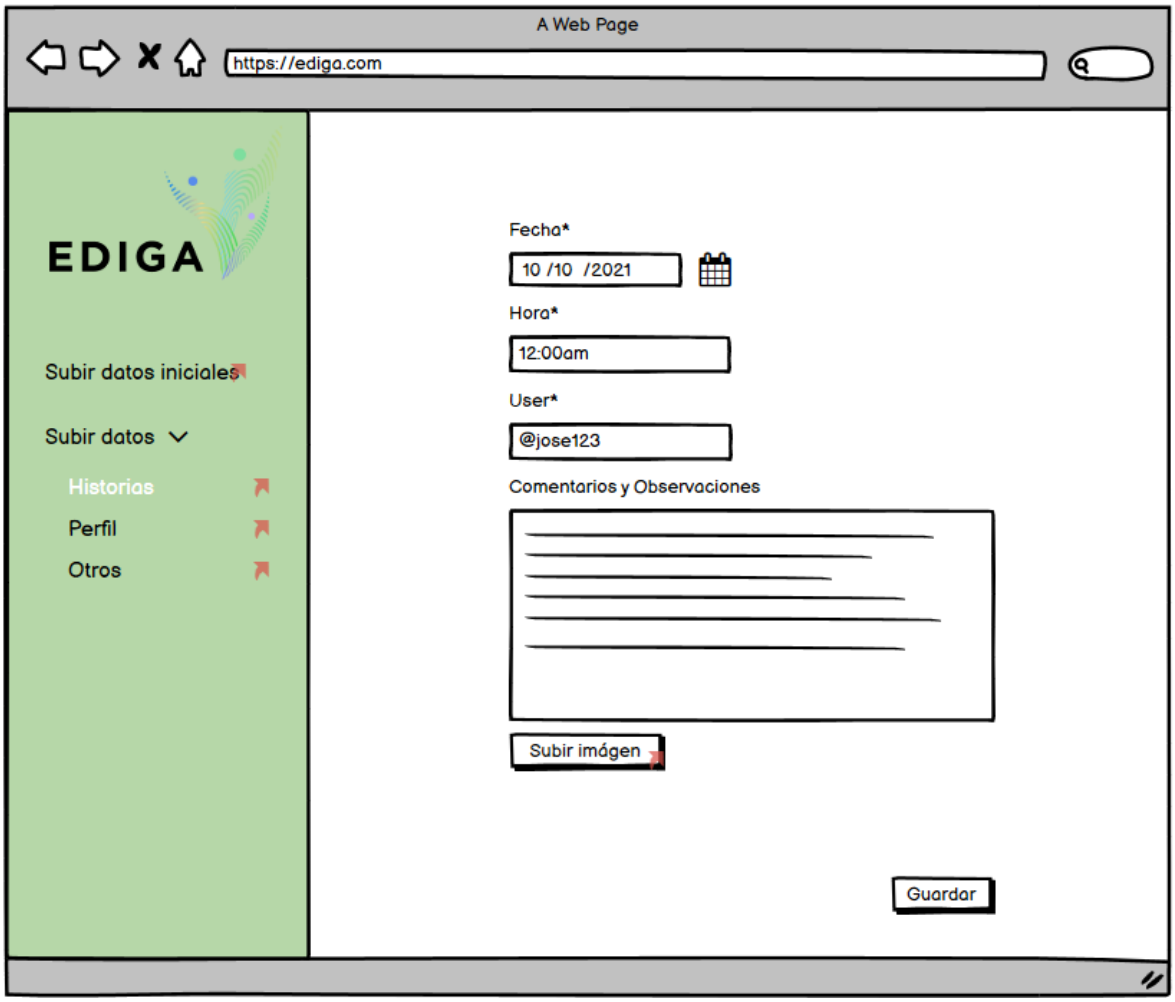

Figura D.2: Agregar observaciones

Figura D.3: Agregar imágenes

Figura D.4: Agregar datos iniciales