BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

23 DE MARZO DE 2021

"APLICACIÓN MÓVIL ANDROID PARA EL LEVANTAMIENTO DE ENCUESTAS APLICADO A LA ACTUALIZACIÓN DEL REGISTRO NACIONAL DE PRODUCTORES DE LECHE"

TESINA PRESENTADA PARA OBTENER EL GRADO DE:

LICENCIATURA EN ELECTRÓNICA

PRESENTA

C. ALBERTO TORRES GIL

DIRECTOR DE TESINA

DRA. MARÍA MONSERRAT MORÍN CASTILLO

ASESORES DE TESINA

DRA. MARÍA MONSERRAT MORÍN CASTILLO

M.C. RICARDO ÁLVAREZ GONZÁLEZ

Índice

Capítulo 1. Introducción

Este proyecto comenzó a inicios de Marzo 2019, en el marco de los programas de precios de garantía, cabe mencionar, que este proyecto está en el contexto del programa de precios de garantía para pequeños y medianos productores de leche en Encarnación de Díaz (Jalisco), y es un proyecto, que se enmarca en el Plan de Desarrollo de este gobierno (sexenio 2018-2024).

En este contexto, se dió a conocer a los productores lecheros de la zona, los programas y apoyos que desde la Federación se darán así como la seguridad de contar con un precio de garantía para pequeños productores de $8.20 pesos por litro entregado a LICONSA (Leche Industrializada Conasupo SA de CV).

Este proyecto fue indispensable para actualizar el RNPL (Registro Nacional de Productores de Leche) y lograr que el subsidio que representa el precio de garantía llegase a quienes corresponde.

Para ello se realizó un levantamiento de encuestas de los productores de leche a nivel Nacional mediante un dispositivo móvil que fuera capaz de capturar digitalmente las encuestas con georeferencia y toma de evidencia (fotografías). Este estudio, así como el proyecto se inició en la región de Los Altos de Jalisco.

Como dispositivo móvil fue seleccionado un smartphone (dispositivo móvil inteligente) Android. Un dispositivo móvil se puede definir como un aparato de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, que ha sido diseñado específicamente para una función. De acuerdo con esta definición existen multitud de dispositivos móviles, desde los reproductores de audio portátiles hasta los navegadores GPS, pasando por los teléfonos móviles inteligentes, los PDAs y los Tablet PCs.

1.1 Objetivo

Desarrollar una aplicación de teléfono móvil para levantar encuestas en campo, georeferenciadas, para ser enviadas posteriormente a los servidores de LICONSA.

Objetivos Específicos:

- Funcionar en un teléfono móvil de sistema operativo Android
- Permitir realizar las encuestas en ambientes remotos aun sin tener acceso a internet.

- Permitir almacenar 3 fotografías (productor, identificación oficial y vacas), georeferencia como evidencia y almacenar la información dentro del dispositivo de forma segura.
- Envíar información a servidores LICONSA a través de un API (*Application Programming Interface*) con protocolo SOAP (*Simple Object Access Protocol*).

1.1 Justificación

En el marco laboral de una Empresa especializada en aplicar encuestas, se le asignó mediante una licitación, el proyecto de Actualizar el RNPL el cual consistía en la realización de encuestas a los productores de Leche que venden leche a LICONSA en el territorio nacional. Para llevar a cabo este proyecto se le requirió también desarrollar una herramienta digital que permitiera la captura de las encuestas en un dispositivo móvil. Esta consistiría en una aplicación nativa para dispositivos móviles Android.

Fue importante y fundamental desarrollar una aplicación móvil desde cero, para que se ajustara perfectamente a todas y cada una de las necesidades de este proyecto con impacto nacional.

La empresa optó por realizar una aplicación nativa Android porque se deseaba una herramienta que fuera capaz de utilizar el hardware de geolocalización y la cámara para tomar fotografías desde un dispositivo móvil, además de poder funcionar en entornos remotos sin acceso a Internet, estas características solo podrían ser llevadas a cabo desde una aplicación nativa

Capítulo 2. Dispositivos móviles

Este capítulo tiene como finalidad dar a conocer los aparatos que fueron marcando hitos en la historia de la tecnología de los dispositivos móviles y el surgimiento de sus aplicaciones y sistemas operativos más populares.

2.1 Antecedentes de los dispositivos móviles

Primer teléfono móvil en Estados Unidos

En la *Figura 1*, se muestra el modelo *DynaTAC 8000X* creado en 1983 por Motorola, este modelo fue diseñado por el ingeniero de Motorola Rudy Krolopp en 1983. El modelo pesaba 793 gramos y tiene unas dimensiones completas de 33 x 4.45 x 8.9 centímetros y tenía un valor de casi $4,000 dólares estadounidenses [1].

En su pantalla, tenía *LEDs* rojos que mostraban los números que se marcaban con el teclado. Respecto al teclado, además de los números y teclas de marcación que era un grupo de doce teclas, se tenía un apartado de funciones especiales:

Rcl: para la rellamada
Clr: limpiaba la pantalla
Snd: marcaba el número
Sto: almacenaba en memoria el número en pantalla
End: terminaba las llamadas
Fcn: tecla de función
Pwr: tecla de encendido
Lock: bloqueba el teclado del teléfono
Vol: ajustaba el volumen

Figura 1: Teléfono móvil Motorola DynaTAC 8000X

El Teléfono de Nokia: Mobira Talkman

En la *Figura 2* se muestra el modelo Mobira Talkman de Nokia creado en 1984, que solucionaba el problema básico del DynaTAC el cual solo permitía un funcionamiento máximo de 60 minutos. El modelo de Nokia, el Mobira Talkman incluía una autonomía de varias horas de funcionamiento continuo, este modelo tuvo una notable demanda en los mercados nórdicos y ello llevó a Nokia a tener nuevos clientes, incluyendo los estadounidenses y los británicos. La necesidad de contar con teléfonos de más fácil manejo y transportación, derivó en innovaciones tecnológicas que fueron muy notorias en los tamaños y el peso de los mismos.

Figura 2: Teléfono móvil Nokia Mobira Talkman

Motorola MicroTAC

Motorola siguió innovando en el terreno móvil y en 1989 lanzó el teléfono móvil más pequeño y ligero de la época, el primer móvil con diseño de tapa, que permitía reducir el tamaño. En la *Figura 3* se muestra el Motorola MicroTac, el cual era un teléfono celular fabricado por primera vez como un análogo de la versión en versiones compatibles con GSM(*Global System for Mobile communications*) y TDMA (*Time Division Multiple Access*). El MicroTac introdujo un diseño nuevo e innovador "*flip*", esto estableciólа norma y se convirtió en el modelo para los teléfonos plegables modernos. "*TAC*" era una abreviatura de: Total del área de cobertura.

Figura 3: Teléfono móvil Motorola MicroTac

Motorola desarrolló teléfono para ser utilizado en automóviles

En 1994 Motorola crea un teléfono para ser utilizado en automóviles, su modelo de teléfono 2900, este modelo se muestra en la *Figura 4*, el teléfono venía conectado por cable, como un teléfono tradicional a una bolsa que integraba el transmisor, el receptor y una batería más pesada. Ello hacía posible una emisión con mayor potencia, algo importante en aquéllos años cuando la cobertura no era todo lo buena que hoy día.

Figura 4: Teléfono móvil Motorola 2900

Nokia: el primer smartphone

En 1997 Nokia crea el primer *smartphone* que venía con una *CPU* derivada de un Intel 386 y 8 Mbytes de *RAM*. En la *Figura 5* se muestra teléfono en cuestión, el Nokia 9000i . Trajo al mundo el diseño tradicional de los Nokia *Communicator* en el que el teléfono podía abrirse de manera horizontal, mostrando una pantalla panorámica y un teclado *QWERTY*. El teléfono podía recibir y enviar faxes, *SMS* y emails.

Figura 5: Teléfono móvil Nokia 9000i

2.2 Internet móvil: Sus comienzos

En 1999 se empiezan a popularizar las conexiones WAP (*Wireless Application Protocol*), algo que Nokia comenzó a implementar en sus terminales, en particular con su Nokia 7110 mostrado en la *Figura 6*, conocido por ser el móvil que publicitaron en la primera película de la saga Matrix. Las aplicaciones móviles empezaron a revolucionar la vida diaria, las primeras aplicaciones móviles datan de finales de los 90, estas eran principalmente, lo que se conoce como la agenda, aplicaciones de video juegos como *ARCADE* games, los editores de *ringtone*, etc, estas cumplían funciones muy elementales y su diseño era bastante simple.

Figura 6: Teléfono móvil Nokia 7110

Primer *PocketPC*

Los primeros *PocketPC* iniciaron en el año 2000 y tenían como sistema operativo el llamado Windows CE 3.0. Por su nombre podría parecer que era una versión aligerada del sistema operativo más utilizado en el mundo, pero en realidad no tenía mucho que ver con éste, ya que el *PocketPC* era un dispositivo de mano que solo permitia grabar, enviar y recibir e-mails, contactos, citas, mostrar archivos multimedia, juegos, intercambiar mensajes de texto y navegar por la web.

La compañía RIM comienza a despegar

En la figura *7* se muestra el modelo 5810 de BlackBerry el cual fue presentado en el año 2002, el primer modelo de la serie BlackBerry en integrar soporte de datos móviles. Gracias a dicha característica y a su teclado, disponía de funciones de agenda personal y soporte de email push.

En este mismo año se lanzan al mercado los *Tablet PC*, dispositivos que iban equipados con Windows XP *Tablet PC Edition*, y permitían su manejo a través de un lápiz. Tenía soporte para reconocimiento de escritura a mano y reconocimiento de voz, pero su uso seguía siendo prácticamente el mismo que el de un ordenador portátil.

Figura 7: Teléfono móvil BlackBerry 5810

2.3 Tipos de dispositivos móviles

T38 y *DuPont Global Mobility Innovation Team* en 2005 propusieron las categorías de dispositivos móviles con los siguientes estándares:

- Dispositivo Móvil de Datos Limitados (*Limited Data Mobile Device*); teléfonos móviles clásicos. Se caracterizan por tener un pantalla pequeña de tipo texto. Ofrecen servicios de datos generalmente limitados a *SMS* y acceso *WAP*.
- Dispositivo Móvil de Datos Básicos (*Basic Data Mobile Device*); se caracterizan por tener una pantalla de mediano tamaño, menú o navegación basada en iconos, y ofrecer acceso a emails, lista de direcciones, *SMS* y en algunos casos, un navegador web básico. Un ejemplo de este tipo de dispositivos son los teléfonos inteligentes ("*smartphones*").
- Dispositivo Móvil de Datos Mejorados (*Enhanced Data Mobile Device*); se caracterizan por tener pantallas de medianas a grandes (por encima de los 240 x 120 *pixels*), navegación de tipo *stylus*, y que ofrecen las mismas características que el "Dispositivo Móvil de Datos Básicos" (*Basic Data Mobile Devices*) más aplicaciones nativas como aplicaciones de *Microsoft Office Mobile (Word, Excel, PowerPoint)* y aplicaciones corporativas usuales, en versión móvil, como *Sap*, portales *intranet*, etc. Este tipo de dispositivos incluyen sistema operativo *Windows Mobile*.

Los Ultra-mobile PC o UMPC

En 2006 *Windows* entra en el ámbito de los *Ultra-mobile PC*, con unos equipos más pequeños también con lápiz y pantalla *TFT*. Muchos fabricantes lanzaron este tipo de dispositivos pero en realidad no gozaron de demasiado éxito entre los usuarios.

Nacimiento del primer iPhone y las APPs

En 2007 Apple anunció el smartphone más popular de la historia, el iPhone, y con él llegaron multitud de mejoras al mercado, como es la popularidad de las pantallas táctiles, una facilidad para acceder a Internet y capacidades multimedia avanzadas. En 2009, surgió el tercer modelo de iPhone *3GS*, y otros dispositivos que también

reproducieron varias de sus características pero basados en otros sistemas operativos como son Nokia N97, Palm Pre o cualquier otro dispositivo con Android por ejemplo el HTC Hero.

La evolución de las *APPs* se dió rápidamente gracias a las innovaciones en tecnología WAP (*Wireless Application Protocol*) que es un estándar abierto internacional para aplicaciones que utilizan las comunicaciones inalámbricas, por ejemplo: acceso a servicios de Internet desde un teléfono móvil y la transmisión de data EDGE (*Enhanced Data Rates for GSM Evolution*), la cual es una tecnología de telefonía móvil celular, que actúa como puente entre las redes 2G y 3G.

Esto vino acompañado de un desarrollo muy fuerte de los celulares. Entre ellas Android, la competencia más grande del sistema operativo del iPhone, y con esto empieza el boom de las apps, juegos, noticias, diseño, arte, fotografía, medicina y todo en las manos de las personas gracias a la revolución de las aplicaciones móviles.

2.4 Aparece el primer dispositivo Android de la historia

En la *Figura 8* se muestra el HTC Dream comercializado también como T-Mobile G1 y denominado popularmente Google Phone o GPhone, el cual es un dispositivo de telefonía móvil construido por HTC. Fue lanzado al mercado el 22 de octubre de 2008, a un precio estimado de $179 usd, siendo, el primer dispositivo móvil de comunicación en incorporar el sistema operativo móvil de Google denominado Android. Entre sus características principales están la incorporación de una pantalla táctil, sistema intuitivo de acceso instantáneo a información con solo unos cuantos toques de pantalla, acceso completo a funciones de conectividad y navegabilidad en internet, cámara de 3.2 *Megapixeles* con autofoco, pantalla de inicio totalmente personalizable, acceso a 3G de alta velocidad y Wi-Fi, capacidad de optimizar y personalizar el dispositivo mediante diversas aplicaciones, teclado deslizable, acceso directo a todas las aplicaciones Google, tales como Gmail y Google Maps. El dispositivo integraba una tarjeta de memoria de 1 *Gigabyte*.

Figura 8: Teléfono móvil HTC Dream

2.5 Evolución de las Aplicaciones Móviles

La aparición de las primeras aplicaciones móviles tenía como razón principal organizar el trabajo de ejecutivos y profesionales. Agendas, calculadoras, bloc de notas, editores de texto, hojas de cálculo, contactos, email, etc. Sin embargo, con el paso del tiempo, el sector del entretenimiento fue creciendo. El teléfono móvil había dejado de ser un aparato con el que su pueden hacer y recibir llamadas y mensajes de texto para transformarse es una potente máquina, equiparable a un ordenador, con la que podía llevar a cabo casi cualquier tarea diaria [2].

2.6 Ventajas del uso de las Aplicaciones Móviles Nativas

La esencia de las aplicaciones nativas es que están escritas para una plataforma específica, hoy en día las más utilizadas son iOS y Android. Esto significa que los desarrolladores pueden proporcionar una integración mucho mejor entre las funcionalidades de las aplicaciones con las características de hardware de los móviles: Bluetooth, NFC, sensor GPS, cámara fotográfica, video, micrófono, acelerómetro, giroscopio, etc. El acceso a estas funciones es especialmente importante para trabajar con datos como la ubicación geográfica del usuario [3].

En el capítulo 3 se abordará el desarrollo de una aplicación nativa en teléfono móvil inteligente, por ser los tipos de dispositivos más utilizados y conocidos en la actualidad, los que ofrecen mayor variedad de aplicaciones multimedia y los que más posibilidades de evolución presentan en este sentido.

Capítulo 3. Diseño y desarrollo de aplicación móvil Android

En este capítulo se abordará el proceso que se llevó a cabo para la realización de la aplicación móvil en un dispositivo Android para la realización de encuestas.

Como primer paso, se emprendió la revisión del padrón de productores de leche considerando los siguientes parámetros: Centro de Acopio, Ubicación del Centro de Acopio, Ubicación del Centro de Producción, Vacas en Producción, Vacas Secas, Producción de litros de leche por día y litros vendidos a LICONSA por día.

Entre las situaciones que se presentaron, fue que al realizar dicha revisión aparecieron ciertos inconvenientes o factores no contemplados, donde existen productores que pudiendo ser dados de alta en el padrón por cumplir los requisitos (por ser pequeños y medianos productores) no habían podido ingresar o se les había negado el ingreso por no estar actualizadas las reglas de operación, y otros tantos productores que estando dentro del padrón desconocían el procedimiento para vender su producción de leche a LICONSA mediante un Centro de Acopio autorizado.

Para resolver la situación anterior y actualizar de forma clara el padrón, se aprovechó la experiencia que la empresa tiene en la realización de encuestas desde hace ya varios años, y fue seleccionada por SEGALMEX (Seguridad Alimentaria Mexicana) para encuestar a los productores de leche, principalmente a los que venden leche a LICONSA y con estos datos actualizar el RNPL.

En este marco, se desarrolló el proyecto basado en el siguiente cuestionario diseñado para solicitar información a los productores de leche, el cuestionario fue desarrollado por SEGALMEX e incluía un conjunto de preguntas y respuestas de tipo texto, numéricas y de opción múltiple.

En la **Tabla 1**, se muestra una comparación de las ventajas y desventajas de realizar encuestas en papel contra realizarla mediante una aplicación móvil. Una de las características importantes de realizar encuestas en papel, es que los costos son mínimos en comparación del equipo y desarrollo de software que se utiliza en una aplicación móvil para realizar encuestas, en este último aunque el costo es elevado presenta muchas ventajas, por ejemplo la reducción en el tiempo de la digitalización de la información, la precisión de los datos obtenidos y la capacidad de almacenar fotografías.

	Impresa	Aplicación móvil
Tiempo en digitalizar la Información	Muy Alta	Ninguna
Costos extras en Recursos Humanos para digitalizar información	Alta	Ninguna
Costo en materiales o equipo para realizar las encuestas	Muy Baja	Alta
Precisión de información	Mediana	Muy Alta
Capacidad para tomar fotografías	Ninguna	Muy Alta
Capacidad para tomar Georeferenciación	Ninguna	Muy Alta

Tabla 8: Comparativa entre realizar encuestas en papel contra realizarlas mediante una aplicación móvil

El cuestionario desarrollado para levantar la encuesta contenía las siguientes preguntas

- ¿Cuántas vacas tiene en total?
- Del total de sus vacas, ¿Cuántas vacas son sólo para la producción de leche?
- Del total de sus vacas, ¿Cuántas vacas son para la cría de becerros y ordeña? (Doble propósito)
- Del total de sus vacas, ¿Cuántas vacas secas tiene?
- Del total de sus vacas (Doble propósito y producción de leche), ¿Cuánta leche obtiene al día?
- En promedio, ¿Cuántas vacas ordeña por día?
- En promedio, ¿Cuántos litros diarios de leche obtiene por vaca?
- En un día, ¿Cuántas veces ordeña sus vacas?
- ¿Vende leche a LICONSA a través de un Centro de Acopio?
- Observaciones

3.1 Diagrama UML de Casos de Uso

Habiendo conocido el alcance de la herramienta móvil a desarrollar, se diseñó el sistema en base a dos diagramas UML (Lenguaje Unificado de Modelado). El UML se utiliza para especificar, visualizar, construir y documentar funciones de un sistema de software. El UML pretende unificar la experiencia de técnicas de modelado e incorporar las mejores prácticas para la documentació de software. Cabe mencionar que el UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo de software y dar apoyo a la mayoría de los procesos de desarrollo orientado a objetos [4].

El primer diagrama que se desarrolló fue el diagrama de casos de uso para visualizar los actores participantes del sistema el cual se muestra en la *Figura 9* y un segundo diagrama de clases para establecer las clases que se usaron al desarrollar la aplicación móvil.

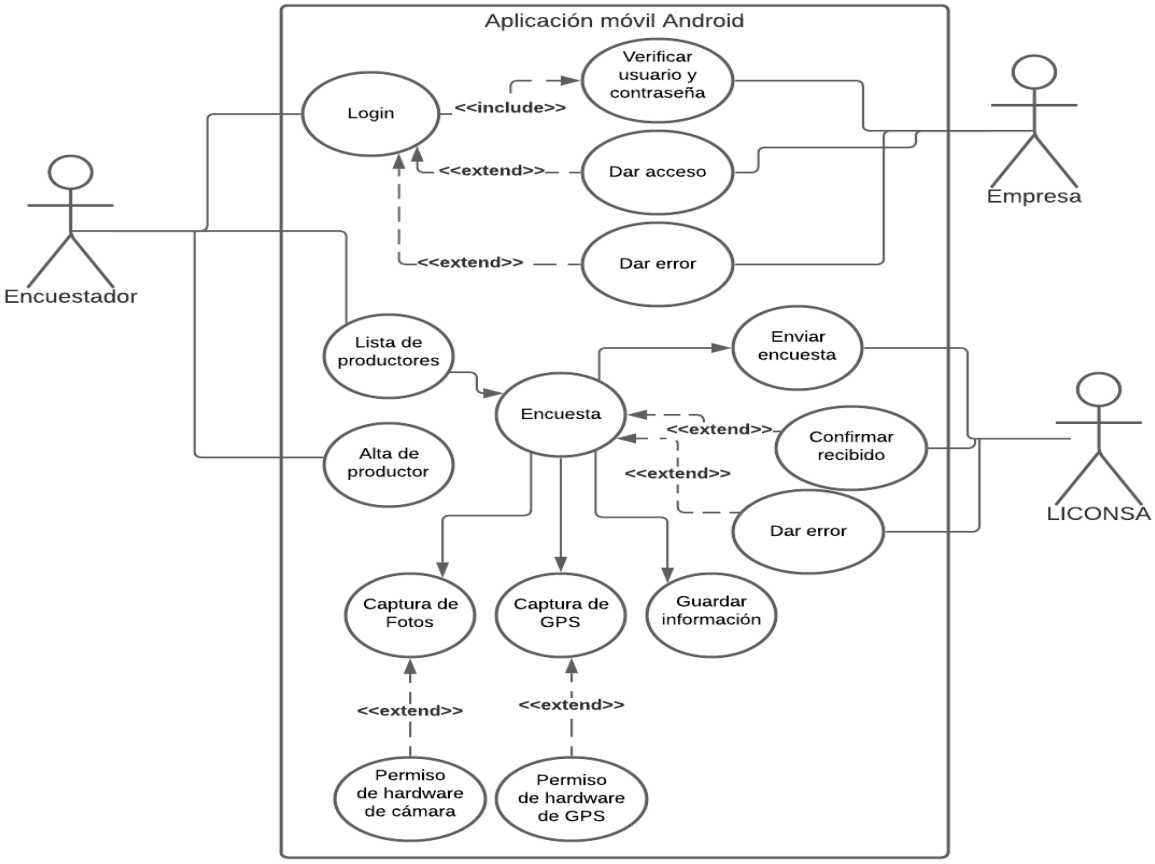


Figura 9: Diagrama de Casos de Uso del sistema de la aplicación móvil Android para realizar encuestas

3.2 Diagrama de Clases

Habiendo establecido los actores participantes en el diagrama de Casos de Uso, se realizó un segundo diseño orientado a las clases que dominarían en el sistema, siendo las clases las que se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje. Cada clase es un modelo que define un conjunto de variables y métodos apropiados para operar con dichos datos. Este segundo diagrama conocido como Diagrama de Clases el cual se muestra en la *Figura 10*, se utilizó para desarrollar las clases de la aplicación móvil para realizar encuestas.

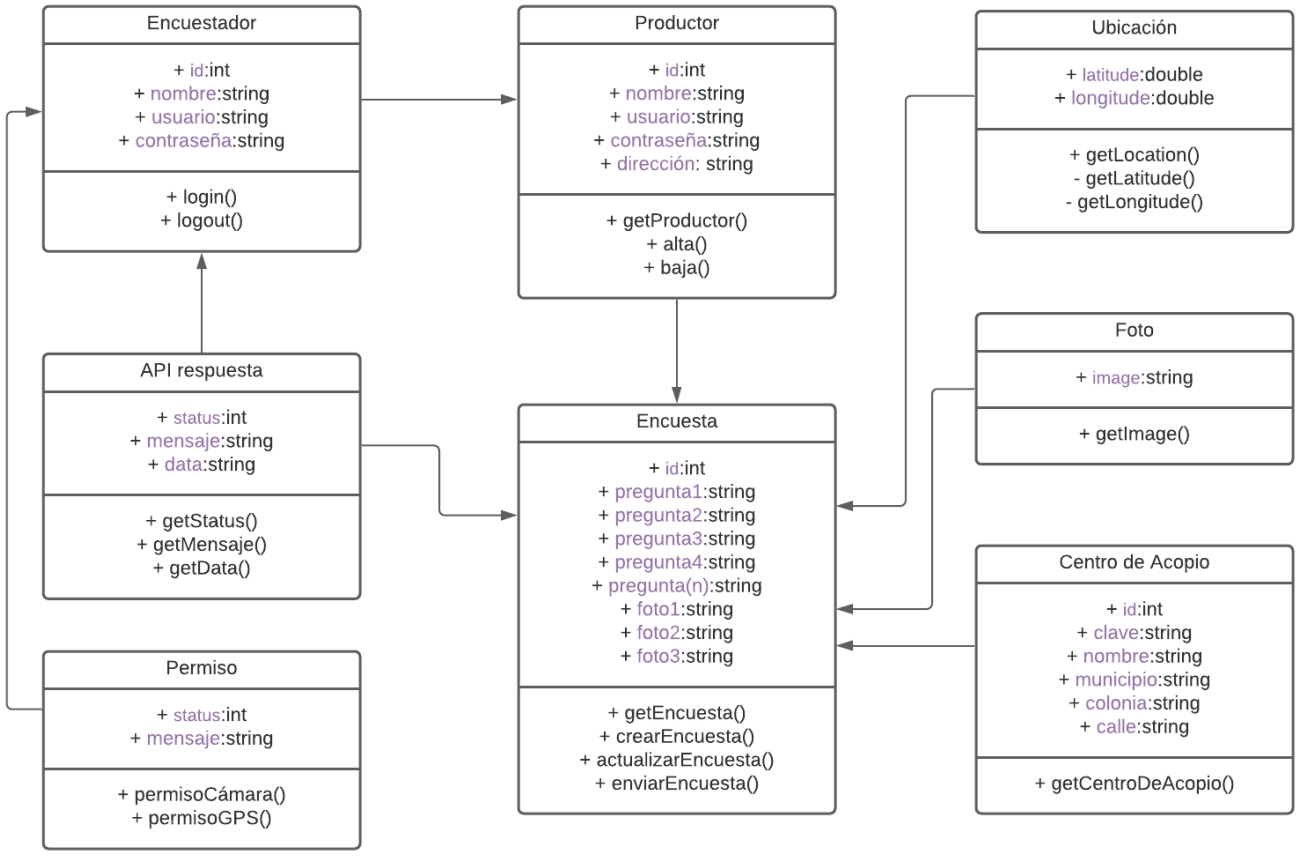


Figura 10: Diagrama de Clases del sistema de la aplicación móvil Android para realizar encuestas

2.3 Diseño de interfaz gráfica de usuario

A continuación se describe el desarrollo de las interfaces gráficas que se utilizaron siendo estas las vistas de *Login*, Alta de productor, Listado de encuestas y Levantamiento de encuesta.

3.3.1 *Login*

Primera pantalla donde el encuestador puede acceder a la aplicación, este se autentifica por medio de un usuario y contraseña. Una vez dentro de la aplicación, el encuestador tiene acceso a la lista de productores a encuestar, esta lista se encuentra vacía en un inicio y a medida que visita a los productores de leche los va dando de alta en la aplicación.

El procedimiento para crear el diseño de la interfaz gráfica, se puede realizar de dos maneras:

a) Declarando los elementos de la interfaz gráfica en XML(*Extensible Markup Language*), XML es una adaptación del "*Standard Generalized Markup*

Language"; el cual es un lenguaje que permite la organización y el etiquetado de documentos.

Ya que un diseño define la estructura de una interfaz de usuario, Android utiliza el lenguaje XML para declarar los elementos que formarán parte de la interfaz de usuario, y da esta alternativa proporcionando un vocabulario XML simple que coincide con las clases y subclases de las vistas.

Adicionalmene, para facilitar la adopción del lenguaje XML, también se puede utilizar la intefaz gráfica de Android Studio para crear el diseño XML.

b) Creando una instancia de elementos de diseño durante el tiempo de ejecución. La aplicación puede crear objetos de las vistas y manipular sus propiedades de forma programática.

En este proyecto se optó por la primera opción que es la declaración de los elementos en el archivo XML, ya que declarar la interfaz gráfica en XML permite separar la presentación de la aplicación del código que controla su comportamiento. El uso de archivos XML también facilita la creación de diferentes diseños para diferentes tamaños de pantalla y orientaciones.

Android también ofrece la flexibilidad de usar uno de estos métodos o ambos para crear la interfaz gráfica de la aplicación. Por ejemplo, permite declarar los diseños predeterminados de la aplicación en XML y, luego, modificar el diseño durante el tiempo de ejecución [5].

Al usar vocabulario XML de Android, se pueden crear rápidamente diseños de interfaz gráfica y de los elementos de pantalla que se contienen, de la misma manera que se crean páginas web en HTML(HTML es un lenguaje de marcado que se utiliza para el desarrollo de páginas de Internet. Se trata de la siglas que corresponden a *HyperText Markup Language*), con una serie de elementos anidados.

Cada archivo de diseño debe contener exactamente un elemento raíz. Una vez que se define el elemento raíz, se pueden agregar objetos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que define el diseño.

En el diagrama 1, se muestra el Diagrama a bloques de la aplicación móvil, en el cual se observa como entrada un sistema de *login* y como salida la conexión con los servidores de Liconsa. Internamente en la aplicación móvil se muestra la lista de productores, alta de los mismo así como también la captura de la encuesta y toma de evidencias.

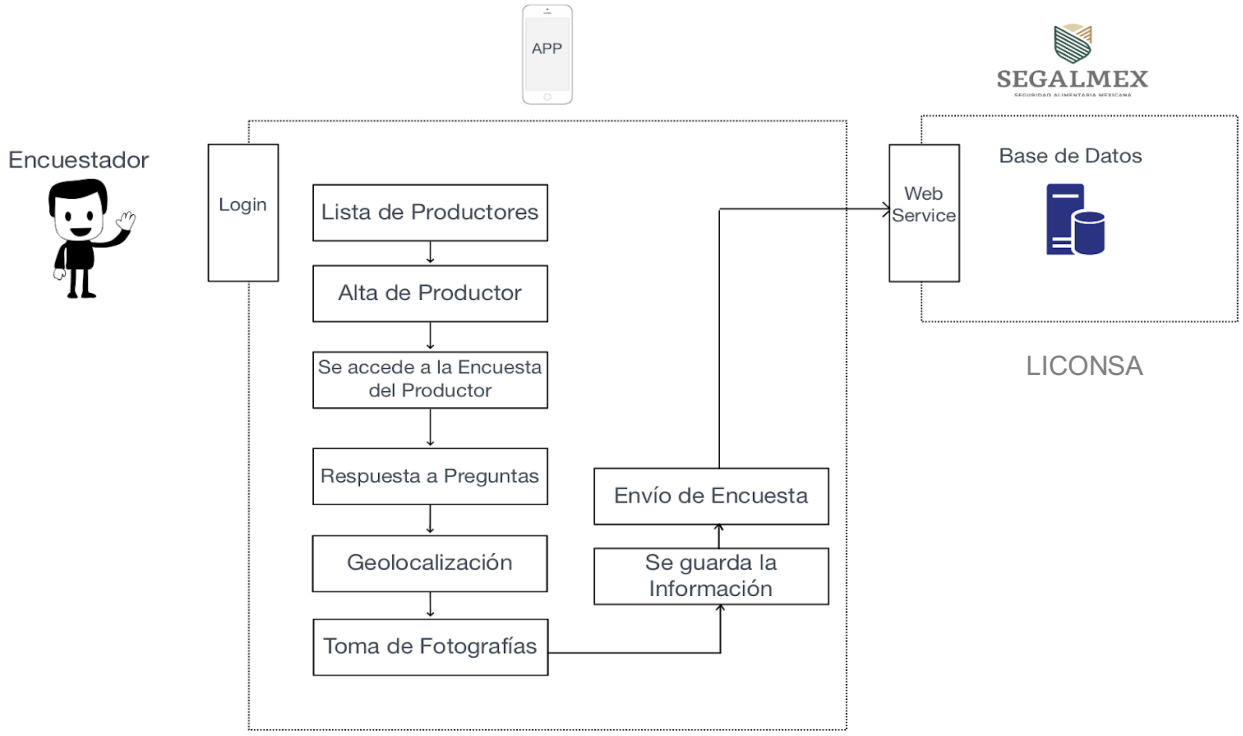


Figura 11: Diagrama de bloques del sistema que se desarrolló

Para la vista del sistema de *login*, se generó un layout (esquema para visualizar la distribución de los elementos y formas dentro de un diseño), nombrado *activity_login.xml*, el nombre del layout hace referencia al *login activity* (*activity* proviene del inglés que literalmente significa actividad, un *activity* es cada una de las pantallas o vistas que forman una aplicación), utilizando la extensión xml.

RelativeLayout es un grupo de imágenes que muestra vistas en posiciones relativas. La posición de cada vista puede especificarse como relativa a elementos equivalentes (como a la izquierda o por debajo de otra vista) o en posiciones relativas al área *RelativeLayout* superior (como alineada a la parte inferior, izquierda o central).

RelativeLayout permite que vistas secundarias especifiquen su posición relativa a la vista superior o entre sí (especificada por ID). De esta manera, se pueden alinear dos elementos por el borde derecho o hacer que uno esté por debajo del otro, en el centro de la pantalla, en el centro a la izquierda, y así sucesivamente.

De manera predeterminada, todas las vistas secundarias se dibujan en la esquina superior izquierda del diseño, por lo que se debe definir la posición de cada vista utilizando las diversas propiedades de diseño disponibles en *RelativeLayout*.

El valor de cada propiedad de diseño es un valor booleano que habilita una posición de diseño relativa al elemento *RelativeLayout* principal o un ID que hace referencia a otra vista en el diseño en el que se debe posicionar la vista.

Entre algunas de las muchas propiedades de diseño disponibles para las vistas de un grupo *RelativeLayout*, se incluyen las siguientes:

android:layout_centerInParent
Si el valor es "true", el centro de esta vista coincidirá con el centro del elemento superior.

android:layout_centerVertical
Si el valor es "true", centra este elemento secundario en posición vertical dentro de su elemento superior.

android:layout_below
Posiciona el borde superior de esta vista debajo de la vista especificada con un ID de recurso.

android:layout_toRightOf
Posiciona el borde izquierdo de esta vista a la derecha de la vista especificada con un ID de recurso.

Dentro del *Relative Layout*, se tienen dos elementos posicionados relativamente, uno que es el *LinearLayout* mostrado en la *Figura 12* y otro que es *ProgressBar* mostrado en la *Figura 13*.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:gravity="center"
    android:orientation="vertical">
```

Figura 12: Ejemplo de código LinearLayout

```
<ProgressBar
    android:id="@+id/login_progress_bar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:visibility="gone"/>
```

Figura 13: Ejemplo de código ProgressBar

LinearLayout es un grupo de vista que alinea todos los campos secundarios en una única dirección, de manera vertical u horizontal. Pudiéndose especificar la dirección del diseño con el atributo android:orientation.

Todos los campos secundarios de un *LinearLayout* se apilan uno detrás de otro, por lo cual una lista vertical solo tendrá un campo secundario por fila, independientemente del ancho que tengan, y una lista horizontal solo tendrá la altura de una fila (la altura del campo secundario más alto, más el relleno). Un *LinearLayout* respeta los márgenes entre los campos secundarios y la gravedad (alineación a la derecha, centrada o a la izquierda) de cada campo secundario.

LinearLayout también es compatible con la asignación de un volumen a campos secundarios individuales con el atributo android:layout_weight. Este atributo asigna un valor de "importancia" a una vista en términos de la cantidad de espacio que debe ocupar en la pantalla. Un valor de volumen más grande posibilita la expansión para llenar el espacio restante en la vista primaria. Las vistas secundarias pueden especificar un valor de volumen y, luego, todo espacio restante en la vista del grupo se asigna a los campos secundarios según la proporción de su volumen declarado. El volumen predeterminado es cero.

Por ejemplo, para crear un diseño lineal en el que cada campo secundario use la misma cantidad de espacio en la pantalla, se define el android:layout_height de cada vista en "0dp" (para un diseño vertical) o el android:layout_width de cada vista en "0dp" (para un diseño horizontal). Luego, se fija el android:layout_weight de cada vista en "1".

También se pueden crear diseños lineales en los que los elementos secundarios utilicen diferentes cantidades de espacio en la pantalla:

a) Si hay tres campos de texto y dos de ellos declaran un volumen igual a 1 mientras al restante no se le asigna volumen, el tercer campo de texto sin volumen no se expandirá. En cambio, solo ocupará el área que requiera su contenido. Los otros dos campos de texto, por otro lado, se expandirán de manera equitativa a fin de llenar el espacio restante después de que se midan los tres campos.

b) Si hay tres campos de texto y dos de ellos declaran un volumen de 1 mientras al tercer campo se le asigna un volumen de 2 (en lugar de 0), entonces ahora se declara más importante que los otros dos, por lo que obtiene la mitad del espacio total restante, mientras que los dos primeros comparten el resto por igual.

Para indicar el progreso de una operación, se utiliza el elemento *ProgressBar* que es un elemento en movimiento dentro de la interfaz de usuario para indicar que se está procesando la información. La barra de progreso admite dos modos para representar el progreso: determinado e indeterminado.

Se utiliza el modo indeterminado cuando no se sabe cuánto tiempo llevará una operación. El modo indeterminado es el valor predeterminado para la barra de

progreso y muestra una animación cíclica sin una cantidad específica de progreso indicada.

El modo determinado se utiliza cuando se desea mostrar que se ha producido una cantidad específica de progreso. Por ejemplo, el porcentaje restante de un archivo que se está recuperando, la cantidad de registros en un lote escrito en la base de datos o el porcentaje restante de un archivo de audio que se está reproduciendo.

Para indicar un progreso determinado, se establece el estilo de la barra de progreso en R.style.Widget_ProgressBar_Horizontal y se establece la cantidad de progreso.

En nuestro caso utilizamos el la propiedad predeterminada del *ProgressBar* que es el modo indeterminado, ya que lo utilizaremos para notificarle al usuario que se está procesando su acceso al sistema.

También utilizamos el elemento *ImageView* mostrado en la *Figura 14* el cual sirve para posicionar el logotipo de la empresa en el centro de la pantalla.

```
<ImageView
    android:layout_width="150dp"
    android:layout_height="150dp"
    android:layout_gravity="center_horizontal"
    android:src="@drawable/logo"/>
```

Figura 14: Ejemplo de código ImageView

Cuando se necesitan agregar imágenes a la interfaz gráfica, por ejemplo para mostrar imágenes estáticas en la aplicación, se puede usar la clase *Drawable* y sus subclases para dibujar imágenes y formas, cabe mencionar que una imagen estática es una imagen que no cambia en el tiempo, como analogía se tiene una constante a diferencia de una variable. Una clase *Drawable* es una abstracción general que representa algo que se puede dibujar. Las diversas subclases ayudan en situaciones específicas de imágenes, y se pueden extender para definir los objetos propios de elementos de diseño que tienen comportamientos únicos.

También se puede preferir usar SVG (del inglés *Scalable Vector Graphics*), que es un elemento de diseño vectorial, que define una imagen con un conjunto de puntos, líneas y curvas, junto con la información de color asociada. Esto permite que se ajusten los elementos de diseño vectoriales a diferentes tamaños sin que se pierda la calidad.

En nuestro caso hacemos referencia a un archivo de imagen desde los recursos del proyecto para agregar gráficos a la aplicación. Los tipos de archivo admitidos son PNG (preferido), JPG (aceptable) y GIF (desaconsejado). Los íconos de las apps, logotipos y otros gráficos, son ideales para esta técnica.

Para usar un recurso de imagen, agregamos el archivo al directorio res/drawable/ del proyecto. Y se hace referencia al recurso de imagen desde el código o el diseño XML. De cualquier modo, se hace referencia a él con un ID de recurso.

Los recursos de imagen que se colocan en el directorio res/drawable/ se pueden optimizar automáticamente con la compresión de imágenes sin pérdidas mediante la herramienta AAPT(*Android Asset Package Tool*) durante el proceso de compilación. Por ejemplo, un archivo PNG con color verdadero que no requiera más de 256 colores se puede convertir a un archivo PNG de 8 bits con una paleta de colores. De ese modo, se genera una imagen que tiene la misma calidad, pero que requiere menos memoria. Como resultado, los archivos binarios de imágenes que se colocan en este directorio pueden cambiar en el momento de la compilación. Si se planea leer una imagen como un flujo de bits para convertirla en un mapa de bits, se colocan las imágenes en la carpeta res/raw/, donde la herramienta AAPT no las modifica [6].

También se utilizan en este proyecto dos campos para que el encuestador escriba su usu.ario y contraseña, para ello se utilizan dos elementos *EditText* mostrados en la *Figura 15* y en la *Figura 16* respectivamente.

```
<!-- Encuestador -->
<EditText
    android:id="@+id/login_encuestador_txt"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:maxLines="1"
    android:inputType="textCapCharacters"
    android:textAlignment="center"
    android:layout_marginTop="5dp"
    android:hint="Encuestador" />
```

Figura 15: Ejemplo de código EditText de tipo texto

```
<!-- Version -->
<EditText
    android:id="@+id/login_clave_txt"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:maxLines="1"
    android:inputType="number"
    android:textAlignment="center"
    android:layout_marginTop="5dp"
    android:hint="XXXXXXX" />
```

Figura 16: Ejemplo de código EditText de tipo número

EditText es un elemento de interfaz de usuario para ingresar y modificar texto. Cuando se define un elemento de edición de texto, debe especificar el atributo *inputType*.

La elección del tipo de entrada configura el tipo de teclado que se muestra, los caracteres aceptables y la apariencia del texto de edición. Por ejemplo, si desea aceptar un número secreto, como un pin único o un número de serie, puede

establecer *inputType* en "numericPassword". Un *inputType* de "numericPassword" da como resultado una edición de texto que solo acepta números, muestra un teclado numérico cuando está enfocado y enmascara el texto que se ingresa para mayor privacidad.

También puede recibir devoluciones de llamada cuando un usuario cambie el texto agregando un *TextWatcher* al texto de edición. Esto es útil cuando desea agregar la funcionalidad de guardado automático a medida que se realizan cambios, o validar el formato de entrada del usuario.

En nuestro caso seleccionamos el valor de *textCapCharacters* y *number*, debido a que necesitábamos el nombre de usuario en mayúsculas y como contraseña solo números. De esta manera el teclado que se mostraba permitía mayúsculas por default y en la contraseña solo números.

Para finalizar el diseño de esta vista de Login, se agregó un elemento *Button*, el cuál manejó la acción de enviar el acceso del usuario para autentificación, este elemento se puede ver en la *Figura 17*.

```
<Button
    android:id="@+id/login_encuestador_btn"
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="Iniciar" />
```

Figura 17: Ejemplo de código Button

Un ***Button*** es un elemento de interfaz de usuario, que el usuario puede tocar o hacer clic para realizar una acción.

Cada botón está diseñado para utilizar el fondo de botón predeterminado del sistema, que a menudo es diferente de una versión de la plataforma a otra. Mejorar esta visualización de botón con un estilo coloreado de fondo.

Para personalizar botones individuales con un fondo diferente, se especifica el atributo android: background con un recurso dibujable o de color. Alternativamente, se puede aplicar un estilo para el botón, que funciona de manera similar a los estilos HTML para definir múltiples propiedades de estilo, como el fondo, la fuente y el tamaño.

También se puede optar por un diseño de "sin bordes". Los botones sin bordes se parecen a los botones básicos, excepto que no tienen bordes ni fondo, pero aún cambian de apariencia durante diferentes estados, como cuando se hace clic.

Para redefinir a fondo la apariencia de un botón, se puede especificar un fondo personalizado. Sin embargo, en lugar de proporcionar un mapa de bits o color

simple, su fondo debe ser un recurso de lista de estado que cambie la apariencia según el estado actual del botón.

Se puede definir la lista de estados en un archivo XML que defina tres imágenes o colores diferentes para usar en los diferentes estados de los botones.

En nuestro caso un estilo predefinido para el botón fue suficiente. Y para especificar la acción de cuando se presiona el botón, se configuró un detector de clics en el objeto del botón en el código del *ActivityLogin*, como se muestra en la *Figura 18*.

Antes de autentificar al usuario se realizó una verificación que consiste en revisar la existencia del nombre del encuestador y su contraseña/clave. Posteriormente se envia la información al servidor. Mientras el usuario espera su autentificación, el *ProgressBar* se habilita visualmente para informar al usuario que se esta procesando su solicitud de autentificación.

```
login_encuestador_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        encuestador = login_encuestador_txt.getText().toString();
        clave = login_clave_txt.getText().toString();

        if ("".equals(encuestador)) {
            login_encuestador_txt.setError("Falta escribir encuestador");
            login_encuestador_txt.requestFocus();
            return;
        }
        if ("".equals(clave)) {
            login_clave_txt.setError("Falta escribir la clave");
            login_clave_txt.requestFocus();
            return;
        }

        login_progress_bar.setVisibility(View.VISIBLE);
        Map<String, String> params = new HashMap<~>();
        params.put( k: "name", encuestador);
        params.put( k: "pass", clave);
        MyApi.customPost(mContext,  url: Constants.BACKEND_URL+"/api/intelli/croix_login"
    }
});
```

Figura 18: Ejemplo de código de la acción del botón login

En la *Figura 19* se muestra la vista final de *Login* con los elementos del Logotipo de la empresa, los campos para ingresar nombre de usuario/contraseña y por último el botón de Iniciar, el cuál enviará la información para autentificar al usuario.

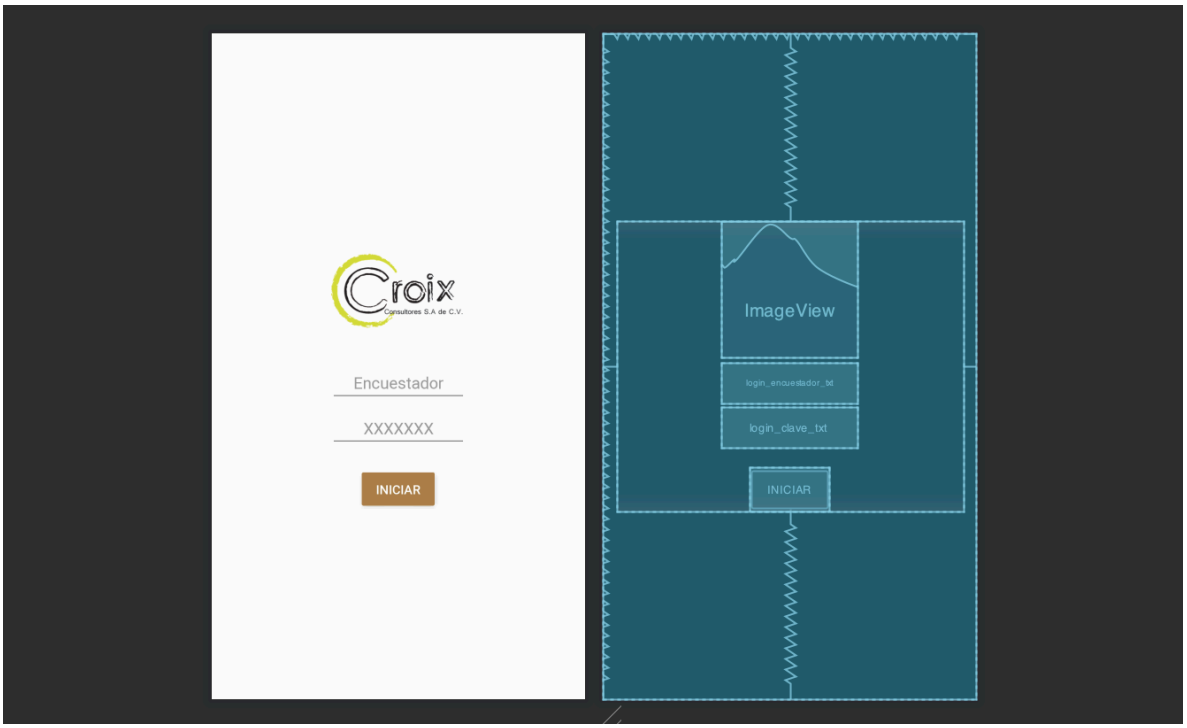


Figura 19: Pantalla de login

En la *Figura 20*, se observa la lógica que se realiza en caso de que el usuario fuera autentificado, dirigiendo al usuario a la vista principal “Main Activity”, en caso contrario se notifica al usuario de su acceso incorrecto.

```
if(apiResponse.getCode()==200){
    editor = sharedPrefs.edit();
    editor.putString(Constants.PREF_ENCUESTADOR, encuestador).commit();
    goToMain();
}else{
    try{
        Handler handler = new Handler(Looper.getMainLooper());
        handler.post(new Runnable() {
            @Override
            public void run() {
                login_progress_bar.setVisibility(View.GONE);
                if(apiResponse.getCode()==402){
                    Log.i(TAG,  msg: "zz contraseña incorrecta");
                    Toast.makeText(mContext,  text: "Contraseña incorrecta", Toast.LENGTH_SHORT).show();
                }else{
                    Log.i(TAG,  msg: "zz error 1b");
                    Toast.makeText(mContext,  text: "Error Code 1b", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

Figura 20 Ejemplo de código de acceso al usuario

3.3.2 Alta de productor

Para la vista de alta de productor, como se muestra en la *Figura 21* se utiliza un *LinearLayout* con orientación vertical sin ningún elemento con posición relativa.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="15dp"
    android:orientation="vertical"
    tools:context=".SurveyAddActivity">
```

Figura 21 Ejemplo de código de LinearLayout con orientación vertical

Se agregan elementos de tipo *TextView* y *EditText* para solicitar la información del productor a dar de alta, en la *Figura 22* se observa el código del *layout* para solicitar el nombre, en la *Figura 23* el código de *layout* para solicitar el apellido paterno y en la *Figura 24* el código de *layout* para solicitar el apellido materno.

```
    <!-- NOMBRE -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="NOMBRE DEL PRODUCTOR"/>
    <EditText
        android:id="@+id/edit_Nombrep"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:maxLines="1"
        android:inputType="textCapCharacters"
        android:hint=""/>
```

Figura 22 Ejemplo de código TextView y EditText para solicitar nombre

```
    <!-- APELLIDO PATERNO -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:text="APELLIDO PATERNO DEL PRODUCTOR"/>
    <EditText
        android:id="@+id/edit_Apelidopp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:maxLines="1"
        android:inputType="textCapCharacters"
        android:hint=""/>
```

Figura 23: Ejemplo de código TextView y EditText para solicitar apellido materno

```
<!-- APELLIDO MATERNO -->
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    android:text="APELLIDO MATERNO DEL PRODUCTOR"/>
<EditText
    android:id="@+id/edit_Apellidomp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:maxLines="1"
    android:inputType="textCapCharacters"
    android:hint=""/>
```

Figura 24: Ejemplo de código TextView y EditText para solicitar apellido materno

El elemento *TextView* muestra texto al usuario y de manera opcional, permite editarlo. Un *TextView* es un editor de texto completo, sin embargo, de inicio la opción para editar texto viene configurada para no permitir la edición a menos que se especifique.

En la *Figura 25* se observa el código de la vista con los botones de “Cancelar” y “Aceptar” donde tienen la función de cancelar la captura del productor y dar de alta al productor, respectivamente.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginTop="15dp"
    android:gravity="center">
    <Button
        android:id="@+id/btnCancelarNuevoProductor"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dp"
        android:text="Cancelar"
        style="@style/Widget.AppCompat.Button.Colored"/>
    <Button
        android:id="@+id/btnAgregarProductor"
        style="@style/Widget.AppCompat.Button.Colored"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Guardar"/>
</LinearLayout>
```

Figura 25 Ejemplo de código de dos Button's para aceptar y cancelar respectivamente

Antes de dar de alta al nuevo productor, se verifica que los tres campos “Nombre, apellido paterno y apellido materno ” sean proporcionados, como se muestra en la *Figura 26*, en caso contrario, se le notifica al encuestador que esta información esta incompleta.

```
if ("".equals(string_edit_Nombrep)) {
    edit_Nombrep.setError("Escribe el nombre del Productor");
    edit_Nombrep.requestFocus();
    return;
}
if ("".equals(string_edit_Apelidopp)) {
    edit_Apelidopp.setError("Escribe el apellido paterno del Productor");
    edit_Apelidopp.requestFocus();
    return;
}
if ("".equals(string_edit_Apellidomp)) {
    edit_Apellidomp.setError("Escribe el apellido materno del Productor");
    edit_Apellidomp.requestFocus();
    return;
}
```

Figura 26 Ejemplo de código con validadores de texto

En la *Figura 27*, se muestran todos los elementos del “Alta de productor”.

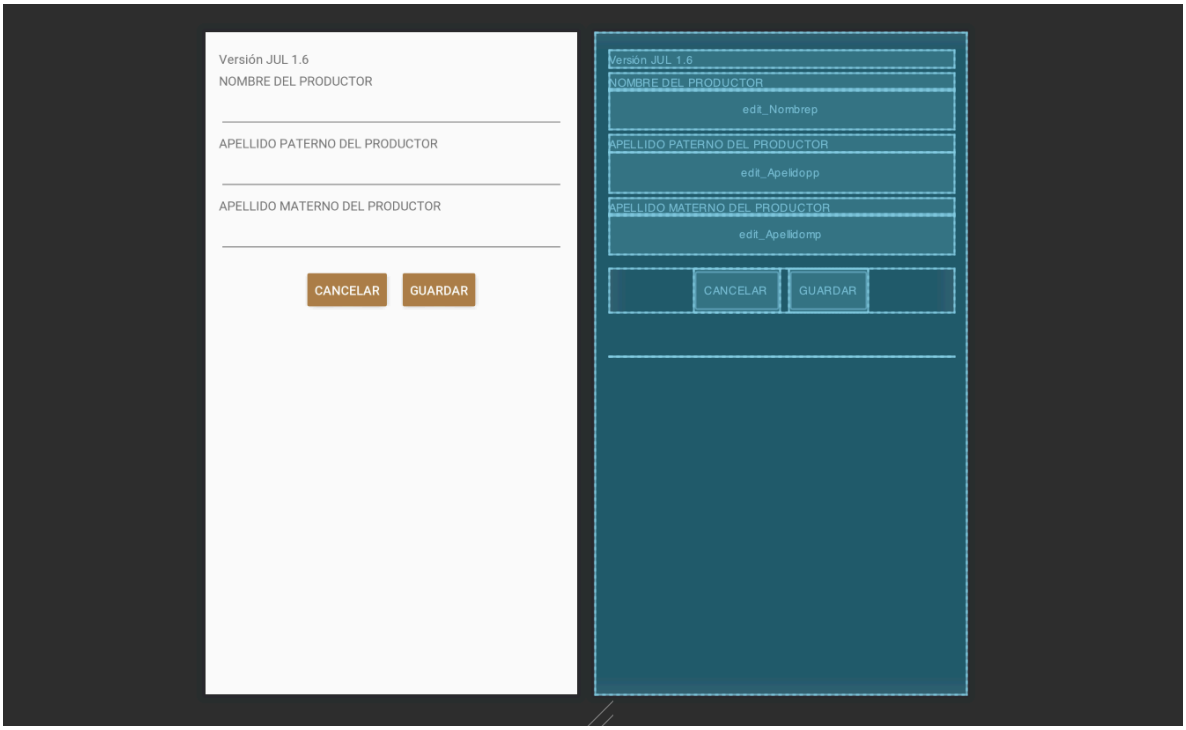


Figura 27 Pantalla de alta de productor

3.3.3 Listado de encuestas

Esta vista integra el elemento *RecyclerView* que almacena el listado de encuestas, este elemento es declarado en *MainActivity* el cual es el *activity* principal de la aplicación.

Un *Activity* es un componente clave de una aplicación para Android, y la forma en que se inician y se crean las actividades es una parte fundamental del modelo de aplicación de la plataforma. A diferencia de los paradigmas de programación en los que las apps se inician con un método *main*(), el sistema Android inicia el código en una instancia de *Activity* invocando métodos de devolución de llamada específicos que corresponden a etapas específicas de su ciclo de vida.

La experiencia con una aplicación para dispositivos móviles difiere de la versión de escritorio, ya que la interacción del usuario con la aplicación no siempre comienza en el mismo lugar. En este caso, no hay un lugar específico desde donde el usuario comienza su actividad. Dependiendo de dónde se abra una aplicación, es el lugar de destino de la misma. Si un usuario no ha iniciado sesión, entonces la aplicación inicia en la pantalla de login, pero si un usuario ya ha iniciado sesión previamente, entonces la pantalla donde iniciará no será la de *login*, en tal caso el usuario iniciará directamente en la pantalla principal.

El componente *Activity* está diseñado para facilitar este paradigma, de esta manera cada *Activity* puede invocar a otro *Activity* dentro o fuera de la aplicación. El *Activity* sirve como el punto de entrada para la interacción de una aplicación con el usuario.

Un *Activity* proporciona la ventana en la que la aplicación dibuja su interfaz gráfica. Por lo general, esta ventana llena la pantalla, pero puede ser más pequeña y flotar sobre otras ventanas. Generalmente, un *Activity* implementa una pantalla en una aplicación [7].

La mayoría de las aplicaciones contienen varias pantallas, lo cual significa que incluyen varios *Activities*. Por lo general, un *Activity* en una aplicación, se especifica como el *Activity* principal (*MainActivity*), que es la primera pantalla que aparece cuando el usuario inicia la app. Luego, cada actividad puede iniciar otra actividad a fin de realizar diferentes acciones. Por ejemplo, en nuestro caso la primera pantalla es la vista de Login y una vez que el usuario inicia sesión su pantalla principal será la otorgada por el *MainActivity* donde el encuestador verá la lista de encuestas y cada encuesta perteneciente a un productor de leche. A partir de aquí, desde el *Activity* principal se puede dar de alta nuevos productores de leche o bien editar alguna encuesta de un productor de leche.

Si bien los *Activities* trabajan en conjunto a fin de crear una experiencia del usuario coherente en la aplicación, cada *Activity* se relaciona con otros *Activities*; es por esto que al usar activities en una aplicación, se debe registrar información sobre estas en el manifiesto de la aplicación (*AndroidManifest.xml*) y administrar los ciclos de vida de las activities de manera apropiada.

Para que esta aplicación pudiera usar las pantallas antes mencionadas, fueron declararlas en el manifiesto junto con las propiedades del *intent* y filtros de *intent*, como se muestra en la *Figura 28*.

```
<activity
    android:name="com.caydeem.croixjul.LoginActivity"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name="com.caydeem.croixjul.MainActivity"
    android:screenOrientation="portrait"/>
<activity
    android:name="com.caydeem.croixjul.SurveyEditActivity"
    android:screenOrientation="portrait"/>
<activity
    android:name="com.caydeem.croixjul.SurveyAddActivity"
    android:screenOrientation="portrait"/>
```

Figura 28: Ejemplo de código del Manifiesto

Una ***Intent*** es un objeto de mensajería que se usa para solicitar una acción de otro componente de una aplicación. Si bien los *intents* facilitan la comunicación entre componentes de varias formas, existen tres casos de uso principales:

a) Iniciar una actividad. Un *Activity* representa una única pantalla en una aplicación. Se puede iniciar una nueva instancia de un *Activity* pasando un *Intent* a *startActivity*(). El Intent describe el *Activity* que se debe iniciar y contiene los datos necesarios para ello.

   Si se desea recibir un resultado de la actividad cuando finalice, se llama a *startActivityForResult*(). El *activity* recibe el resultado como un objeto *Intent* separado en la devolución de llamada de *onActivityResult*() del *Activity*.

b) Iniciar un servicio. Un *Service* o Servicio, es un componente que realiza operaciones en segundo plano sin una interfaz de usuario. Se puede iniciar un servicio usando métodos de la clase *Service*. Se inicia un servicio para realizar una operación única pasando un Intent a *startService*(). El Intent describe el servicio que se debe iniciar y contiene los datos necesarios para ello.

   Si el servicio está diseñado con una interfaz “cliente-servidor”, siendo el “cliente” el *software* que accederá a los servicios proporcionados por el servidor, se puede establecer un enlace con el servicio de otro componente pasando una Intent a *bindService*().

c) Transmitir una emisión. Una emisión es un aviso que cualquier aplicación puede recibir. El sistema transmite varias emisiones de eventos, como cuando se inicia el sistema o comienza a cargarse el dispositivo. Se puede transmitir una emisión a otras aplicaciones pasando un *Intent* a *sendBroadcast*() o *sendOrderedBroadcast*().

Respecto a los filtros de *intents* son una función muy útil de la plataforma de Android. Estos proporcionan la capacidad de iniciar una actividad tanto en una solicitud explícita, como también de una implícita. En esta aplicación se especifica una solicitud explícita del Login para indicar al sistema que debe iniciar este *activity* al ejecutar la aplicación mediante el action *MAIN* con la propiedad *LAUNCHER*. Para hacerlo posible se declaró un atributo <*intent-filter*> en el elemento <activity>. La definición de este elemento incluye un elemento <*action*> y, de manera opcional, un elemento <*category*>. Estos elementos se combinaron para especificar el tipo de *intent* al que puede responder la actividad [8].

En la *Figura 29*, se muestra el elemento *RecyclerView*, el cual es una versión avanzada para mostrar listas. En el *RecyclerView* existen varios componentes diferentes que trabajan juntos para mostrar los datos. El contenedor general de la interfaz de usuario es un objeto *RecyclerView* que se agrega al diseño. El objeto *RecyclerView* se completa por sí solo con vistas que brinda el administrador de diseño de proporciones. Se puede usar uno de los administradores de diseño estándar (como *LinearLayoutManager* o *GridLayoutManager*), o bien implementar uno personalizado [9].

```
<!-- Survey List -->
<LinearLayout
    android:id="@+id/main_survey_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="visible"
    android:orientation="vertical">
        <android.support.v7.widget.RecyclerView
            android:id="@+id/recyclerView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
</LinearLayout>
```

Figura 29: Ejemplo de código RecyclerView

Las vistas incluidas en la lista están representadas por objetos contenedores de vistas. Esos objetos son instancias de una clase que se define extendiendo el objeto *ViewHolder* de *RecyclerView*. Cada objeto contenedor de vistas es responsable de mostrar un elemento individual con una vista.

El objeto *RecyclerView* crea solamente la cantidad de objetos contenedores de vistas que sean necesarios para mostrar la parte en pantalla del contenido dinámico, más algunos adicionales. A medida que el usuario se desplaza por la lista, el objeto

RecyclerView toma las vistas fuera de pantalla y vuelve a vincularlas con los datos que se desplazan en la pantalla.

Un adaptador, que se crea extendiendo *RecyclerView.Adapter*, administra los objetos contenedores de vistas. El adaptador crea contenedores de vistas, según sea necesario, y los vincula con sus datos. Para hacerlo, asigna el contenedor de vistas a una posición y llama al método *onBindViewHolder*() del adaptador. Este método usa la posición del contenedor de vistas para determinar cuál debe ser el contenido, en función de su posición en la lista.

Este modelo de *RecyclerView* realiza gran parte del trabajo de optimización:

a) Cuando se completa por primera vez la lista, crea y vincula algunos contenedores de vistas a cualquier lado de la lista. Por ejemplo, si la vista muestra las posiciones de la lista 0 a 9, el objeto *RecyclerView* crea y vincula estos contenedores de vistas, y es posible que también cree y vincule el contenedor de vistas para la posición 10. De esta manera, si el usuario se desplaza por la vista, el siguiente elemento estará listo para mostrarse.

b) A medida que el usuario se desplaza por la vista, el objeto *RecyclerView* crea nuevos contenedores de vistas según sea necesario. También guarda los contenedores de vistas que se desplazaron fuera de la pantalla para que puedan volver a usarse. Si el usuario cambia la dirección de desplazamiento, se pueden recuperar los contenedores de vistas que se desplazaron fuera de la pantalla. Por otro lado, si el usuario continúa el desplazamiento en la misma dirección, los contenedores de vistas que salieron de la pantalla por más tiempo pueden volver a vincularse con datos nuevos. No es necesario crear el contenedor de vistas o aumentar su vista; en cambio, la app simplemente actualiza el contenido de la vista para que coincida con el elemento nuevo con el que está vinculada.

c) Cuando se cambian los elementos que se muestren, se puede notificar al adaptador y este vuelve a vincular los elementos afectados.

En la *Figura 30* se muestra como se agregaron las Bibliotecas de compatibilidad v7 al proyecto para poder utilizar el *RecyclerView*. Se suelen utilizar estas bibliotecas de compatibilidad para que las aplicaciones optimizadas para versiones recientes de android también puedan ser compatibles con versiones de android anteriores.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:recyclerview-v7:28.0.0'
    implementation 'com.android.support:appcompat-v7:28.0.0'
```

Figura 30 Ejemplo de código de dependencias

Después de agregar el *RecyclerView* al diseño, también se conecta a un administrador de diseño y se adjunta un adaptador para que se visualicen los datos como se muestra en la *Figura 31*.

```
surveyList = new ArrayList<>();
surveyAdapter = new SurveyAdapter(surveyList);
RecyclerView.LayoutManager mLayoutManager = new LinearLayoutManager(getApplicationContext());
recyclerView.setLayoutManager(mLayoutManager);
recyclerView.setItemAnimator(new DefaultItemAnimator());
recyclerView.setAdapter(surveyAdapter);
```

Figura 31: Ejemplo de código de un RecyclerView con administrador de diseño

Para ingresar todos tus datos a la lista, se extendió la clase *RecyclerView.Adapter* como se muestra en la *Figura 32*. Este objeto crea vistas para elementos y reemplaza el contenido de algunas de las vistas por nuevos elementos de datos cuando el elemento original ya no está visible.

```
public class SurveyAdapter extends RecyclerView.Adapter<SurveyAdapter.MyViewHolder> {

    private List<Survey> surveyList;

    public void setListOfSurveys(List<Survey> surveyList) { this.surveyList = surveyList; }

    public SurveyAdapter(List<Survey> surveys) { this.surveyList = surveys; }
```

Figura 32 Ejemplo de código de un RecyclerView con un adapter

En la *Figura 33*, se observa como el administrador de diseño llama al método *onCreateViewHolder*() del adaptador. Fue necesario que este método construyera un objeto *RecyclerView.ViewHolder* y configurara la vista que se usó para mostrar su contenido. El tipo del contenedor de vistas debió coincidir con el tipo declarado en la firma de la clase Adapter. Por lo general, se configura la vista con un archivo de diseño XML, en nuestro caso item_encuesta.xml. Dado que el contenedor de vistas todavía no está asignado a ningún dato en particular, el método no configura realmente el contenido de la vista.

```
@NonNull
@Override
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) {
    View filaMascota = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_encuesta
    return new MyViewHolder(filaMascota);
}
```

Figura 33 Ejemplo de código de la creación de un ViewHolder

El administrador de diseño vincula el contenedor de vistas con sus datos. Para hacerlo, llama al método *onBindViewHolder*() del adaptador como se muestra en la *Figura 34*, y pasa la posición del contenedor de vistas en *RecyclerView*. Es necesario que el método *onBindViewHolder*() busque los datos correspondientes y los usa para completar el diseño del contenedor de vistas, de esta manera el administrador de diseño vincula nuevamente cualquier contenedor de vistas afectado, lo que permite la actualización de sus datos.

```
@Override
public void onBindViewHolder(@NonNull MyViewHolder myViewHolder, int i) {
    Survey survey = surveyList.get(i);

    String cveca = survey.getCveca();
    String foliocuestionario = survey.getFoliocuestionario();
    String nombrep = survey.getNombrep();
    String apelidopp = survey.getApelidopp();
    String apellidomp = survey.getApellidomp();
    String status = survey.getStatus();

    myViewHolder.item_cveca.setText(cveca);
    myViewHolder.item_foliocuestionario.setText(foliocuestionario);
    myViewHolder.item_nombrep.setText(nombrep+" "+apelidopp+" "+apellidomp);
    if(status.equals("Enviada")){
        myViewHolder.item_status.setTextColor(Color.parseColor( colorString: "#4CAF50"));
    }else if(status.equals("Nuevo")){
        myViewHolder.item_status.setTextColor(Color.parseColor( colorString: "#2196F3"));
    }else if(status.equals("Error al enviar")){
        myViewHolder.item_status.setTextColor(Color.parseColor( colorString: "#B60505"));
    }else{
        myViewHolder.item_status.setTextColor(Color.parseColor( colorString: "#222222"));
    }
    myViewHolder.item_status.setText(status);
}
```

Figura 34 Ejemplo de código del método onBindViewHolder

Se personalizan los objetos *RecyclerView* para satisfacer las necesidades específicas. La personalización realizada fue diseñar la vista para cada contenedor de vistas y escribir el código para actualizar esas vistas con los datos correspondientes dependiendo del status de la encuesta (Enviada, nueva y error al enviar).

El objeto *RecyclerView* se usa junto con un administrador de diseño para posicionar los elementos individuales en la pantalla y determina cuándo volver a usar las vistas de elementos que ya no ve el usuario. Para volver a usar (o reciclar) una vista, un administrador de diseño puede solicitar al adaptador que reemplace el contenido de la vista por un elemento diferente del conjunto de datos. Reciclar la vista de esta forma mejora el rendimiento evitando que se creen vistas innecesarias o realizando búsquedas de *findViewById*() con un alto costo. La Biblioteca de compatibilidad de Android incluye tres administradores de diseño estándar, cada uno de los cuales ofrece muchas opciones de personalización:

a) *LinearLayoutManager* organiza los elementos en una lista unidimensional.

b) *GridLayoutManager* organiza los elementos en una cuadrícula bidimensional.

c) *StaggeredGridLayoutManager* organiza los elementos en una cuadrícula bidimensional, en la que cada columna está ligeramente desplazada en función de la anterior.

En nuestro caso utilizamos *LinearLayoutManager* para organizar los elementos en una lista unidimensional y el resultado obtenido se muestra en la *Figura 36.*

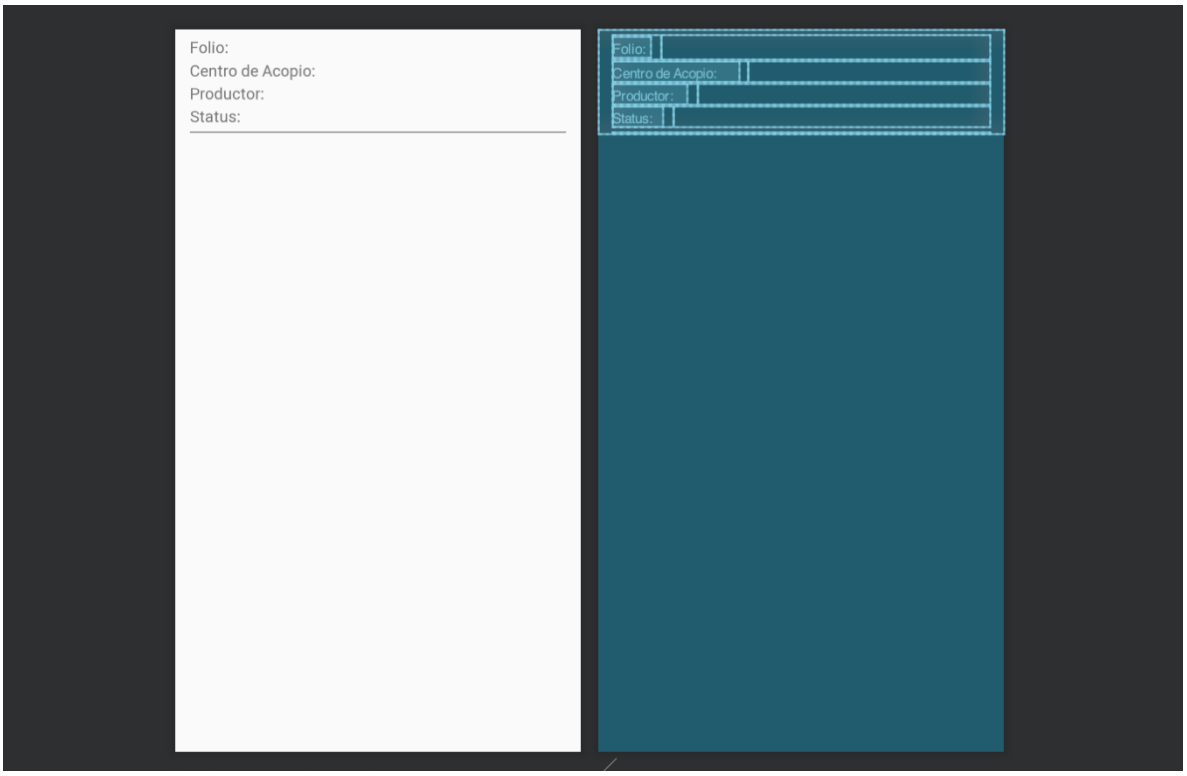


Figura 35 Pantalla de la lista de Productores

3.3.4 Levantamiento de encuesta

En la *Figura 36*, se observa el código de la vista del levantamiento de encuesta y la vista final en la *Figura 37*, donde se utilizó el elemento *ScrollView*, este elemento permite desplazar la jerarquía de vista ubicada dentro de él. La vista de desplazamiento puede tener solo un elemento directo colocado dentro de él. Para agregar varias vistas dentro de la vista de desplazamiento, se añaden al elemento secundario,y este agrega al grupo de vistas que se deseen. En la aplicación se utilizó el elemento ScrollView, dentro del *ScrollView* un elemento *LinearLayout*, y dentro del *LinearLayout* se colocaron vistas adicionales como lo fueron las etiquetas y campos de la encuesta. Es importante mencionar que el elemento *ScrollView* solo admite el desplazamiento vertical.

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:focusable="true"
        android:focusableInTouchMode="true"
        android:clickable="true"
        android:padding="20dp"
        android:orientation="vertical">
```
Figura 36: Ejemplo de código de ScrollView

.

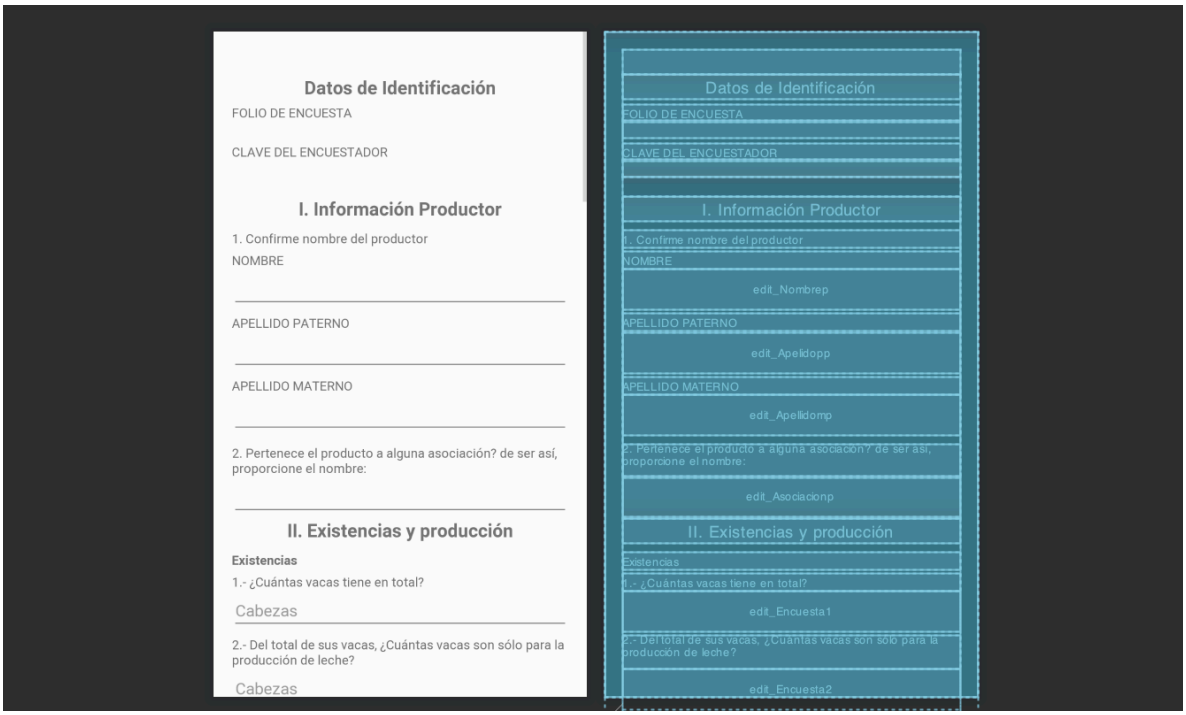


Figura 37 Pantalla de la encuesta

En la Figura 37 se muestra la sección de la aplicación donde se capturan las respuestas de los Productores de Leche, y donde también se toma la Georeferenciación del lugar y las fotografías necesarias como evidencia, el encuestador no guarda la encuesta hasta ser completada en su totalidad sin dejar preguntas sin contestar. Esta información es almacenada dentro del dispositivo una vez que ha finalizado la encuesta y permanece segura hasta ser enviada a LICONSA.

3.4 Desarrollo del servicio de Geolocalización (*GPS*)

Una de las funciones requeridas en el desarrollo de esta aplicación móvil fue la detección de ubicaciones. Los encuestadores llevaron sus dispositivos móviles a todas partes de México, por lo que el conocimiento de la ubicación permite brindar un mejor conocimiento de la población. Las *API* de ubicación están disponibles en los Servicios de Google Play los cuales permiten obtener el conocimiento de la ubicación en las aplicaciones android con seguimiento automático de la ubicación [10] como se muestra en la *Figura 38*.

Mediante las *API* de ubicación de los Servicios de Google Play, la aplicación pudo solicitar la ubicación más reciente del dispositivo del usuario. En la mayoría de los

casos, para conocer la ubicación actual del usuario, por lo general equivale a la ubicación más reciente del dispositivo.

```
public class MyLocationListener implements LocationListener {
    @Override
    public void onLocationChanged(Location location) {

        if(location!=null){
            if(isBetterLocation(location, trackingLocation)) {
                if(trackingLocation==null){
                    trackingLocation=location;
                }
```

Figura 38: Ejemplo de código de LocationListener

Para poder utilizar los servicios de ubicación se debió solicitar el permiso de ubicación *ACCESS_FINE_LOCATION,* esto se muestra en la *Figura 39*.

```
private void checkPermission() {
    if (ContextCompat.checkSelfPermission( context: this,
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        if (ActivityCompat.shouldShowRequestPermissionRationale( activity: this,
                Manifest.permission.ACCESS_FINE_LOCATION)) {
            requestPermission();
        } else {
            requestPermission();
        }
    } else {
        locationGranted=true;
        startService(new Intent(mContext, TrackingService.class));
    }
}

private void requestPermission(){
    ActivityCompat.requestPermissions( activity: this,
            new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
            Constants.MY_PERMISSIONS_MAP);
}
```

Figura 39: Ejemplo de código para solicitar permiso de geolocalización

Una vez que fue creado el “cliente” de servicios de ubicación, se puedo obtener la ubicación más reciente del dispositivo de un usuario llamando al método *getCurrentLocation*() como se muestra en la *Figura 40*.

```
private LatLng getCurrentLocation() {
    if(TrackingService.trackingLocation!=null){
        return new LatLng(TrackingService.trackingLocation.getLatitude(),
                TrackingService.trackingLocation.getLongitude());
    }else{
        return null;
    }
}
```

Figura 40: Ejemplo de código para recibir la ubicación del usuario

3.5 Acceso a la cámara del dispositivo para toma de fotografías

Para completar la encuesta se debían tomar 3 fotografías como evidencia, una del productor, otra de su identificación y una más de sus vacas. En Android, el delegar acciones a otras aplicaciones es mediante la invocación de un Intent que describe lo que desea hacer. Este proceso tiene tres partes: el Intent, una llamada para iniciar la *Activity* y por último el manejar los datos de la imagen cuando se devuelve al *activity*.

En la *Figura 41* se muestra la función que se utilizó para invocar un *intent* y capturar las fotos.

```
private void dispatchTakePictureIntent(String photoName, int imageCapture) {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile(photoName, imageCapture);
        } catch (Exception e) {
            Log.d(TAG, msg: "zz error="+e);
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Uri photoURI = FileProvider.getUriForFile( context: this,
                    Constants.AUTHORITY,
                    photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            startActivityForResult(takePictureIntent, imageCapture);
        }
    }
}
```

Figura *41: Ejemplo de código para guardar imágenes*

El método *startActivityForResult*() muestra el primer componente de actividad que manejará el *intent*.

Una vez que se tomó la foto, la forma de recuperar la imagen y hacer algo con ella, es a través de la respuesta que entrega la aplicación de cámara de Android codificando la foto en el Intent de devolución que se entrega a *onActivityResult*() comom se muestra en la *Figura 42*.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    try {
        if (resultCode == RESULT_OK) {
            if (requestCode == Constants.REQUEST_GET_SINGLE_FILE_1 ||
                    requestCode == Constants.REQUEST_GET_SINGLE_FILE_2 ||
                    requestCode == Constants.REQUEST_GET_SINGLE_FILE_3 ) {
                Uri selectedImageUri = data.getData();
```

Figura 42: Ejemplo de código para recibir la respuesta de la cámara del dispositivo móvil

3.6 Almacenamiento de encuestas en la memoria interna del dispositivo

Una vez que las respuestas de las encuestas fueron capturadas por parte del encuestador, la aplicación almacena esta información en la base de datos del dispositivo.

Guardar datos en una base de datos es ideal para los datos estructurados o para datos que se desean almacenar con seguridad y con persistencia dentro de un dispositivo. En este proyecto se utilizó la base de datos *SQLite* propia del dispositivo Android.

Uno de los principios fundamentales de las bases de datos *SQL* es el esquema, el cual es una declaración formal de la manera en la que la base de datos está organizada. En la *Figura 43*, se muestra el esquema que refleja en las instrucciones SQL que se utilizó para crear la base de datos.

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(String.format("CREATE TABLE IF NOT EXISTS %s(id integer primary key autoincrement, " +
            "Cveca text, " +
            "Foliocuestionario text, " +
            "Fechaentrevista text, " +
            "Cveencuestador text, " +
            "Nombrep text, " +
            "Apelidopp text, " +
            "Apellidomp text, " +
            "Encuesta1 text, " +
            "Encuesta2 text, " +
            "Encuesta3 text, " +
            "Encuesta4 text, " +
            "Encuesta5 text, " +
            "Encuesta6 text, " +
            "Encuesta7 text, " +
            "Encuesta8 text, " +
```

Figura 43 Ejemplo de código de la creación de la tabla para el almacenamiento de encuestas en la memoria del dispositivo

Una vez que se definió el esquema de la base de datos, se implementaron los métodos para crear y leer los elementos de la base de datos y sus tablas.

Al igual que los archivos que se guardan en el almacenamiento interno del dispositivo, Android almacena la base de datos en la carpeta privada de la aplicación. Los datos están seguros porque, de forma predeterminada, esta área no es accesible para otros usuarios o aplicaciones.

La clase *SQLiteOpenHelper* contiene un conjunto útil de API para administrar la base de datos. Cuando se utiliza esta clase para obtener referencias a la base de

datos, el sistema realiza las operaciones de crear y actualizar la base de datos, que pueden llevar bastante tiempo, solo cuando es necesario y no durante el inicio de la app, es por esto que fue necesario llamar a *getWritableDatabase*() como se muestra en la *Figura 44* o *getReadableDatabase*(), al momento de leer o escribir en la base de datos [11].

```
public boolean eliminarEncuestas() {
    SQLiteDatabase baseDeDatos = dataBaseHelper.getWritableDatabase();
    baseDeDatos.execSQL("delete from "+ Constants.DB_TABLE);
    return true;
}

public long surveyCreate(Survey survey) {
    SQLiteDatabase baseDeDatos = dataBaseHelper.getWritableDatabase();
    ContentValues valoresParaInsertar = new ContentValues();
    valoresParaInsertar.put("Cveca", survey.getCveca());
    valoresParaInsertar.put("Foliocuestionario", survey.getFoliocuestionario());
    valoresParaInsertar.put("Fechaentrevista", survey.getFechaentrevista());
    valoresParaInsertar.put("Cveencuestador", survey.getCveencuestador());
    valoresParaInsertar.put("Nombrep", survey.getNombrep());
    valoresParaInsertar.put("Apelidopp", survey.getApelidopp());
    valoresParaInsertar.put("Apellidomp", survey.getApellidomp());
```

Figura 44: Ejemplo de código para agregar encuestas de la memoria del dispositivo móvil

En la *Figura 45* se muestran los métodos de *insert*(), los cuales muestrean el ID de la fila recién creada o en caso contrario retornando -1 (si hubo un error al insertar los datos), esto pasaría si se llegara a tener un conflicto con los datos preexistentes en la base de datos.

```
Survey surveyNew = new Survey();
surveyNew.setNombrep(string_edit_Nombrep);
surveyNew.setApelidopp(string_edit_Apelidopp);
surveyNew.setApellidomp(string_edit_Apellidomp);
surveyNew.setStatus("Nuevo");
surveyNew.setCveencuestador(encuestador);
surveyNew.setFoliocuestionario(foliocuestionario);

long id = surveyController.surveyCreate(surveyNew);
if (id == -1) {
    //error
    Toast.makeText( context: SurveyAddActivity.this,  text: "Error al guardar. Intenta de nuevo",
} else {
    finish();
}
```

Figura 45: Ejemplo de código para insertar información en la base de datos a través de un controlador

Para leer información desde una base de datos, utilizamos el método *query*() pasando los criterios de selección. El método *query*() combina elementos de escritura o lectura como en cualquier lenguaje *SQL*, donde los resultados de la consulta se muestran en un objeto Cursor.

Con cláusula *WHERE*, se especifican las instrucciones de selección en el método *query*() como se muestra en la *Figura 46*.

```
public ArrayList<Survey> getSurveysByEncuestadorCode(String cveencuestador) {
    Log.d(TAG, msg: "zz in getSurveysByEncuestadorCode="+cveencuestador);
    ArrayList<Survey> surveys = new ArrayList<>();
    // readable porque no vamos a modificar, solamente leer
    SQLiteDatabase baseDeDatos = dataBaseHelper.getReadableDatabase();
    String[] args = new String[] {cveencuestador};

    Cursor cursor = baseDeDatos.rawQuery( sql: "SELECT * FROM " +Constants.DB_TABLE+ " WHERE Cveencuestador=?

    if (cursor == null) {
        return surveys;
    }
```

Figura 46: Ejemplo de código para obtener información de la base de datos

Si se desea ver una fila en el cursor, se utiliza uno de los métodos de movimiento Cursor, a los que se llaman antes de comenzar a leer valores. Como el cursor comienza en la posición -1, cuando se llama a *moveToNext*(), se coloca la "posición de lectura" en la primera entrada de los resultados y se muestra si el cursor ya pasó o no la última entrada del conjunto de resultados. En cada fila se puede leer el valor de una columna llamando a uno de los métodos get Cursor, como *getString*() o *getCount*() para retornar la cantidad total de elementos como se muestra en la *Figura 47*. Para cada uno de los métodos get, opcionalmente se puede pasar la posición del índice de la columna que se desea llamando a *getColumnIndex*(). Cuando se terminan de ver los resultados, se llama a *close*() en el cursor para liberar los recursos.

```
public Integer countAllEncuestas() {
    String countQuery = "SELECT  * FROM " + Constants.DB_TABLE;
    SQLiteDatabase baseDeDatos = dataBaseHelper.getWritableDatabase();
    Cursor cursor = baseDeDatos.rawQuery(countQuery,  selectionArgs: null);
    int count = cursor.getCount();
    cursor.close();
    return count;
}
```

Figura 47: Ejemplo de código para contar el número de encuestas existentes en la base de datos

3.7 Seguridad en el acceso y envío de información

Tomando como referencia el diagrama de casos de uso del sistema especificado en la *Figura 9*, existían dos capas de seguridad para poder hacer llegar la encuesta a los servidores de LICONSA, la primer capa de seguridad se implementaba por un acceso por medio de usuario y contraseña en el cual cada encuestador tenía una diferente, esto ayudaba a resolver el uso indebido de la aplicación por alguna persona ajena a la empresa y al grupo de encuestadores. La segunda capa de seguridad esta orientada a la comunicación entre sistemas, ya que el *API* proporcionada por LICONSA requiere una llave que se debe enviar en cada petición de envío de información para autentificar que se trata de la aplicación que se había desarrollado para fines de la actualización del padrón de productores de leche.

3.8 Envío de encuestas al servicio web de LICONSA

El envío de la información llega a los servidores de LICONSA mediante un Web Service que basa su comunicación bajo el protocolo SOAP (*Simple Object Access Protocol*), el cual es un protocolo estándar de comunicación por medio de intercambio de datos XML.

En la *Figura 48* se muestra un mensaje SOAP, el cual es un documento XML ordinario con una estructura definida en la especificación del protocolo. Para enviar la encuesta en el elemento body como se muestra en la *Figura 49*. La estructura la conformaron las siguientes partes obligatorias:

- Envelope: Siendo la raíz de la estructura, que es la parte que identifica al mensaje SOAP como tal.
- Body: contiene toda información relativa a la encuesta.

```
String soap = "<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
        "<Envelope xmlns=\"http://www.w3.org/2003/05/soap-envelope/\">" +
        "<Body>" +
        "<Main.Execute xmlns=\"PadronProductores\">" +
        "<Foliocuestionario>"+survey.getFoliocuestionario()+"</Foliocuestionario>" +
        "<Fechaentrevista>"+myDate+"</Fechaentrevista>" +
        "<Cveencuestador>"+encuestador+"</Cveencuestador>" +
        "<Nombrep>"+survey.getNombrep()+"</Nombrep>" +
        "<Apelidopp>"+survey.getApelidopp()+"</Apelidopp>" +
        "<Apellidomp>"+survey.getApellidomp()+"</Apellidomp>" +
```

Figura 48: Ejemplo de código de un mensaje SOAP

```
RequestBody body = RequestBody.create(mediaType, soap);
Request request = new Request.Builder()
        .url(Constants.WS_SEND_SURVEY)
        .post(body)
        .addHeader( name: "content-type",  value: "text/xml")
        .build();
```

Figura 49: Ejemplo de código del envío de información SOAP

En la *Figura 50* se encuentra el código para poder enviar esta información, se necesitó un “cliente” *HTTP*, la primera opción fue utilizar el “cliente” de Android, *HttpURLConnection* que sirve para enviar y recibir datos de la web. Utilizar este “cliente” requería que se escribiera una gran cantidad de código repetitivo dentro de *AsyncTask* o los métodos de subprocesos en segundo plano. Es por esto que se optó por utilizar el “cliente” *OkHttp*, el cual proporciona en Android una implementación de las interfaces *HttpURLConnection* y *Apache Client* para trabajar directamente en la parte superior de Java Socket sin utilizar ninguna dependencia adicional [12]. La ventaja de utilizar *OkHttp* fue tener un código más compacto y el soporte para llamadas asíncronas. Mediante un “cliente” como se muestra en la *Figura 50* que permite realizar una llamada asíncrona.

```
OkHttpClient pre_client = new OkHttpClient.Builder()
        .connectionSpecs(Collections.singletonList(spec))
        .addInterceptor(logging)
        .build();

OkHttpClient client = HttpClient.trustAllSslClient(pre_client);
MediaType mediaType = MediaType.parse("text/xml");
```

Figura 50: Ejemplo de código del "cliente" HTTP

Despues de enviar el request por parte del "Cliente", se espera una respuesta de vuelta (*Callback*) en la función (*onResponse*) y en caso de que la respuesta fuera satisfactoria se cambiaba el status de la encuesta a "Enviada" como se muestra en la *Figura 51*.

```
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onResponse(Call call, final Response response) {
        if (response.isSuccessful()) {
            try{
                String responseString=response.body().string();
                String[] parseResponse =responseString.split( regex: "</Salida>");
                String isTrue =parseResponse[0].substring(parseResponse[0].length() - 4);
                Handler handler = new Handler(Looper.getMainLooper());
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        progress_bar.setVisibility(View.GONE);

                        if(isTrue.toLowerCase().equals("true")){
                            survey.setStatus("Enviada");
                            long id = surveyController.saveStatus(survey);
                            finish();
```

Figura 51: Ejemplo de código de la respuesta HTTP

Capítulo 4. Pruebas y puesta en marcha

Las pruebas de la aplicación iniciaron en el emulador integrado en Android Studio llamado AVDM (*Android Virtual Device Manager*), como se muestra en el la *Figura 53*, donde fue seleccionado un dispositivo virtual Pixel 4 con API 28, resolución de 1080px y procesador x86

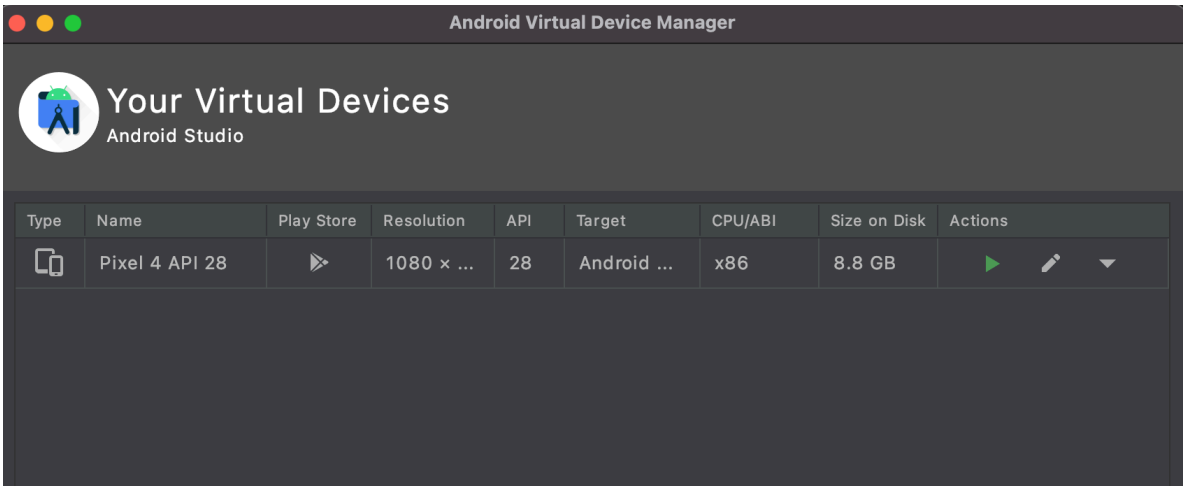


Figura 53: Selección de dispositivo Pixel 4 en AVDM

En la *figura 54* se muestra la captura de la encuesta en el dispositivo virtual Pixel 4.

Figura 54: Ejecución de la aplicación móvil en dispositivo virtual

El AVDM ofrece la posibilidad de simulazar ubicaciones geográficas para realizar pruebas de geolocalización con el GPS, en la *figura 55* se observa una posición en el mapa indicando una ubación en el estado de Puebla.

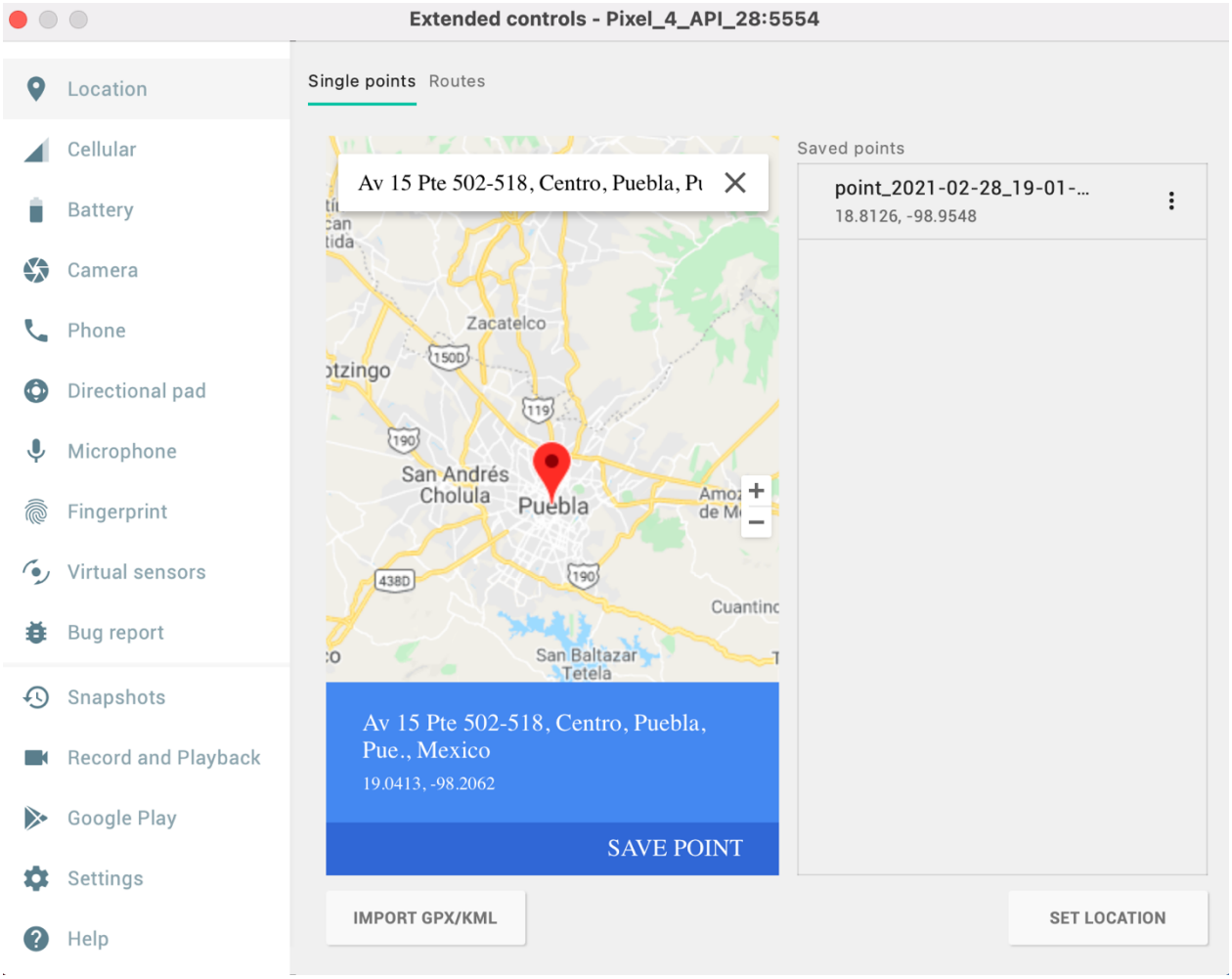


Figura 55: Simulación de geolocalización en el dispositivo virtual

Posteriormente se realizaron pruebas en un dispositivo físico como se muestra en la *Figura* 56. Siendo un dispositivo móvil con sistema operativo Android de marca Motorola modelo G6 con tamaño de memoria de 3GB y un *CPU Qualcomm Snapdragon 450* de 8 núcleos y 1.8 GHz. Respecto a la computadora en la que se desarrolló fue una Macbook Pro de 8GB de memoria *RAM* y procesador Intel core i5.

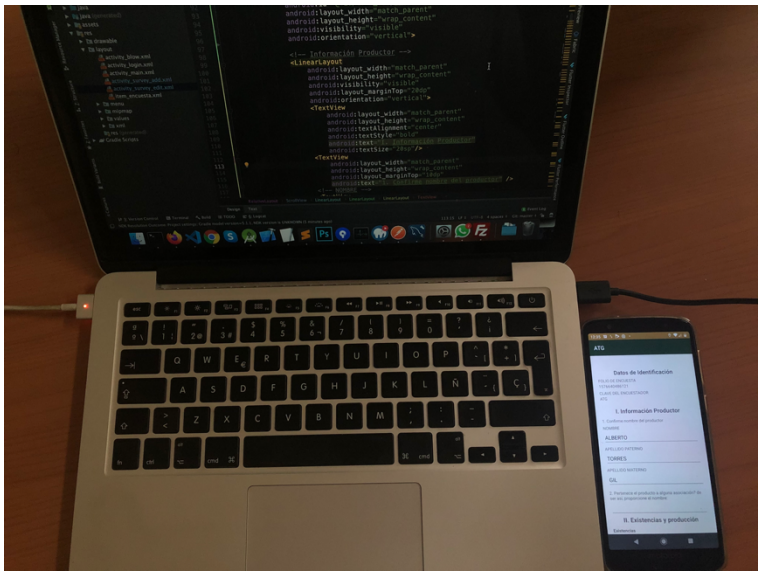


Figura 56: Ejecución de la aplicación en un dispositivo móvil físico

4.1 Requerimientos mínimos

Los requerimientos mínimos para ejecutar la aplicación fueron establecidos a partir de medir el consumo promedio de los dispositivos android, de los cuales básicamente el sistema operativo consume 1GB de memoria *RAM*, dado esta información de consumo memoria, más el consumo de 128 MB de *RAM* utilizados por la aplicación de la encuesta nos hicieron fijar un requerimiento mínimo de 2GB de memoria *RAM*, ya que comercialmente se suelen vender dispositivos móviles ANDROID con memoria *RAM* en múltiplos de 1GB.

En la Figura 57 y 58 se muestran los recursos de memoria consumidos en promedio por un dispositivo android marca Motorola G6 de 3GB de memoria RAM, en el cual

se observa un consumo promedio de 1.6 GB por parte del Sistema e interfaz gráfica del usario.

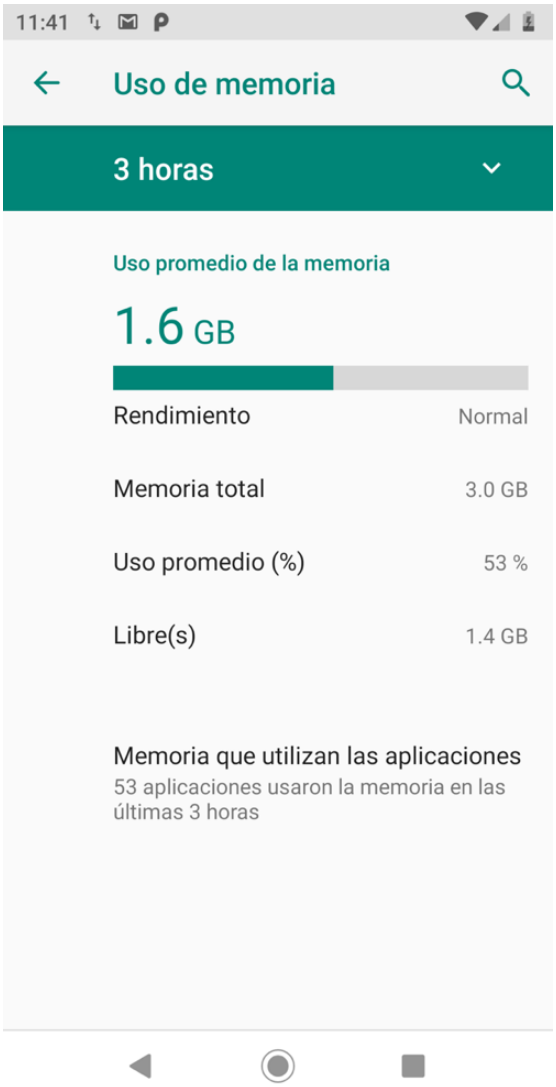


Figura 57 Uso de memoria RAM utilizada en promedio en un dispositivo android Motorola G6

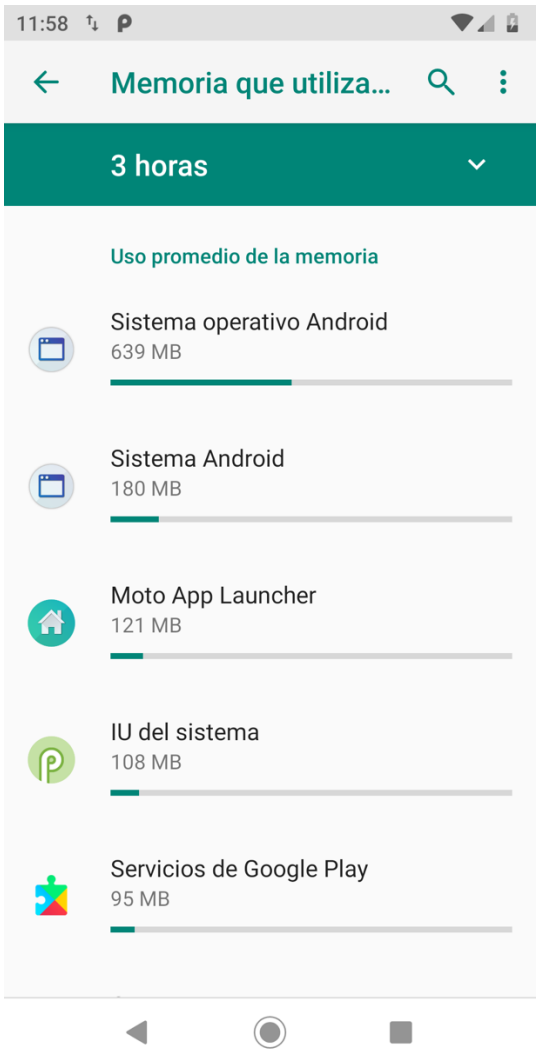


Figura 58 Uso promedio de memoria RAM segmentado por las principales aplicaciones y servicios en un dispositivo android Motorola G6

4.2 Recursos de *hardware* consumidos por la aplicación móvil en Android

A continuación se muestra el consumo de recursos por parte de las actividades de Login, Main y de la captura de la encuesta, en los cuales podemos visualizar un consumo de memoria *RAM* con rangos de hasta 128MB.

En la *Figura 59* se muestran los recursos de hardware consumidos por la actividad de *Login*.

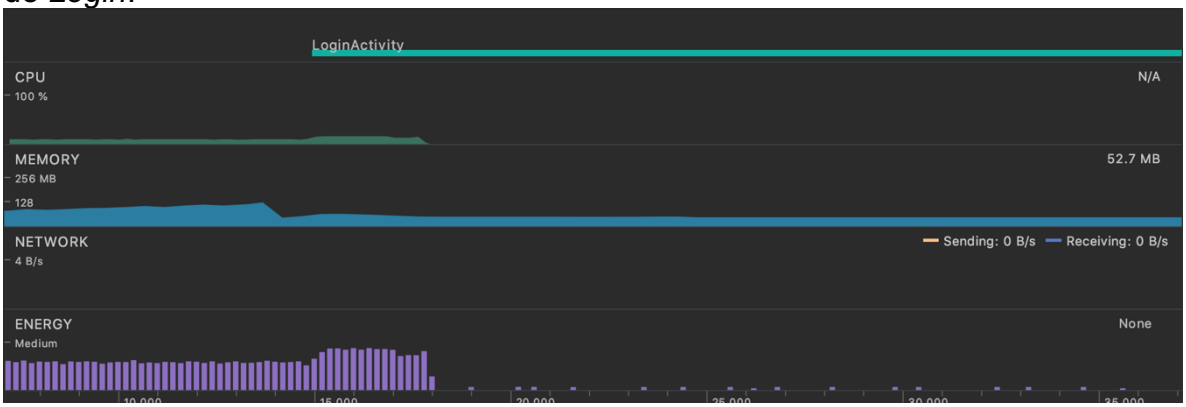


Figura 59: Ejecución de la aplicación en un dispositivo móvil físico

En la *Figura 60* se muestran los recursos de *hardware* consumidos por la actividad de la pantalla principal

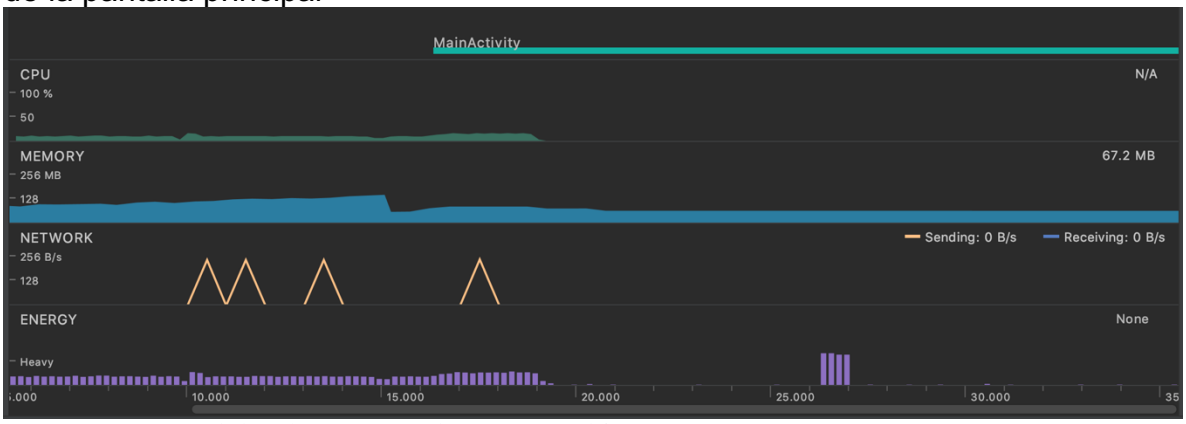


Figura 60 Ejecución de la aplicación en un dispositivo móvil físico

En la *Figura 61* se muestran los recursos de *hardware* consumidos por la actividad de la encuesta

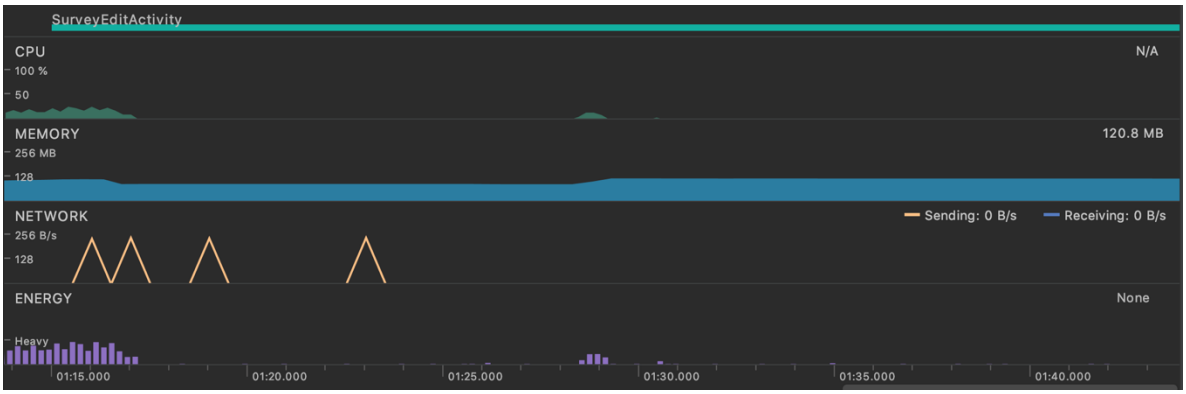


Figura 61: Ejecución de la aplicación en un dispositivo móvil fisico

4.3 Resultados

Con la ayuda de esta aplicación se lograron encuestar a un total de 23,439 productores.

En la *Figura 62* se muestra el mapa de calor de los Productores Pequeños con un total de 18,684

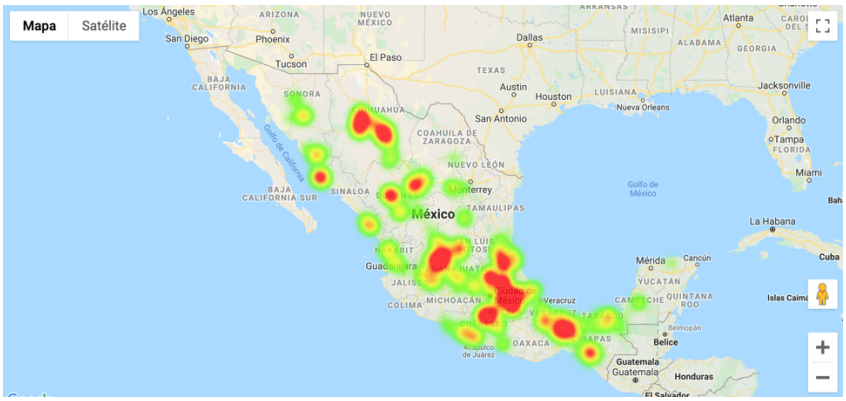


Figura 62 Distribución de productores pequeños de leche en México

En la *Figura 63* se muestra el mapa de calor de los Productores Medianos con un total de 4,277

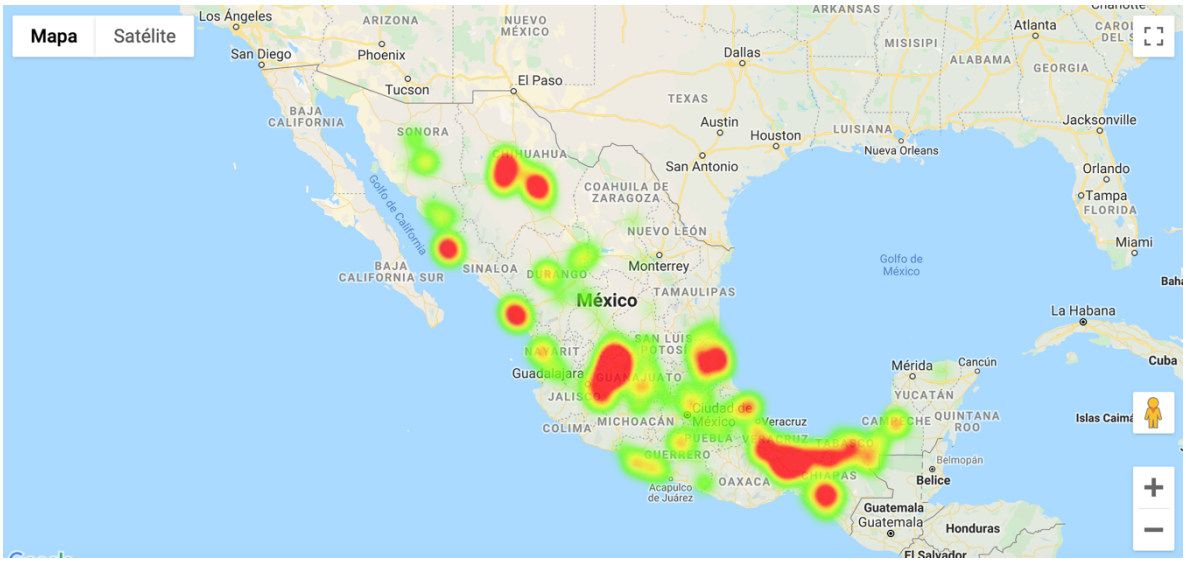


Figura 63 Distribución de productores medianos de leche en México

En la *Figura 64* se muestra el mapa de calor de los Productores Grandes con un total de 478

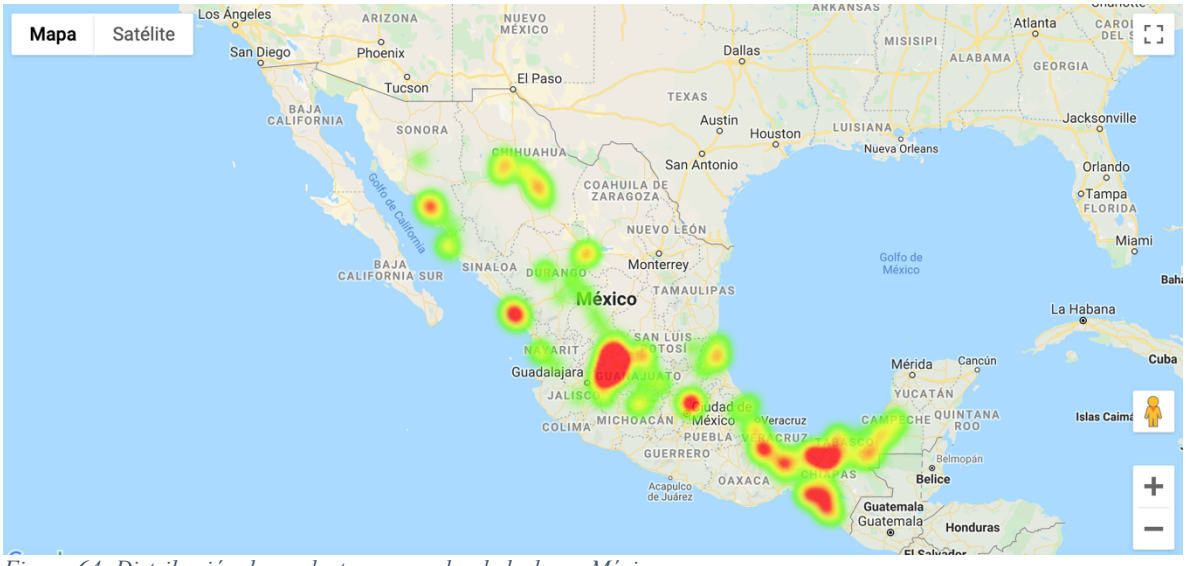


Figura 64: Distribución de productores grandes de leche en México

4.4 Conclusiones

Esta aplicación con estas características antes mencionadas, ayudó al equipo de encuestadores a realizar su labor de forma más eficiente que si la hubieran realizado en papel, ya que que al tener la encuesta desde su captura en forma digital, no hubo procesos intermedios en digitalizar la información previniendo alterar la información a diferencia que si se hubiera realizado en papel.

Para el desarrollo de la aplicación fue necesario contar con encuestadores que recopilaran la información de los productores de leche, recopilando la georeferenciación del lugar y tomando fotografías como evidencia de sus vacas. Estos datos georeferenciados ayudarían a actualizar el padrón de productores de leche, identificar que zonas necesitarían más centro de acopio y eliminar a los intermediarios para que el precio de garantía beneficiara directamente a los pequeños productores [13].

La aplicación móvil para la captura de encuestas facilitó mucho el trabajo en sitio y aunque se redujo el error humano al restringir los campos de texto o numérico de las respuestas, es posible que se cometieran errores de percepción por parte de los encuestadores como sucede en cualquier otro campo laboral.

Se obtuvo una ventaja significativa al desarrollar una aplicación móvil nativa hecha a la medida, ya que fue 100% adaptable a las necesidades de Liconsa y fue evolucionando en base a necesidades específicas que se presentaron en el desarrollo del proyecto a lo largo de las reuniones junto con el equipo técnico de Liconsa.

Los dispositivos móviles que se utilizaron en la captura de encuestas tenían como mínimo 3GB de memoria RAM el cual fue un recurso de hardware suficiente para operar la aplicación de captura de encuestas junto con posibles aplicaciones de mensajería y tener espacio extra de 500 MB de RAM aproximadamente para operar sin complicaciones.

Los dispositivos de gama alta toman fotografías de muy alta calidad, por lo que al guardar las imágenes con la resolución en que fueron tomadas y enviarlas con esa resolución a Liconsa se tuvieron errores de timeout, por lo que fue necesario reducir el peso de las imágenes sin comprometer la calidad comprimiéndolas en formato JPG.

Una mejor solución en el envío y almacenamiento de imágenes pudo haber sido enviarlas como archivos a través de http en lugar de base64, ya que en la práctica el almacenamiento de imágenes codificadas en base64 ocupó mucho espacio en la base de datos de Liconsa ocasionando errores de espacio en la tabla de las encuestas de tipo “*table space exceeded*”.

Glosario

Centro de acopio: Establecimiento que cuenta con la infraestructura necesaria para recolectar la producción de leche de productores.

Producción de leche obtenida: Se refiere a la cantidad total de litros de leche producida por la unidad de producción, durante el periodo de referencia.

Existencias de vacas doble propósito: Total de vacas del ganado bovino que por lo menos han tenido un parto y están destinadas tanto a la producción de leche como a la crianza de becerros.

Promedio de litros obtenidos por vaca: El volumen total de leche que se obtiene en la unidad de producción entre el número total de vacas ordeñadas.

Vaca seca: Es la vaca que se encuestra en la última etapa de preñez, cuya lactancia ha sido terminada y que está siendo preparada para la próxima lactancia. Secar una vaca es parar de ordeñarla, este periodo es de cerca de 60 días entre los dos periodos de lactancia.

Alta de productor. Para dar de alta a un productor dentro de la aplicación, este debería ingresar su nombre, apellido paterno y apellido materno, una vez dado de alta en la aplicación, podría acceder a la siguiente pantalla para levantar la encuesta.

Bibliografía

[1] Escribano, D. (2019, 18 marzo). *Esta es la historia de las aplicaciones móviles*. Skyscanner España. https://www.skyscanner.es/noticias/esta-es-la-historia-de-las-aplicaciones-moviles

[2] Colaboradores de Wikipedia. (2019, 24 enero). *Aplicación móvil*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_m%C3%B3vil

[3] *Elegir aplicaciones nativas o multiplataforma: ventajas e inconvenientes*. (2018). Aplicaciones para móviles. https://www.aplicacionesparamoviles.com/desarrollo-de-aplicaciones-moviles-nativo-plataforma/

[4] Rumbaugh, J., Jacobson, I., Booch, G., & Grady Booch. (1999). *El Lenguaje Unificado de Modelado*. Addison-Wesley.

[5] *Guías para desarrolladores | Desarrolladores de Android*. (2019). Android Developers. https://developer.android.com/guide?hl=es-419

[6] *Glide v4: Fast and efficient image loading for Android*. (2019). Glide. https://bumptech.github.io/glide/

[7] *Introducción a las actividades | Desarrolladores de Android*. (2019). Android Developers. https://developer.android.com/guide/components/activities/intro-activities?hl=es-419

[8] *Intents y filtros de intents | Desarrolladores de Android*. (2019). Android Developers. https://developer.android.com/guide/components/intents-filters?hl=es-419

[9] *Create a List with RecyclerView | Desarrolladores de Android*. (2019). Android Developers. https://developer.android.com/guide/topics/ui/layout/recyclerview

[10] *Request location updates | Desarrolladores de Android*. (2019). Android Developers. https://developer.android.com/training/location/request-updates

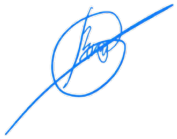
[11] *Save data using SQLite | Desarrolladores de Android*. (2019). Android Developers. https://developer.android.com/training/data-storage/sqlite

[12] Square, Inc. (2019). *OkHttp*. Square. https://square.github.io/okhttp/

[13] República, P. (2019). *Diversas intervenciones durante la entrega de Precios de Garantía para pequeños y medianos productores y productoras de leche en Jalisco*. gob.mx. https://www.gob.mx/presidencia/prensa/diversas-intervenciones-durante-la-entrega-de-precios-de-garantia-para-pequenos-y-medianos-productores-y-productoras-de-leche-en-jalisco

Firmas de asesores

Dra. María Monserrat Morín Castillo



M.C. Ricardo Álvarez González