ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO MULTIMEDIA PARA EL REGISTRO Y ALMACENAMIENTO DE LA LENGUA NATIVA QUICHUA DEL ECUADOR A TRAVÉS DE UNA APLICACIÓN MÓVIL ANDROID

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

BOLÍVAR IVÁN ROBAYO PAGUAY

bolivar.robayo@epn.edu.ec

DIRECTOR: ING. RICARDO XAVIER LLUGSI CAÑAR, MSc.

ricardo.llugsi@epn.edu.ec

CODIRECTOR: ING. PABLO ANÍBAL LUPERA MORILLO, PhD.

pablo.lupera@epn.edu.ec

Quito, febrero 2019

AVAL

Certificamos que el presente trabajo fue desarrollado por Bolívar Iván Robayo Paguay, bajo nuestra supervisión.

Ing. Ricardo Xavier Llugsi Cañar, MSc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

Ing. Pablo Aníbal Lupera Morillo, PhD.
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Bolívar Iván Robayo Paguay, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Bolívar Iván Robayo Paguay

DEDICATORIA

A mi familia, por ser el motor que siempre estuvo apoyándome. Cada esfuerzo es y será siempre para ustedes, son el pilar que me ha permitido seguir adelante a pesar de cualquier tropiezo o dificultad. Cada línea de código, pensamiento, palabra, idea e imagen que contenga este trabajo es dedicado a ustedes.

A todas las personas que estuvieron pendiente del proceso de realización de este proyecto.

Iván R.

AGRADECIMIENTO

Me faltan palabras para expresar todo el cúmulo de sensaciones que cruzan por mí, agradezco infinitamente a Dios, por brindarme los conocimientos, fuerza y valores necesarios para culminar con éxito esta etapa de mi vida.

A mi familia, por toda la paciencia y amor que siempre me han tenido.

Al Consejo del Gobierno Comunitario San Francisco de Chibuleo, por su ayuda, apoyo y participación en la ejecución de este proyecto.

A cada miembro de la comunidad San Francisco de Chibuleo, muchas gracias, en especial a los 30 miembros que participaron de forma directa en este proyecto.

A Carlos Olmedo por toda la ayuda brindada antes, durante y después de la ejecución del proyecto, gracias por todas las gestiones e información compartida.

A Juan José Espín y Martha Lligalo, docentes de la Unidad Educativa del Milenio Chibuleo, gracias por la retroalimentación brindada en la elaboración del presente proyecto.

A todos los estudiantes de la Escuela Politécnica Nacional que participaron en la adquisición de datos, mis sinceros agradecimientos, sin ustedes esto no hubiera sido posible.

A mi codirector de tesis, Ing. Pablo Lupera, PhD, por estar siempre disponible ante cualquier duda, por el tiempo y la ayuda brindada a lo largo de todo el proyecto.

A mi director de tesis, Ing. Ricardo Llugsi, MSc, por su completo apoyo en todo este proceso, por la guía en este camino y por estar siempre ahí, dispuesto a brindar su tiempo y experiencia.

Gracias a todos, fue una gran experiencia, quedará marcada en mi vida, he aprendido muchas cosas, tanto en lo técnico como personal, anhelo seguir trabajando en proyectos enfocados a la preservación de lenguas nativas en peligro de extinción.

Iván R.

ÍNDICE DE CONTENIDO

RESUMEN

A lo largo de la historia muchas lenguas nativas han desaparecido y con ellas un conjunto de conocimientos ancestrales que van desde tradiciones, rituales, leyendas, hasta conocimientos en medicina ancestral, cosmología, armonía del espíritu, percepción del universo, entre otros.

Múltiples organismos han realizado incontables esfuerzos por preservar lenguas en peligro de extinción. Este proyecto técnico se enmarca en este propósito y pretende aportar con el diseño e implementación de una herramienta tecnológica multimedia, que facilite el proceso de registro y almacenamiento de lenguas nativas.

Gracias al avance de la tecnología, hoy es posible almacenar una gran cantidad de datos en la nube, para luego poder acceder a ellos desde cualquier parte del mundo a través de un navegador web. Con base en esta posibilidad, surge la idea de preservar lenguas nativas en la nube, que es un medio seguro, de fácil acceso y alta disponibilidad.

El prototipo del proyecto fue desarrollado para el sistema operativo móvil Android, y como servicios de acopio en la nube se utilizó los proporcionados por la cuenta de almacenamiento del proveedor de servicios de nube Microsoft Azure.

La recolección de datos del proyecto se realizó en la comunidad indígena San Francisco de Chibuleo, perteneciente al pueblo Chibuleo de la nacionalidad Kichwa, y ubicada al sur – occidente de la provincia de Tungurahua, en la sierra central de Ecuador. Aquí se registró una muestra del idioma quichua, alcanzando un nivel de confianza mayor al 95%, con un margen de error menor al 5%.

PALABRAS CLAVE: Lenguas nativas, conocimientos ancestrales, comunidad indígena, San Francisco de Chibuleo, computación en la nube, almacenamiento en la nube, aplicación Android.

ABSTRACT

Through the years many native languages have disappeared and with them a lot of ancient knowledge. Traditions, rituals and legends, knowledge about ancient medicine, astrology, soul harmony, and perception of the universe, among others, have also disappeared.

Many institutions have made countless efforts to preserve languages that are on the brink of extinction. This technical project is framed within this purpose and pretends to contribute with the design and implementation of a technological multimedia tool that facilitates the registry and storage process of native languages.

Thanks to the breakthrough of technology, today, is possible to store a big deal of data in a cloud so we can access to this data from any part of the world using any search engine. Based in this possibility, the idea of preserving native languages in a cloud emerges because it is a safe place, easy to access and highly available.

The prototype of the project was developed for the Android Operating System, Microsoft Azure used the cloud storage services provided by any of the carrier's accounts.

The data collected for this Project was from the community of San Francisco in Chibuleo, which belongs to the Chibuleo Town - Kichwa located in the south-west region of the Tungurahua Province in Ecuador's Central Sierra. A sample of the Quichua Language was taken in this place, reaching a level of confidence of 95% with an error margin of less than 5%.

KEYWORDS: Native languages, ancient knowledge, community, San Francisco in Chibuleo, cloud computing, cloud storage, Android application.

1. INTRODUCCIÓN

La organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura (UNESCO) [1] estima que a nivel mundial 2500 lenguas están en peligro de extinción y 10% de éstas, extintas desde 1950. A pesar de que existen acuerdos internacionales y políticas de estado para preservar este tipo de lenguas, no son suficientes para garantizar la vitalidad de una lengua.

Toda lengua, independiente de su grado de difusión, plasma una cosmovisión única y específica, por ejemplo, las lenguas indígenas han sido utilizadas para transmitir conocimientos ancestrales de una generación a otra; tales como: tradiciones, rituales, leyendas, medicina, percepción del mundo y el universo, entre otras.

El fin de una lengua conlleva a un desvanecimiento cultural inevitable y en muchos casos irreversible, además de la ausencia de un patrimonio intangible. El impacto de esta pérdida no puede medirse en su totalidad hasta ser demasiado tarde.

En Latinoamérica las lenguas indígenas tienden a desaparecer, por varios factores que van desde el aislamiento de los mismos pueblos, hasta la discriminación que ha sufrido la población indígena, ocasionando que los hablantes nativos dejen de enseñar su lengua a sus hijos.

La ACL (*The Association for Computational Linguistics*, Asociación de Lingüística Computacional) [2] indica que existen esfuerzos por documentar lenguas en peligro de extinción, sin embargo, hay una brecha entre investigadores lingüistas y el personal técnico que desarrolla estas herramientas, debido a que cada profesional tiene un enfoque diferente en su formación académica.

Es necesario desarrollar instrumentos que faciliten el trabajo de investigadores lingüistas, estas herramientas deben ser de fácil acceso y utilización. El presente proyecto contribuye con el diseño e implementación de un prototipo multimedia que facilita el proceso de registro y almacenamiento de la lengua nativa quichua del Ecuador.

1.1 Objetivos

El objetivo general de este Proyecto Técnico es:

- Diseñar e implementar un prototipo multimedia que permita el registro y almacenamiento del idioma quichua en una base de datos alojada en la nube a través de una aplicación móvil Android.

Los objetivos específicos de este Proyecto Técnico son:

- Sintetizar la situación actual del idioma quichua en el Ecuador tomando en cuenta la importancia de la conservación de lenguas nativas como ésta.
- Estudiar las principales plataformas para el desarrollo de aplicaciones móviles Android y para el alojamiento de bases de datos en la nube.
- Establecer los principales requerimientos de un sistema de registro y almacenamiento de lenguas nativas incluyendo una muestra con los términos quichuas más utilizados en las comunidades para el dimensionamiento del sistema.
- Elaborar una aplicación móvil que permita la recolección de datos de pronunciación, escritura y traducción de palabras quichua.
- Diseñar y construir una base de datos alojada en la nube para la organización y almacenamiento de los datos recolectados.

1.2 Alcance

El presente trabajo tiene como alcance registrar una muestra del vocabulario quichua, la recolección de datos se realizará en una comunidad indígena ecuatoriana, cuya primera lengua sea el quichua.

La aplicación móvil Android registrará datos informativos de la comunidad indígena, tales como nombre, ubicación, localización e idioma nativo; datos informativos de los miembros que participen en la recolección de datos como nombre, edad y género; grabará audios para el registro de la pronunciación de cada palabra quichua; registrará la traducción al castellano y la escritura de cada palabra. Para determinar la localización exacta de la comunidad se utilizarán los datos proporcionados por el GPS del dispositivo móvil. Las especificaciones de la aplicación y metadatos a registrar se basan en la etapa de registro audiovisual para la documentación lingüística [3] y [4].

Todos los archivos de audio grabados y datos recolectados serán enviados a través de la conexión de internet del dispositivo móvil hacia una base de datos alojada en la nube.

En [5] se establecen 1920 palabras para el idioma quichua, tomando como referencia este valor, el tamaño mínimo de la muestra con un margen de error de 5% y un nivel de confianza del 95% es de 321 palabras.

La aplicación será un prototipo que posteriormente podrá ser utilizado para el registro de otras lenguas nativas, dependiendo de las características propias de cada lengua se podrá documentar fonemas, palabras, frases, oraciones y conversaciones completas.

1.3 Marco Teórico

1.3.1 Idioma quichua

En el periodo prehispánico, antes que los Incas llegaran a lo que hoy es la República del Ecuador, ya existían diferentes grupos con un alto grado de organización. Luego de que los Incas conquistaron a estos grupos, impusieron el quechua como lengua de uso general en todo su territorio Tawantinsuyo [6].

Figura 1.1 Mapa de la extensión del Tawantinsuyo [6].

Como se ve en la Figura 1.1, el Imperio Inca conocido como Tawantinsuyo está representado por el área punteada, iba desde el sur de Colombia hasta el norte de

Argentina. A pesar de que los Incas usaban la lengua quechua como medida de control para la conquista de sus territorios, existían dos variantes de ésta. El quechua I usado en el Perú central y el quechua II hablado en el resto del Tawantinsuyo, la versión II tenía su propia subdivisión, norteña y sureña. El idioma quichua corresponde a la variante norteña del quechua II [6].

El diccionario de la Real Academia Española (RAE) define a la palabra quichua como una variedad del quechua que se habla en el Ecuador.

1.3.1.1 Situación actual

En la Constitución de la República del Ecuador del 2008 (Artículo 2), se reconoce al quichua como idioma oficial de relación intercultural. La nacionalidad Kichwa abarca a diferentes pueblos indígenas, ubicados principalmente en la sierra del Ecuador, todos los pueblos pertenecientes a esta nacionalidad están vinculados con el idioma quichua.

Según datos del Censo de Población y Vivienda (CPV-2010) [7], los principales pueblos indígenas que pertenecen a la nacionalidad Kichwa se visualizan en la Tabla 1.1

Tabla 1.1 Pueblos indígenas pertenecientes a la nacionalidad Kichwa [7].

Pueblo	**Ubicación**
Chibuleo	Tungurahua
Kañari	Azuay, Cañar
Karanki	Imbabura
Kayambi	Imbabura, Napo, Pichincha
Kichwa Amazonía	Napo, Orellana, Pastaza, Sucumbíos
Kichwa Tungurahua	Tungurahua
Kisapincha	Tungurahua
Kitukara	Pichincha
Natabuela	Imbabura
Otavalo	Imbabura
Panzaleo	Cotopaxi
Puruhá	Chimborazo
Salasaka	Tungurahua
Saraguro	Loja, Zamora Chinchipe
Waranca	Bolívar

En [1] la UNESCO señala que la vitalidad de la lengua quichua está en la categoría de peligro, también señala que el número de hablantes quichua en el Ecuador oscila entre 340.000 y 3’000.000 de hablantes.

1.3.1.2 Pueblo Chibuleo

Es un pueblo indígena perteneciente a la nacionalidad Kichwa, ubicado en la parroquia Juan Benigno Vela, cantón Ambato, provincia de Tungurahua. Aproximadamente hay 12.000 habitantes que pertenecen a este pueblo, divididos en siete comunidades, quienes hablan quichua y castellano, cabe mencionar que su lengua materna es el quichua.

Las comunidades originarias de este pueblo son: San Alfonso, San Francisco, San Luis, y San Pedro, posteriormente se incluyeron: San Miguel, Chacapungo y Pataló Alto [8].

1.3.1.3 Comunidad San Francisco de Chibuleo

La comunidad San Francisco de Chibuleo se encuentra ubicada en el sector Sur – Oeste de la provincia de Tungurahua, a 15,9 km por carretera desde el centro de la ciudad de Ambato, vía Guaranda. La altura promedio de la comunidad San Francisco es de 3200 m sobre el nivel del mar.

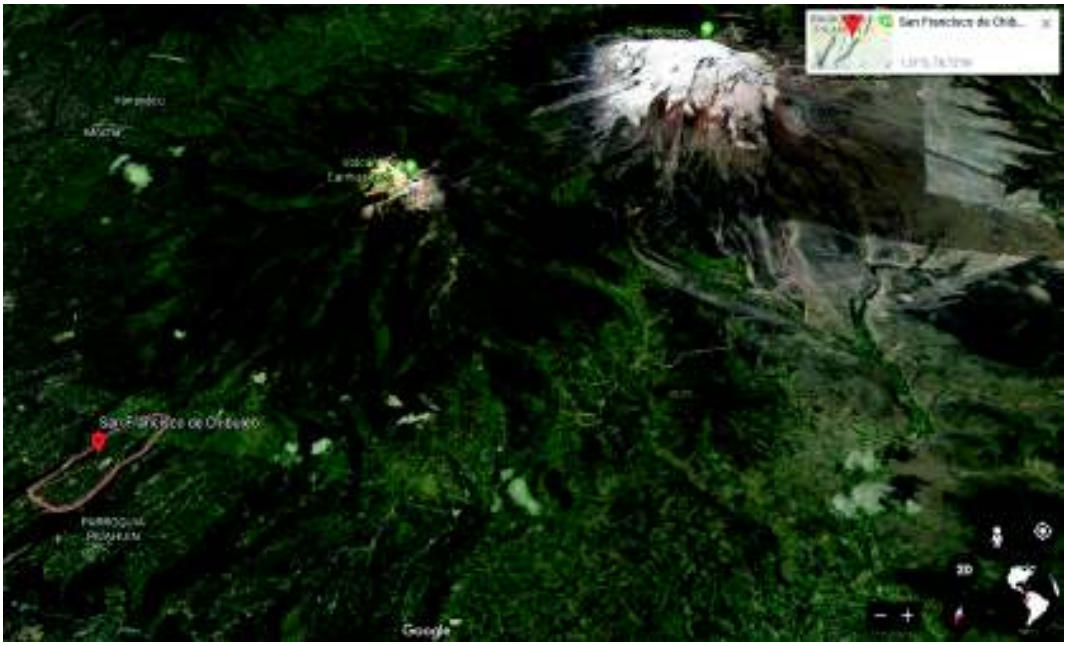


Figura 1.2 Ubicación comunidad San Francisco con relación al volcán Chimborazo.

Se estima que en toda la comunidad San Francisco de Chibuleo existe un aproximado de 600 familias y 2500 habitantes [9].

El quichua en la Comunidad San Francisco de Chibuleo [10].

Durante el trabajo de campo Juan José Espín, miembro de la comunidad y docente de quichua en la Unidad Educativa del Milenio Chibuleo, hizo un recuento de la evolución y trascendencia de esta lengua en los pueblos indígenas:

> Es importante que no desaparezca nuestra lengua […] porque gran parte de lo que ahora es América del Sur era un solo territorio, el Tahuantinsuyo. Ya existían culturas como los Puruháes, Quitus, Kitu Karas, Panzaleos entre otros pueblos que ya hablaban su propio idioma y manejaban su propia cultura. Con la llegada de europeos, un poco se dispersó la cultura y se dio una mezcla del idioma con el castellano, que hasta ahora se habla mucho y se lo conoce como el quichuañol. Inclusive en comunidades indígenas se habla mucho con esta mezcla de palabras. Claro que como lengua quichua ha sido todo lo que es oral, desde hace muchos años, desde los Incas ha venido, trasmitiéndose de generación en generación oralmente. No hay textos escritos, en los últimos años se ha iniciado un proceso de escribir textos en quichua [10].

Para Juan José: "La lengua es nuestra, nuestro propio elemento cultural, entonces deberíamos más bien valorar, rescatar lo que antiguamente ha ido desapareciendo".

Él cuenta que en la comunidad San Francisco de Chibuleo a los niños y niñas les enseñan a escribir y enviar correos en quichua como parte del esfuerzo por mantener el idioma en la vida cotidiana. "Por lo menos con los estudiantes de la comunidad, pero se debería reforzar estas iniciativas con el resto de la sociedad", recalca.

Juan José apunta que es muy difícil sostener el uso de la lengua cuando el español es el lenguaje que se usa para la vida diaria. Por ejemplo, los empaques de medicinas no están escritos en quichua, "En todas las cosas que contengan etiquetas, estas situaciones como que quieren perder la lengua. Pero se intenta difundir a los niños que utilicen el idioma quichua en especial a la madre que es la persona con la que más contacto tiene el bebé", dice y agrega: "El niño tiene que desarrollar como primera lengua el quichua, luego el castellano, es segunda lengua que tiene que aprender para la interrelación y comunicación con el resto de la sociedad".

Por ello, para este docente de la comunidad San Francisco de Chibuleo es de utilidad la existencia de herramientas tecnológicas que ayuden al proceso de enseñanza del idioma quichua como primera lengua.

Lo expresa en estas palabras: "El internet, las computadoras y todo dispositivo electrónico está configurado en castellano o inglés. Es necesario que existan herramientas que permitan visualizar páginas web en quichua, así como el idioma en los dispositivos".

Aunque Juan José hace una referencia general a la tecnología, su testimonio es relevante para resaltar la necesidad de proyectos que vinculen el desarrollo tecnológico con el rescate, conservación y/o difusión de las lenguas ancestrales de pueblos y nacionalidades indígenas del Ecuador, que es parte de los objetivos de este proyecto.

1.3.2 Base de datos

Es la colección de datos organizados, de tal forma que puedan generar información que sea pertinente para una empresa, organización, aplicación, etc. Entre los principales tipos de bases de datos se tienen las relacionales y no relacionales [11].

1.3.2.1 Base de datos relacional

Este tipo de base de datos usa el modelo relacional, en el cual se utiliza un grupo de tablas para organizar datos y relaciones existentes entre ellos. Los datos almacenados en una tabla se conocen como registros, almacenados en las filas. Cada registro tiene un conjunto de atributos llamados campos, representados por las columnas [11].

A continuación, se describe un ejemplo de base de datos relacional que incluye tres tablas: la primera contiene datos de los clientes de un banco, la segunda corresponde a los saldos por cada cuenta y la tercera contiene las cuentas pertenecientes a cada cliente.

Todas las tablas del ejemplo contienen 3 registros cada una. La Tabla 1.2 tiene 4 campos: ID[1], Nombre, Edad y Ciudad; la Tabla 1.3 incluye las columnas Cuenta y Saldo; y la Tabla 1.4 contiene los campos ID y Cuenta.

Tabla 1.2 Tabla cliente.

ID	Nombre	Edad	Ciudad
001	Manuel	21	Quito
002	Luis	35	Cuenca
003	Diego	24	Ibarra

Tabla 1.3 Tabla saldo.

Cuenta	Saldo
54823489	580
28095461	230
57016949	102

Tabla 1.4 Tabla cliente-cuenta

ID	Cuenta
001	54823489
002	28095461
003	57016949

Las bases de datos relacionales contienen registros del mismo tipo para cada tabla, en consecuencia, éstas son de tamaño fijo con respecto a las columnas. Es recomendable

[1] ID, abreviatura de identificador

usarlas en sistemas cuya estructura de datos sea bien conocida, existan muchas relaciones entre tablas y no se espere mayor grado de crecimiento horizontal [11].

Para la administración de este tipo de base de datos, se requiere de un SGBD (Sistema Gestor de Base de Datos), que sirve de intermediario entre éstas y los usuarios que las utilicen. La comunicación entre el administrador del sistema y el SGBD, se realiza a través de un lenguaje llamado SQL (*Structured Query Language*, Lenguaje de Consulta Estructurada), estandarizado por la ISO (*International Organization for Standardization*, Organización Internacional de Estandarización) [11], [12].

Existe una gran variedad de SGBD, los más conocidos y utilizados:

- MySQL
- Microsoft SQL Server
- Oracle
- SQLite
- IBM DB2
- PostgreSQL

1.3.2.2 Base de datos no relacional

Desde la aparición de la web 2.0[2], los usuarios empezaron a subir gran cantidad de datos a los servidores de las aplicaciones, haciendo que éstas usen más recursos de procesamiento y almacenamiento. Una solución alternativa y menos costosa fue crear bases de datos no relacionales, con una estructura más flexible, permitiendo un mejor crecimiento horizontal, mayor rendimiento y escalabilidad [13].

A este tipo de base de datos también se la conoce como No-SQL (*Not Only SQL*), debido que prescinden del modelo relacional, razón por la cual, el tamaño de un registro a otro, dentro de una misma tabla puede variar [13].

A continuación, se detalla un ejemplo de base de datos no relacional. En la Tabla 1.5 se registran datos personales de los usuarios de una red social. La base de datos contiene el registro de tres usuarios en una sola tabla. A pesar de que todos los usuarios pertenecen a la misma red social, los campos de datos personales pueden ser diferentes en cada caso.

[2] Sistema de trasmisión de información a través de internet, también conocido como web social [49].

Tabla 1.5 Tabla de datos personales

Usuario	ID	Nombre	Ciudad	Edad	Correo
Felipe03	001	Felipe Martínez	Ibarra	20	fm@redsocial.ec
Luisa1992	002	Luisa Paredes		25	
MKPancho	003	Miguel Muñoz	Quito		mm@redsocial.ec

Las bases de datos no relacionales pueden contener registros de diferente tipo dentro de una misma tabla por su naturaleza de estructura flexible. Se recomienda usar este tipo de base de datos en situaciones que se prevea tener una gran cantidad de usuarios y se espere un crecimiento horizontal considerable en las tablas.

La principal desventaja frente a bases de datos relacionales es la falta de estandarización, a causa de esto, será necesario una revisión de conceptos específicos, dependiendo del proveedor de servicios de la base de datos.

1.3.3 Computación en la nube

También conocida como *cloud computing*, consiste en un conjunto de servicios informáticos, que pueden ser usados por el usuario a través de Internet. Cada proveedor de nube puede tener una gran variedad de servicios tales como: bases de datos, alojamiento web, máquinas virtuales, software, servidores, redes, sistemas operativos, dispositivos remotos de prueba [14].

Las empresas están migrando su infraestructura a la nube por varias razones, entre las que destacan: alta disponibilidad, mayor escalabilidad, estandarización en servicios, interoperabilidad entre diferentes tecnologías, fácil acceso a nivel mundial, menor costo, mayor velocidad, mejor rendimiento, confiablidad y seguridad [14], [15].

Para la implementación de todos estos servicios se puede optar por diferentes tipos de nube: pública, privada e híbrida [14].

1.3.3.1 Nube pública

Pertenecen al proveedor de nube encargado de su administración. Todos los recursos y servicios informáticos ofrecidos son accesibles para el usuario a través de Internet. El proveedor es propietario de todo el hardware, software y componentes de la infraestructura. El usuario tiene acceso a los servicios mediante un navegador web [14].

1.3.3.2 Nube privada

Son todos los recursos informáticos en la nube, pertenecientes a una organización, compañía o entidad. Todos estos recursos y servicios se encuentran en una red privada, se puede alquilar su alojamiento a un proveedor de nube o almacenarlos de forma local [14].

1.3.3.3 Nube híbrida

Es una combinación de dos o más nubes públicas y/o privadas, que pueden intercambiar datos y aplicaciones entre ellas [16].

1.3.4 Servicios en la nube

Los servicios brindados por los proveedores de nube se pueden clasificar en tres categorías: software como servicio (SaaS), plataforma como servicio (PaaS) e infraestructura como servicio (IaaS) [15].

En la Figura 1.3 se muestra un modelo de computación en la nube, con las tres categorías de servicios.

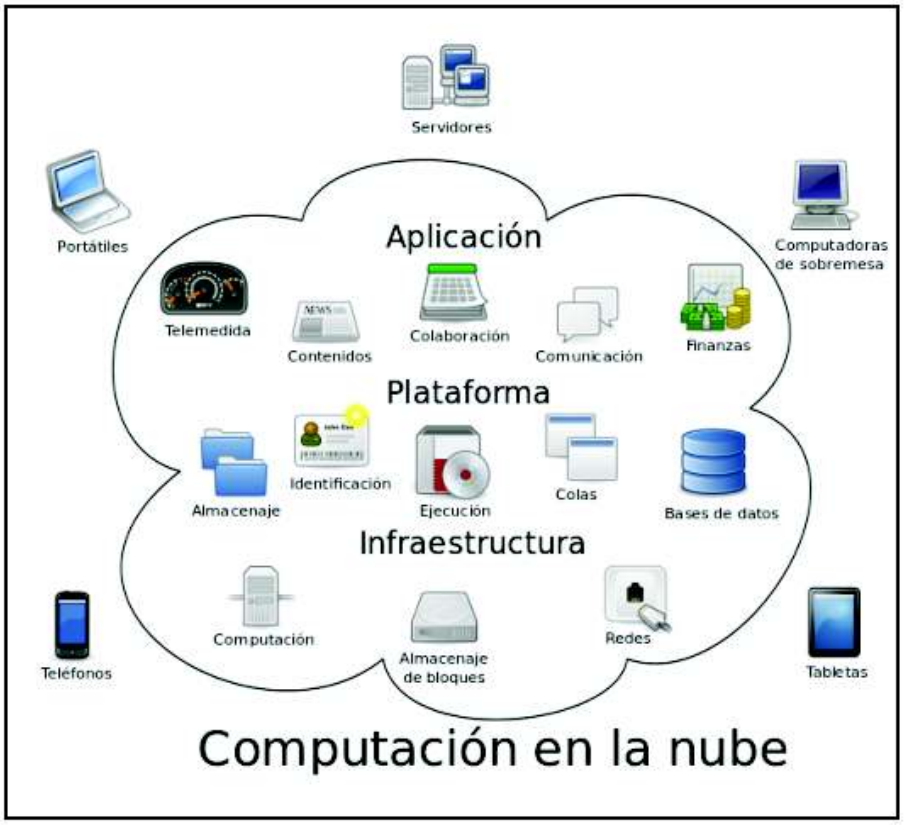


Figura 1.3 Modelo de servicios en la nube [15].

1.3.4.1 Software como servicio (SaaS)

El SaaS (*Software as a Service*), software como servicio permite al usuario tener acceso a aplicaciones remotamente a través de Internet. Puede considerarse la categoría de servicios en la nube usada con mayor frecuencia [15]. Algunos ejemplos de SaaS son:

- Correo electrónico.
- Calendarios.
- Herramientas ofimáticas.
- Google Docs.
- Dropbox.

Todo el mantenimiento del código, actualizaciones, manejo de versiones son responsabilidad de las empresas proveedoras de nube. El usuario final se limita a acceder a los servicios por medio de cualquier navegador web.

1.3.4.2 Plataforma como servicio (PaaS)

PaaS (*Platform as a Service*, Plataforma como Servicio) está enfocada a desarrolladores de software, ya que les permite implementar, probar, administrar y/o actualizar aplicativos móviles, aplicaciones y servicios web; proporcionando herramientas necesarias para estos propósitos. A continuación, se enlistan ejemplos de servicios PaaS [14]:

- Herramientas de desarrollo.
- Administración de bases de datos.
- Servicios de colas.

El usuario de PaaS puede gestionar las aplicaciones desarrolladas, cualquier administración por encima de ese nivel es responsabilidad del proveedor de nube.

1.3.4.3 Infraestructura como servicio (IaaS)

IaaS (*Infrastruture as a Service*, Infraestructura como Servicio) permite el uso de infraestructura TI[3]. Entre los servicios que destacan en esta categoría están [14]:

- Servidores
- Máquinas virtuales

[3] TI, tecnología de la información

- Almacenamiento
- Redes
- Sistemas operativos

El usuario de IaaS puede instalar, administrar y configurar todo el software dentro de los recursos alquilados, la mayoría de los proveedores de nube cobran estos recursos bajo la modalidad pago por uso, es decir se cancela por el tiempo de uso del servicio [14].

1.3.5 Proveedores de servicios en la nube

Son empresas, compañías o entidades que ofrecen servicios orientados a la nube, sea en forma de software, plataforma o infraestructura. Un proveedor puede ofrecer servicios para todas las categorías (SaaS, PaaS, IaaS) o para alguna de éstas. Los principales proveedores de nube son:

- Amazon Web Services.
- Google Cloud Platform.
- Microsoft Azure.

1.3.5.1 Amazon Web Services

AWS (*Amazon Web Services*), proporciona servicios de infraestructura TI desde el año 2006. Brinda servicios de nube a usuarios de 190 países y tiene centros de datos en Estados Unidos, Europa, Brasil, Singapur, Japón y Australia [17].

1.3.5.2 Google Cloud Platform

En 2012, Google agrupó sus servicios de cómputo en la nube bajo el nombre GCP (*Google Cloud Platform*), ofreciendo un conjunto de herramientas TI a todo nivel, estos servicios usan la misma infraestructura perteneciente a aplicaciones como YouTube o el propio buscador de Google [18].

1.3.5.3 Microsoft Azure

En 2010, Microsoft dio a conocer su nueva plataforma de servicios en la nube, al principio ofrecía la categoría PaaS, con el pasar del tiempo se convirtió en uno de los principales proveedores de nube. Posee soluciones para todos los niveles de servicio [19].

Azure tiene alcance global con más de 100 instalaciones de alta seguridad, conectadas bajo una misma red, razón por la cual brinda servicios a 140 países [19].

1.3.6 Arquitectura de la plataforma Android

La plataforma Android está conformada por una pila de software de código abierto sobre el núcleo o kernel Linux. En la Figura 1.4 se observan las partes de la plataforma Android.

Figura 1.4 Pila de software de la plataforma Android [20].

Linux Kernel. - Se encarga de la administración de recursos de hardware, creación de subprocesos y manejo de memoria de bajo nivel. Al basarse en software de código abierto, diferentes marcas de dispositivos móviles adoptan Android como sistema operativo, ya que les facilita el desarrollo de controladores para un kernel conocido [20].

Capa de abstracción de hardware. - También conocida como HAL (*Hardware Abstraction Layer*), está constituida por un conjunto de módulos de biblioteca, esta capa carga la correspondiente librería dependiendo del hardware al que quiera acceder el *framework* de alguna API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) [20].

Bibliotecas C/C++ nativas. - Aunque las aplicaciones Android están escritas mediante las API Java, las implementaciones de éstas pueden estar escritas en lenguaje C/C++. La JNI (*Java Native Interface*, Interfaz Nativa Java) define la forma de administrar el código escrito en lenguaje Java[4] o Kotlin[5] para que interactúe con la programación C/C++ [20], [21].

ART.- (*Android Runtime*, Tiempo de ejecución Android). Cada aplicación ejecuta sus procesos dentro de una instancia ART, que le permite al sistema correr varias máquinas virtuales a la vez. Android utiliza archivos DEX para ejecutar las diferentes instancias ART, estos archivos contienen códigos de bytes, diseñados específicamente para la plataforma Android, optimizando la memoria del dispositivo. Otra funcionalidad de ART es generar puntos de control e informes de fallo en el proceso de depuración, así como gestionar los recursos no utilizados [20].

Java API Framework. - Tanto las aplicaciones del propio sistema operativo Android como las instaladas por el usuario tienen acceso a las mismas API Framework de Java, éstas facilitan la reutilización de ciertos componentes del sistema. Existen algunas API Framework Java con funcionalidades específicas como: controlar los elementos de la interfaz gráfica de la aplicación, gestionar el acceso a recursos, controlar las notificaciones de la barra de estado, administrar el ciclo de vida de la aplicación, permitir el intercambio de datos entre diferentes aplicaciones [20].

Aplicaciones del sistema. – La plataforma Android incluye un conjunto de aplicaciones preinstaladas, propias del sistema: calculadora, reproductor multimedia, correo electrónico, mensajería SMS, calendarios, navegador web, alarma, agenda, calendario, etc. Estas aplicaciones tienen igual prioridad que las instaladas por el usuario, también se puede

[4] Lenguaje de programación de alto nivel, enfocado a objetos y desarrollado por Sun Micro-systems.
[5] Lenguaje de programación de tipado estático, desarrollado por JetBrains.

intercambiar datos entre ellas, a través de la API proveedores de contenido (*Content Providers*) [20].

1.3.6.1 Interfaz de usuario

Es el conjunto de todos los componentes gráficos que pueden ser vistos en pantalla, ayudan a la interacción usuario-aplicación. Existen varios elementos que forman parte de la IU (Interfaz de Usuario), entre los principales controles se tiene:

Botón (*Button*). – El usuario puede tocar este control para realizar alguna acción previamente establecida. Los botones pueden contener texto, iconos, imágenes o una combinación de estos. En la Figura 1.5 se visualizan diferentes tipos de botones.

Figura 1.5 Tipos de botones en Android [22].

Interruptor (*Switch*). – Control que permite cambiar entre dos estados, dependiendo de la selección se ejecutarán acciones diferentes. Los estados preestablecidos son de encendido (*on*) y apagado (*off*). En la Figura 1.6 se observan los estados que puede adoptar un interruptor.

Figura 1.6 Captura de pantalla de los estados de un interruptor [23].

Vista de texto (*TextView*). – Elemento que posibilita mostrar información al usuario en forma de texto, el contenido de este control puede ser estático o dinámico. En caso de ser dinámico, los cambios en el texto estarán administrados por código. En el lado izquierdo de la Figura 1.7 se observan dos vistas de texto, contienen información estática con las etiquetas “Nombre:” y “Edad:” respectivamente.

Figura 1.7 Captura de pantalla con controles tipo texto.

Editar texto (*EditText*). – Mediante este control, el usuario es capaz de introducir datos a la aplicación en forma de texto. La información puede ser ingresada mediante el teclado o a través del micrófono. Por código se puede establecer el número de caracteres a ingresar, así como el tipo de datos que manejará este control. En el lado derecho de la Figura 1.7 se ven dos controles editar texto con la información “Manuel Pérez” y “25” respectivamente, datos ingresados por el usuario.

Vista de lista (*ListView*). – Este control permite la visualización de una lista vertical de elementos, que pueden ser: textos, imágenes, iconos o alguna combinación entre estos. La lista es desplazable (*scrollable*), se pueden añadir elementos que se ubicarán al final. Al presionar algún ítem, se puede asociar alguna acción, previamente establecida por el desarrollador [24]. En la Figura 1.8 se visualizan los nombres de países dentro de una vista de lista.

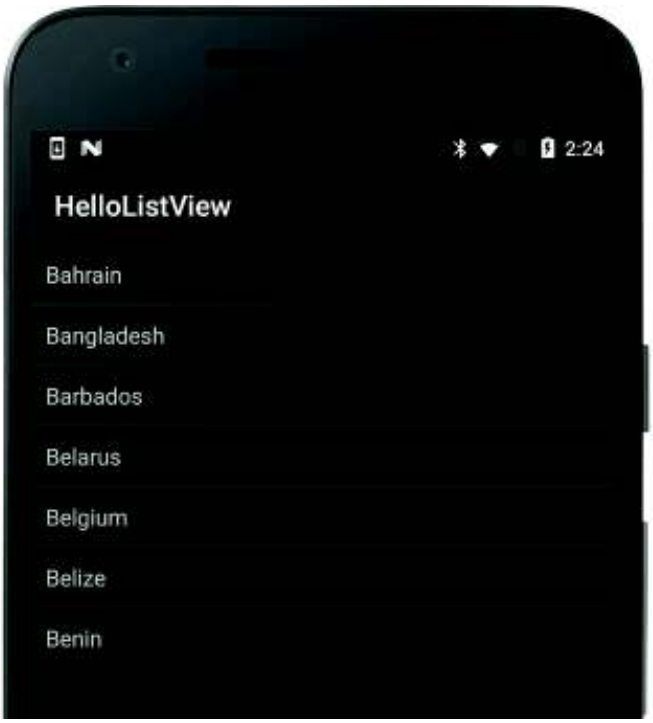


Figura 1.8 Captura de pantalla de un ejemplo de vista de lista [25].

Diálogo de alerta (*AlertDialog*). – Son cuadros de texto que contienen mensajes, con la finalidad de realizar alguna acción que permita interactuar con la aplicación. Estos mensajes pueden incluir título, contenido y hasta tres botones. Para administrar las acciones a seguir después de la visualización de las alertas, estos cuadros ocupan parte de la pantalla [26]. En la Figura 1.9 se observa un ejemplo de diálogo de alerta, se solicita

activar el GPS (*Global Positioning System*, Sistema de Posicionamiento Global). El usuario puede aceptar o cancelar esta acción usando los botones incluidos en la alerta.

Figura 1.9 Captura de pantalla de ejemplo de un diálogo de alerta.

Control de número (*Spinner*). – Permite seleccionar un ítem entre una lista de elementos. Después de realizar la selección, adquiere el valor de dicho elemento, al volver a pulsar sobre el control se despliega nuevamente la lista [27].

Figura 1.10 Captura de pantalla de ejemplo de un control de número [28].

En la Figura 1.10 se visualiza el funcionamiento de un control de número, al pulsar el control se despliega la lista de los planetas del sistema solar, cuando se elige un planeta, la lista se comprime, y el control toma como valor el nombre del planeta seleccionado.

Mensaje de notificación (*Toast*). – Son pequeños mensajes que aparecen por un instante en la pantalla, sin interrumpir la vista actual. Se visualizan en una ventana emergente semi transparente, ocupando el tamaño del mensaje. Dan información al usuario sobre la ocurrencia de cierto evento o finalización de alguna acción [29]. En la Figura 1.11 se ve un mensaje de notificación con el contenido “Registro guardado”.

Figura 1.11 Captura de pantalla de ejemplo de un mensaje de notificación.

Barra de progreso (*ProgressBar*). – Es un elemento que permite visualizar el avance de alguna operación o proceso dentro de la aplicación. Cuando el tiempo de duración del proceso es indefinido, se puede configurar para que la barra de proceso siga un ciclo repetitivo, dando al usuario la sensación de que se está realizando alguna operación [30].

Figura 1.12 Captura de pantalla de ejemplo de barra de progreso [31].

Barra de búsqueda (*SeekBar*). – Es un elemento similar a la barra de progreso, con la diferencia que tiene un pequeño indicador deslizable. Provee mejor visualización del avance del proceso, también permite cambiar la posición del progreso de la barra [32].

En la Figura 1.13 se observa el funcionamiento de una barra de búsqueda, el indicador avanza conforme se escucha el audio del reproductor multimedia.

Figura 1.13 Captura de pantalla de ejemplo de barra de búsqueda.

Notificaciones locales. – En Android existe la barra de notificaciones, en la cual se pueden observar pequeños iconos, ubicados a la izquierda de ésta, correspondientes a las alertas generadas por cada aplicación. Para ver más detalles sobre alguna notificación se puede deslizar la barra hacia abajo [33].

Figura 1.14 Captura de pantalla de ejemplo de una barra de notificaciones [33].

Los elementos de una notificación pueden ser: título, contenido, icono y marca de tiempo. En la Figura 1.14 se ven tres notificaciones con sus respectivos elementos.

Figura 1.15 Captura de pantalla de la barra de notificaciones desplazada [33].

1.3.6.2 Diseño de pantallas

En Android se tiene un diseño (*layout*) por cada pantalla, contiene los elementos de la interfaz de usuario que se visualizarán y la forma en la que están organizados entre sí. El sistema operativo Android usa XML (*eXtensible Markup Language*, Lenguaje de Marcado Extensible), basado en etiquetas, esto permite separar la parte gráfica de la aplicación del código de programación [34].

Existen dos opciones para editar la parte gráfica de la aplicación: Trabajar directamente con el archivo XML, que contendrá etiquetas para cada elemento IU o añadir elementos desde la vista gráfica. Estas dos formas de manipular los elementos están relacionadas entre sí, cualquier cambio en el código XML afectará directamente a la vista gráfica y viceversa.

A continuación, se detallan algunas propiedades del vocabulario XML Android, observadas en la Figura 1.16, el ancho (*layout_width*) y alto (*layout_height*) de los controles pueden adoptar el tamaño del contenedor padre (*match_parent*) o el tamaño del contenido del elemento (*wrap_content*).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android'
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/grabarButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Grabar" />
    <Button
        android:id="@+id/pararButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:enabled="false"
        android:text="Parar" />
    <Button
        android:id="@+id/reproducirButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:enabled="false"
        android:text="Reproducir" />
    <TextView
        android:text="Ejemplo vista de texto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="5dp"
        android:id="@+id/textView1" />
</LinearLayout>
```

Figura 1.16 Captura de pantalla del archivo XML de un diseño de pantalla.

Ciertas propiedades vienen preestablecidas, por ejemplo, los botones están activados por defecto, si el desarrollador desea desactivarlos, puede establecer en falso la propiedad habilitado (*enabled*), dentro de la etiqueta correspondiente.

La propiedad texto (*text*) del botón permite establecer la información a visualizar dentro del mismo. La propiedad id de cada controlador da un identificador único, el cual será utilizado para establecer una relación con la parte programática de la aplicación.

El margen con respecto al controlador padre se establece mediante la propiedad relleno (*padding*), acompañado de las palabras arriba, abajo, izquierda o derecha; de acuerdo con las necesidades del diseño. Existen muchas propiedades específicas para cada control IU, pero son de fácil entendimiento, ya que su propio nombre da una breve idea de cuál es su finalidad.

En la Figura 1.17 se observa la parte gráfica correspondiente al archivo XML de la Figura 1.16.

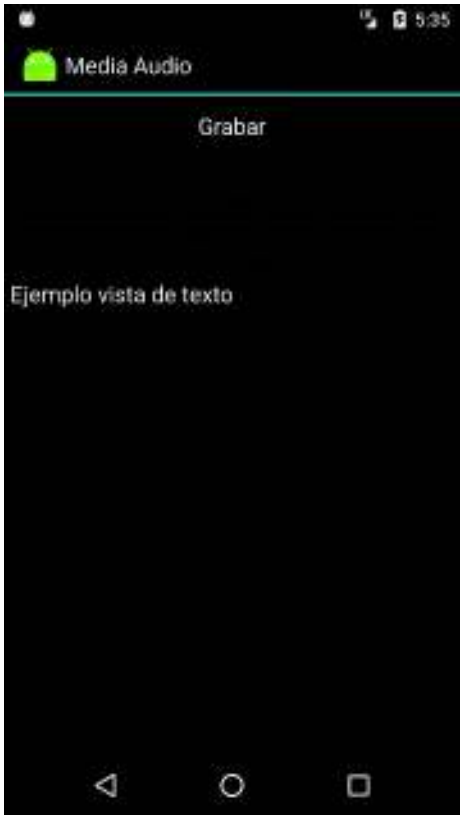


Figura 1.17 Captura de la vista gráfica de un diseño de pantalla.

Diseño lineal (*LinearLayout*). – Es un contenedor de elementos de IU, los cuales pueden estar orientados de forma vertical u horizontal, un diseño lineal puede contener otros contenedores del mismo tipo dentro de sí mismo. La orientación de este control se la fija con la propiedad orientación (*orientation*) de la etiqueta correspondiente al elemento en el archivo XML [35].

Los controles del ejemplo de la Figura 1.17 están dentro de un diseño lineal, todos los elementos están contenidos en la etiqueta XML, cuya orientación es vertical, ver Figura 1.16.

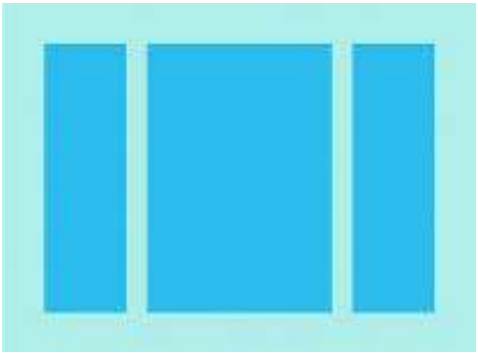


Figura 1.18 Esquema de diseño lineal horizontal [35].

En la Figura 1.18 se ven tres elementos IU dentro de un diseño lineal, la orientación de este es horizontal.

Existen otros contenedores, tales como: diseño relativo (*RelativeLayout*) y de tabla (*TableLayout*). Los elementos ocupan una posición relativa con respecto a otros o se organizan en filas y columnas respectivamente.

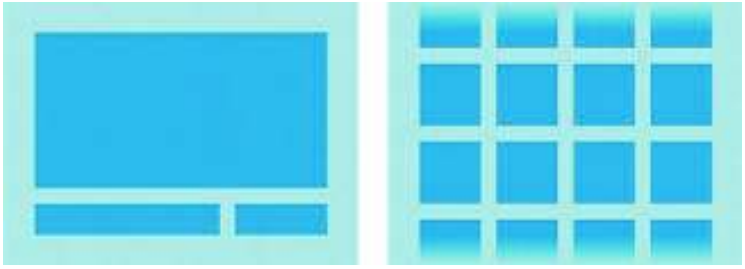


Figura 1.19 Esquemas de diseño relativo y tabla [34].

1.3.6.3 Ciclo de vida de una actividad

Un componente fundamental en el desarrollo de aplicaciones móviles Android es el de actividad (*activity*). Todo lo observado en pantalla pertenece a la misma actividad, es decir que se relaciona con el diseño de pantalla en específico. Además de la experiencia de usuario, también incluye el código de programación necesario para interactuar con dicha pantalla. Una actividad puede cambiar de estado, las diferentes fases que se pueden adoptar conforman el ciclo de vida de esta [36], [37].

Android administra el cambio de estados de actividad a través de métodos de devolución de llamada (*callbacks*), estos permiten al desarrollador mejorar la administración del código a ejecutarse cuando se presenten estas condiciones.

En la Figura 1.20 se observa un diagrama de flujo de los métodos generados en el cambio de estado de actividad. A continuación, una breve descripción de cada uno de ellos.

***OnCreate*. –** Al iniciar una actividad se ejecuta el código contenido en este método, aquí se cargan vistas gráficas, relaciones entre elementos IU y sus ID para poder ser utilizados en código de programación.

***OnStart*. –** Código que se ejecuta inmediatamente después del método *OnCreate*, también se puede ejecutar luego de *OnRestart*. La programación de *OnStart* se invoca justo antes de que la actividad sea visible para el usuario.

***OnResume*. –** En el instante que se invoca el código de este método, la actividad es visible. Se ejecutará después de *OnStart*, o si previamente la actividad estaba en pausa se invocaría después de *OnPause*.

***OnPause*. –** Se ejecuta este método cuando la actividad está a punto de ser remplazada por otra. El código debe ser ligero para que la transición entre actividades sea rápida y transparente al usuario. Desde aquí se puede ir a los métodos *OnResume* u *OnStop*, dependiendo si la actividad actual vuelve a primer plano o si será reemplazada por otra, respectivamente.

***OnStop*. –** El código de este método se ejecuta cuando la actividad ya no es visible para el usuario, es decir, ésta ha pasado a segundo plano (*background*).

***OnRestart*. –** Método que se ejecuta cuando la actividad pasa de segundo a primer plano.

***OnDestroy*. –** Este método es invocado cuando la actividad está a punto de ser eliminada por completo, ejecutado este método la aplicación desaparece por completo. Por lo común la programación de este método se utiliza para liberar recursos.

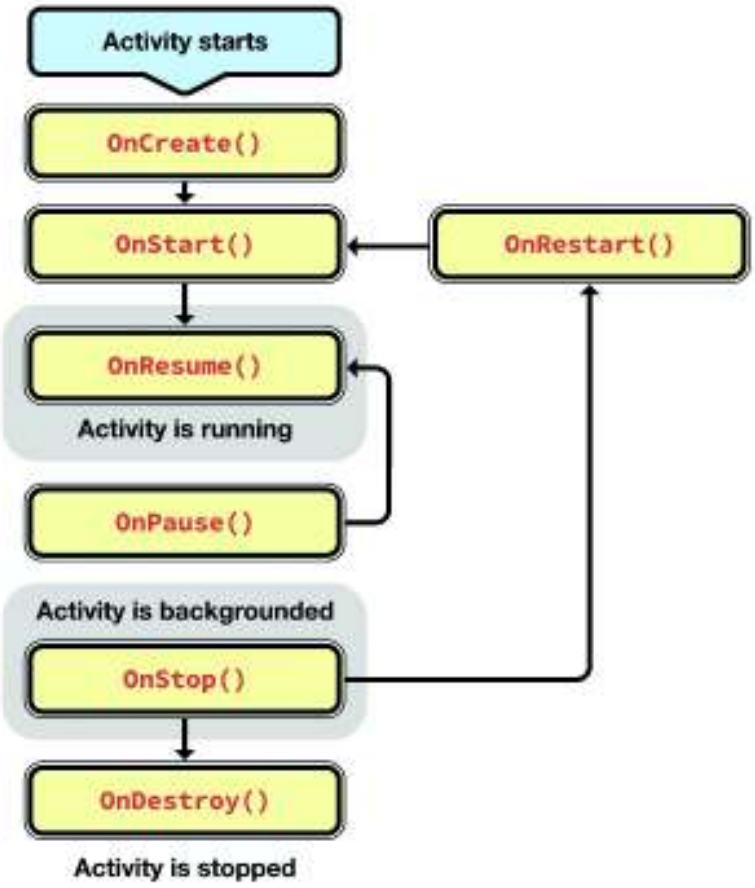


Figura 1.20 Métodos del ciclo de vida de una actividad [36].

El método *OnCreate* siempre se ejecuta, los otros pueden o no ser implementados dependiendo de los requerimientos de la aplicación y del criterio del desarrollador para satisfacer las mismas.

Cuando una actividad se destruye, se pierden los valores de las variables locales, surge la necesidad de buscar alternativas para preservar estos datos, si son necesarios. Para esto se pueden usar variables globales o recurrir al almacenamiento interno del dispositivo móvil.

Las variables globales se deben usar con cuidado, tomando en cuenta que los recursos de memoria RAM del dispositivo son limitados en comparación a los de un computador. Se considera buena práctica usarlas cuando el contenido de éstas será utilizado en gran parte de la aplicación.

El almacenamiento interno es utilizado para preservar datos más allá del tiempo de vida de la aplicación. Las variables globales se pierden cuando se cierra la aplicación, las locales cuando finaliza la actividad a la que pertenecen.

1.3.6.4 Servicios

El sistema operativo Android permite ejecutar código de programación sin necesidad de asociarlo a una vista gráfica, a través de la implementación de servicios, los cuales podrían correr en segundo plano. Dependiendo de los requerimientos del desarrollo se puede configurar el servicio para ejecutarse inclusive después de cerrar la aplicación, en caso de ser necesario [38].

Un servicio se puede iniciar de dos formas, a través de llamadas de inicio (*startService*) o mediante enlace (*bindService*), depende de la finalidad de su implementación. Si se necesita realizar algo de forma separada, sin mayor interacción con componentes de la aplicación, se usa el primer tipo de llamado, caso contrario, cuando se requiere alta interacción, se utiliza el llamado de enlace de servicio. Un servicio iniciado por la llamada de inicio puede ejecutarse indefinidamente, razón por la cual, es necesario implementar finalizaciones a través de código de programación [38].

1.3.6.5 Manifiesto de Android

Toda aplicación Android tiene un archivo de tipo XML con el nombre *AndroidManifest*, ubicado en la dirección raíz del sistema. La información de este archivo es de vital importancia para el sistema operativo, ya que le permite ejecutar todo el código de programación [39].

En [39] se mencionan las principales funcionalidades del manifiesto de Android:

- Nombrar con un identificador único al paquete Java de la aplicación.
- Describir componentes de la aplicación, tales como: servicios, actividades, proveedores de servicios, receptores de mensajería. Para cada elemento, nombra la clase que lo implementa.
- Declarar permisos que debe tener la aplicación para acceder al contenido protegido de las API.
- Declarar permisos que terceros deben poseer para interactuar con los componentes de la aplicación.
- Especificar el nivel mínimo de API que necesita la aplicación.

- Listar las bibliotecas requeridas por la aplicación.

1.3.6.6 Permisos

Cada aplicación Android se ejecuta en su propio espacio de trabajo, por cuestiones de seguridad existen restricciones para acceder a ciertos recursos del sistema, también se restringe el acceso a los componentes de hardware del dispositivo. Los permisos requeridos para el correcto funcionamiento de la aplicación deben ser declarados en el manifiesto de Android [40].

En sistemas Android 6.0 o superiores, el usuario puede administrar los permisos inclusive después de la instalación de la aplicación [40]. En la Figura 1.21 se observa un ejemplo de configuración de permisos para un dispositivo con sistema Android 6.0 o superior.

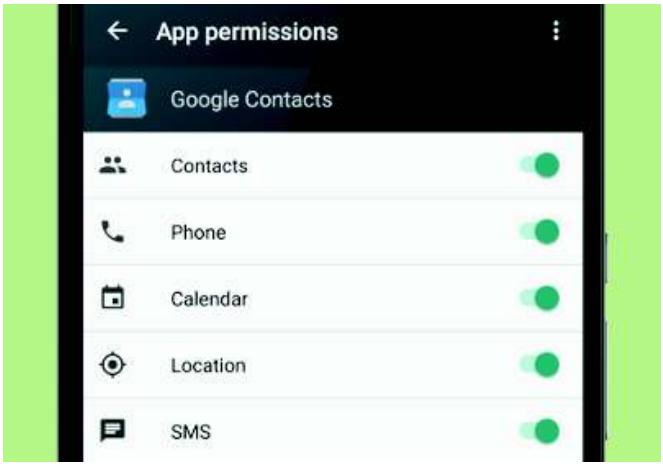


Figura 1.21 Captura de pantalla de los permisos de la aplicación *Google Contacts* [40].

Los permisos son declarados por el desarrollador en el archivo manifiesto de Android y son aceptados por el usuario al momento de instalar la aplicación. La lista de permisos desplegados al usuario son solo aquellos declarados explícitamente en la fase de desarrollo.

1.3.6.7 Nivel de API

El nivel de API es un número entero, asignado a cada versión de la plataforma Android, identificando de manera exclusiva a cada Framework API del sistema. En la Tabla 1.6 se especifica el número de nivel API de las principales versiones del sistema operativo Android [41].

Tabla 1.6 Nivel de API compatible con la versión de la plataforma [41].

Versión de la plataforma Android	Nivel de API
Ice Cream Sandwich 4.0	15
Jelly Bean 4.1	16
Jelly Bean 4.2	17
Jelly Bean 4.3	18
KitKat 4.4	19
KitKat Wear 4.4w	20
Lollipop 5.0	21
Lollipop 5.1	22
Marshmallow 6.0	23
Nougat 7.0	24
Nougat 7.1.1	25
Oreo 8.0	26
Oreo 8.1	27
Pie 9.0	28

El nivel de API es usado para especificar el nivel de compatibilidad de una aplicación, en el archivo de manifiesto Android se pueden definir tres atributos para especificar esta compatibilidad. Los atributos por definir son: *minSdkVersion*, *targetSdkVersion*, *maxSdkVersion*. Aunque el nombre podría sugerir que estos atributos sirven para definir la versión de SDK (*Software Development Kit*, Kit de Desarrollo de Software), no es así, solo especifican el nivel de API de compatibilidad [41].

El *minSdkVersion* es el valor mínimo de nivel de API en el que se puede instalar y ejecutar la aplicación El *targetSdkVersion* es el nivel de API al cual está dirigida la aplicación, este valor debe ser mayor o igual al *minSdkVersion.* El *maxSdkVersion* es el nivel máximo de API para el cual la aplicación podría ejecutarse, la documentación oficial de Android recomienda no definir este valor [41].

1.3.7 Desarrollo de aplicaciones móviles con .NET

Para desarrollar aplicaciones móviles sobre .NET[6] se usa Xamarin como un conjunto de herramientas y bibliotecas que facilitan este proceso. Xamarin permite desarrollo nativo o multiplataforma[7] a través de un solo lenguaje de programación. Para el desarrollo nativo se utiliza Xamarin.Android o Xamarin.iOS, dependiendo del sistema operativo del dispositivo móvil. Para el desarrollo multiplataforma se usa Xamarin.Forms, el lenguaje de programación para todos los casos es C# [42].

[6] Plataforma Microsoft para el desarrollo de software.
[7] Aplicación informática que puede ejecutarse en diferentes sistemas operativos.

1.3.7.1 Xamarin para desarrollo nativo

Con respecto al desarrollo móvil se tienen dos sistemas operativos principales, iOS y Android. Tanto Xamarin.iOS como Xamarin.Android se basan en una versión .NET Framework de código abierto llamada Mono, que puede ejecutarse en plataformas diferentes a la de Microsoft, por ejemplo: Unix, Mac OS X, Linux, entre otras.

En sistemas Android, Xamarin utiliza IL (*Intermediate Language*, Lenguaje Intermedio), cuando se lanza la aplicación, se pasa a ensamblados nativos. A esta forma de compilar se la conoce como JIT (*Just In Time*, Justo a Tiempo).

En plataformas iOS, Xamarin usa el compilador AOT (*Ahead Of Time*, Antes de Tiempo), le permite compilar directamente en ensamblados nativos ARM[8].

Las aplicaciones nativas desarrolladas en Xamarin se compilan en librerías BCL (*Base Class Library*, Biblioteca de Clases Base), cada plataforma Android o iOS empaqueta estas librerías en diferentes archivos, Mono.Android.dll y MonoTouch.dll respectivamente. Estos archivos también contienen contenedores para los SDK de Android e iOS, razón por la cual, se pueden invocar estas API desde código C#.

Xamarin genera archivos ejecutables para Android e iOs de extensión *.apk* y *.app* respectivamente, estos paquetes tienen las mismas extensiones que paquetes generados con el IDE (*Integrated Development Environment*, Entorno de Desarrollo Integrado) de cada plataforma. Los paquetes generados por Xamarin son indistinguibles de los propios a cada plataforma [42].

1.3.7.2 Xamarin para desarrollo multiplataforma

Para el desarrollo multiplataforma, Xamarin utiliza Xamarin.Forms, que es una abstracción de capa de IU, le permite al desarrollador crear interfaces basadas en controles nativos por plataforma. Para poder compartir código entre distintas plataformas se pueden utilizar dos paradigmas: PCL (*Portable Class Library*, Biblioteca de Clase Portable) o SAP (*Shared Asset Project*, Proyecto de Archivos Compartidos).

Adicional al código compartido se necesita crear aplicaciones específicas de cada plataforma [43]. En el código compartido se pueden incluir las siguientes tareas: conexiones a bases de datos, interacción con servidores externos, uso de servicios de terceros, entre otras. Aproximadamente se puede compartir del 70 al 80 % del código.

[8] Arquitectura ARM (*Advanced RISC Machine*), utilizada por dispositivos iOS.

2. METODOLOGÍA

En el presente proyecto se desarrolla un prototipo multimedia que permite el registro oral de la lengua quichua, para este fin se utilizará una aplicación Android, todos los datos y metadatos serán almacenados tanto en el almacenamiento interno del dispositivo como en la nube. Para una visualización más fácil de datos y metadatos se creará un archivo ofimático que esté sincronizado con la base de datos. El proyecto pretende la creación de una herramienta que facilite el trabajo de investigadores lingüistas.

Para el diseño gráfico de la aplicación se trabajará con bosquejos (*sketches*) de todas las actividades, las mismas que serán diseñadas en base a los requerimientos del sistema. El código de todo el prototipo será desarrollado en base a diagramas de: clases, casos de uso, navegación de actividades y secuencia. Todos estos elementos serán tratados con detalle en la parte de diseño del prototipo y orientados a los requerimientos del sistema.

2.1 Requerimientos del sistema

El sistema ha sido dividido en tres módulos, a continuación, se detallan los requerimientos de cada uno:

2.1.1 Aplicación Android

El usuario de la aplicación deberá ingresar datos informativos que identifiquen a la comunidad indígena, al hablante nativo y a la persona que recolecte los datos. Los datos que identificarán a una comunidad serán los siguientes:

- Nombre de la comunidad.
- Pueblo al que pertenece.
- Provincia.
- Región.
- Latitud.
- Longitud.

Para los hablantes nativos se recolectarán los siguientes datos:

- Nombre.
- Edad.
- Género.

La persona que realice la adquisición de datos se llamará recolector de datos y deberá identificarse con:

- Nombre.
- Género.
- Identificación.

La aplicación realizará registros, estos estarán conformados por archivos de audios, los mismos que serán generados a través de grabaciones, que pueden ser de: fonemas, palabras, frases, oraciones y conversaciones de la lengua nativa. Cada registro debe contener datos informativos de la comunidad, hablante y recolector; previamente ingresados por el usuario. A parte de esta información los registros deben identificarse con los siguientes datos adicionales:

- Tipo de registro.
- Escritura en castellano.
- Escritura en nativo.
- Fecha de creación.
- Hora de creación.
- Latitud y longitud del lugar en que se realizó el registro.
- Duración de la grabación.

Todos los datos y metadatos de registros serán guardados en el almacenamiento interno del dispositivo móvil, luego se enviarán de manera automática a la base de datos. El usuario podrá ver el avance de subida de los archivos y también podrá visualizar cuales ya se encuentran en la base de datos.

El usuario de la aplicación podrá reproducir los registros previamente guardados, mientras otros se estén subiendo a la nube, inclusive podrá realizar nuevos registros, todas estas tareas deben ser transparentes para el usuario. También podrá pausar la sincronización, para ahorrar datos móviles hasta disponer de una conexión wifi.

2.1.2 Almacenamiento en la nube

Todos los archivos de audio, datos y metadatos generados por la aplicación Android serán almacenados en la nube pública. Para los archivos de audio se utilizará almacenamiento tipo BLOB (*Binary Large Object*, Objeto Binario Grande). Los datos y metadatos serán guardados en la nube a través de una base de datos de tipo no relacional.

Tanto archivos como datos deberán ser enviados desde el dispositivo hasta la nube a través de la conexión de Internet del dispositivo, este proceso se realizará mediante el protocolo de transferencia seguro HTTPS (*Hyper Text Transfer Protocol Secure*, Protocolo Seguro de Transferencia de Hipertexto).

2.1.3 Visualización de datos y metadatos desde la nube

El sistema también deberá contar con un punto terminal para visualización de datos, que sea amigable y de fácil uso para el usuario. Podría ser el caso que el investigador lingüista no maneje conceptos y habilidades técnicas relacionados con el uso y manipulación de bases de datos, ya que estos conocimientos no son propios de su formación académica.

Paro estos fines se usará un archivo de hoja de cálculo Excel, el cual deberá sincronizarse con la base de datos, las actualizaciones serán de una sola vía, manteniendo la integridad de la información recopilada.

2.2 Diseño del prototipo

Luego de revisar los requerimientos del sistema de cada módulo, se procede con la elaboración de varios diagramas UML[9] *(Unified Modeling Language*, Lenguaje Unificado de Modelado), que servirán de punto de partida para la implementación del prototipo.

2.2.1 Diagrama de Casos de Uso

Los diagramas de casos de uso describen la funcionalidad del sistema y sirven para mejorar la comprensión de cómo interactúa el usuario con la aplicación Android. En base a los requerimientos se realizarán cuatro diagramas de casos de uso, que son:

- Metadatos.
- Nuevo registro.
- Reproducir registros.
- Envío de datos a la nube.

[9] Permiten realizar representaciones de modelos de sistemas de software.

Adicional a estos diagramas de casos de usos, se desarrollará otro que describa la funcionalidad del módulo que corresponde a la visualización de datos y metadatos desde la nube.

A todos los sistemas externos y/o usuarios que interactúen con el sistema se los denomina actores, representados por un icono de persona dentro del diagrama de casos de uso.

2.2.1.1 Diagrama de casos de usos Metadatos

En la Figura 2.1 se observa el diagrama de casos de uso correspondiente al subsistema metadatos de la aplicación Android, está conformado por tres casos de usos principales que son Ingresar metadatos, Visualizar metadatos y Visualizar detalles.

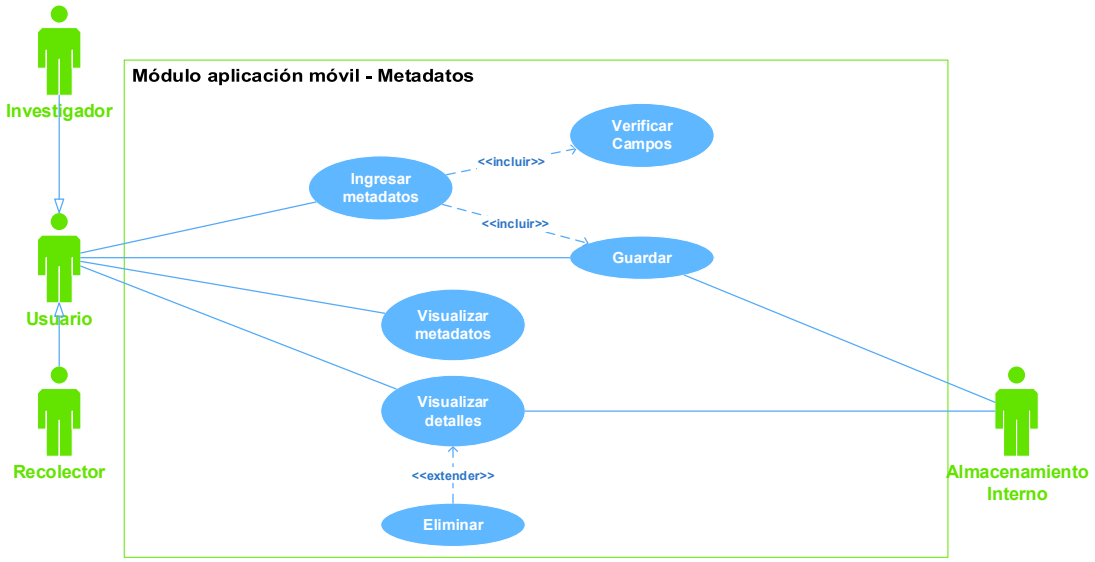


Figura 2.1 Diagrama de casos de usos del subsistema Android – Metadatos

Ingresar metadatos posibilitará que el usuario agregue todos los metadatos sobre las comunidades, hablantes y usuarios. Ingresar metadatos incluye los casos de usos Verificar campos y Guardar. Verificar campos permitirá la validación de todos los metadatos ingresados y Guardar registrará toda la información en el almacenamiento interno del dispositivo.

Visualizar metadatos permitirá al usuario ver las comunidades, hablantes y usuarios previamente ingresados. Visualizar detalles permitirá ver los metadatos de la comunidad, hablante o usuario seleccionado. Visualizar detalles extiende al caso de uso Eliminar, que

permitirá borrar del almacenamiento interno al campo seleccionado; incluyendo sus metadatos.

2.2.1.2 Diagrama de casos de usos Nuevo registro.

En la Figura 2.2 se ve el diagrama que pertenece al subsistema Nuevo registro de la aplicación Android, está formado por tres casos de uso principales que son: Seleccionar metadatos, Ingresar datos y Grabar audio.

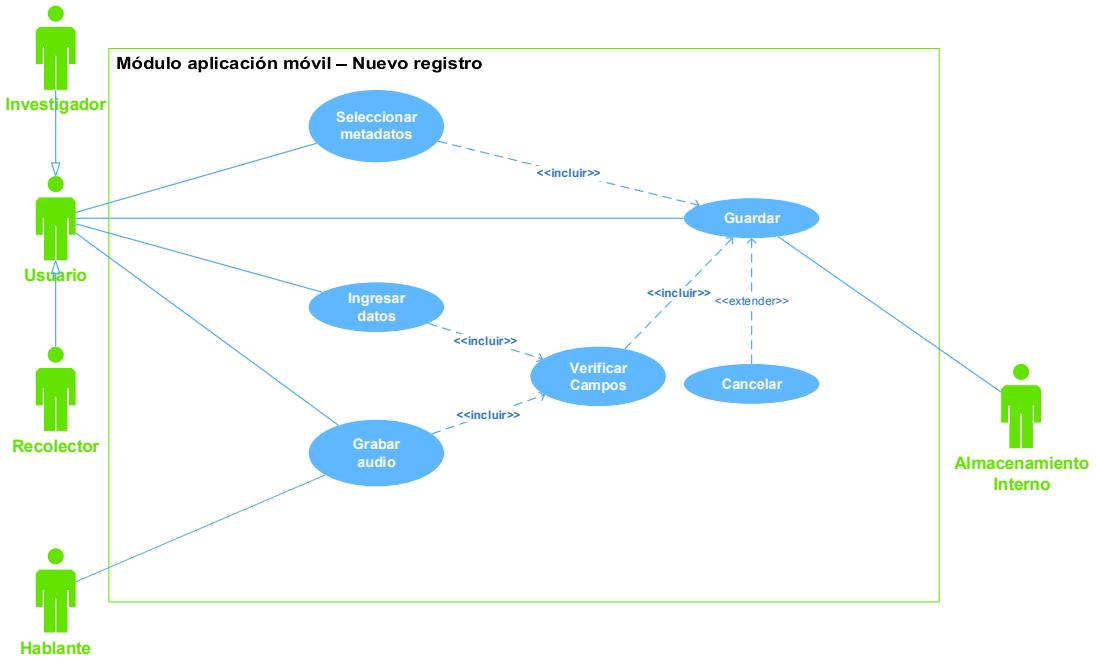


Figura 2.2 Diagrama de casos de usos del subsistema Android - Nuevo registro

Seleccionar metadatos permitirá al usuario seleccionar la comunidad, hablante y usuario relacionados al nuevo registro con sus respectivos metadatos. Ingresar datos facilitará el ingreso de información propia de cada registro y Grabar audio realizará la adquisición de voz del hablante nativo.

Tanto Ingresar datos como Grabar audio incluyen el caso de uso Verificar campos que permitirá la comprobación del contenido de información ingresada. Seleccionar metadatos no incluye Verificar campos, como se ve en la Figura 2.1, la validación de esa información se realiza en el ingreso de metadatos.

Seleccionar metadatos y Verificar campos incluyen el caso de uso Guardar, que permitirá almacenar el registro y toda su información en el almacenamiento interno del dispositivo. Guardar extiende al caso de uso Cancelar que será usado para anular la acción de guardar el registro.

2.2.1.3 Diagrama de casos de usos Reproducir registros

En la Figura 2.3 se observa el diagrama correspondiente al subsistema de la aplicación Android, permitirá reproducir los registros locales, contiene el caso de uso principal Visualizar registros, que permitirá al usuario ver una lista con todos los registros.

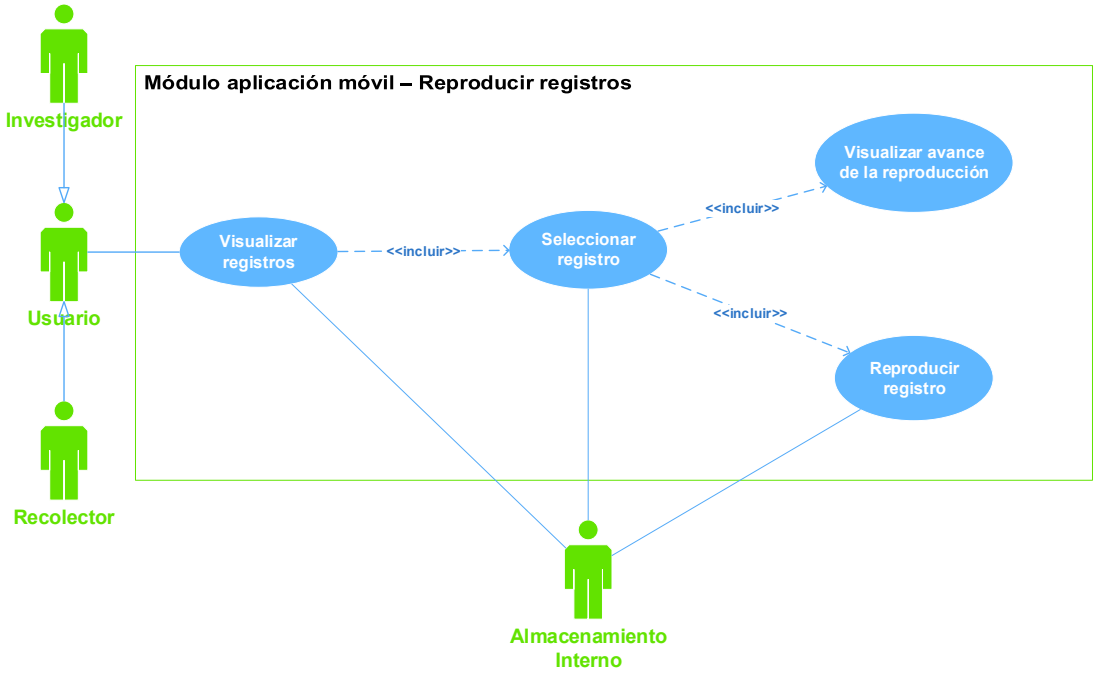


Figura 2.3 Diagrama de casos de usos del subsistema Android - Reproducir registros

Visualizar registros incluye al caso de uso Seleccionar registro, que permitirá al usuario escoger un elemento de toda la lista de registros. Seleccionar registro incluye los casos de uso Visualizar avance de la reproducción y Reproducir registro.

Visualizar avance de la reproducción ayudará a constatar su progreso y Reproducir registro se encargará de reproducir del archivo de audio asociado al registro.

2.2.1.4 Diagrama de casos de usos Envío de datos a la nube

En la Figura 2.4 se ve el diagrama de casos de usos correspondiente a la aplicación Android que facilitará el proceso de envío de registros y toda su información a la nube. El diagrama está formado por dos casos de usos principales: Iniciar sincronización y Parar sincronización.

Iniciar sincronización incluye dos casos de uso. Subir audios enviará los archivos que contienen los sonidos nativos al almacenamiento tipo BLOB, que estará alojado en la nube. Subir metadatos enviará la información correspondiente a cada registro a la base de datos de tipo no relacional, almacenada en la nube.

Para la gestión de nombres únicos, tanto para los archivos de audio, como para sus metadatos se usará un servicio de mensajería de cola, alojado en la nube.

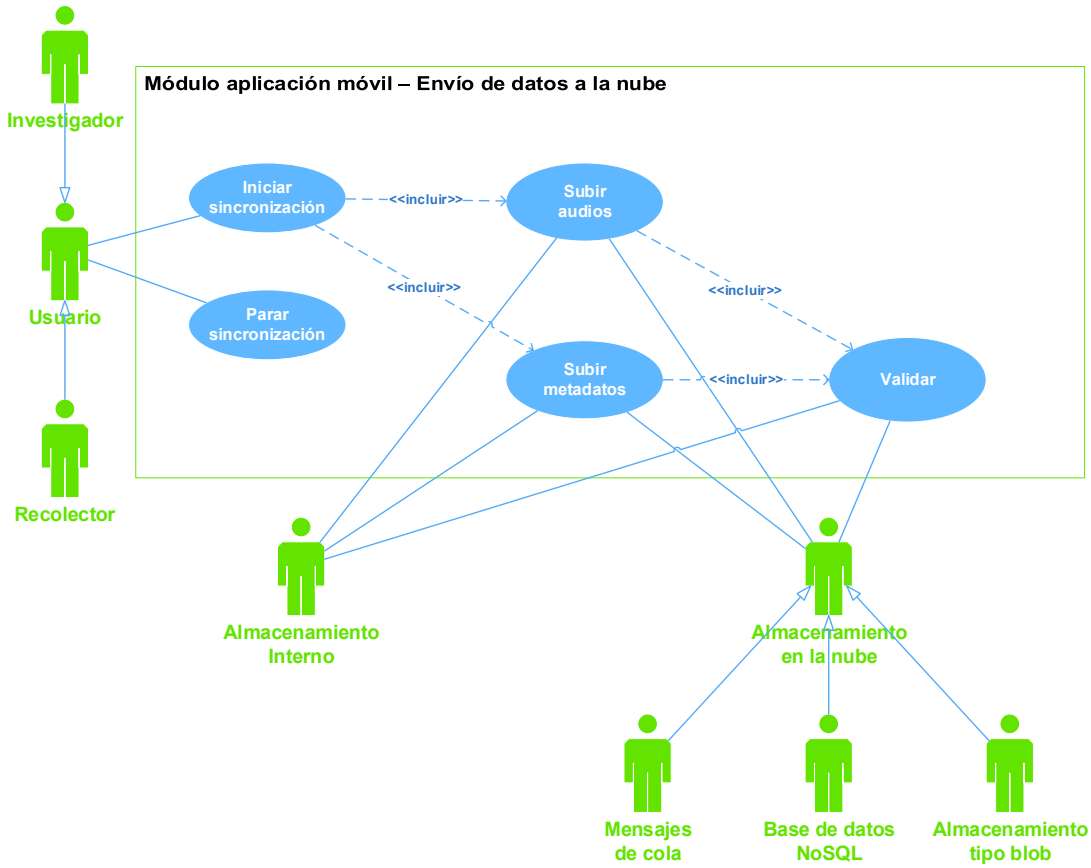


Figura 2.4 Diagrama de casos de usos del subsistema Android – Envío de datos a la nube

2.2.1.5 Diagrama de casos de usos Herramienta ofimática

En la Figura 2.5 se observa el diagrama de casos de usos correspondiente a la visualización de datos y metadatos desde la nube. El diagrama tiene tres actores principales que son: Investigador, Almacenamiento en la nube y Almacenamiento PC. Almacenamiento en la nube se generaliza con Base de datos NoSQL y Almacenamiento tipo BLOB.

Existen tres casos de uso en el diagrama: Visualizar datos, Sincronizar nube y Descargar audios. Visualizar datos permitirá al investigador ver la información de la totalidad de registros que se encuentren almacenados en la nube. Sincronizar nube facilitará la actualización de cambios en la nube hacia el archivo Excel y Descargar audios permitirá al investigador bajar los archivos de audio desde la nube.

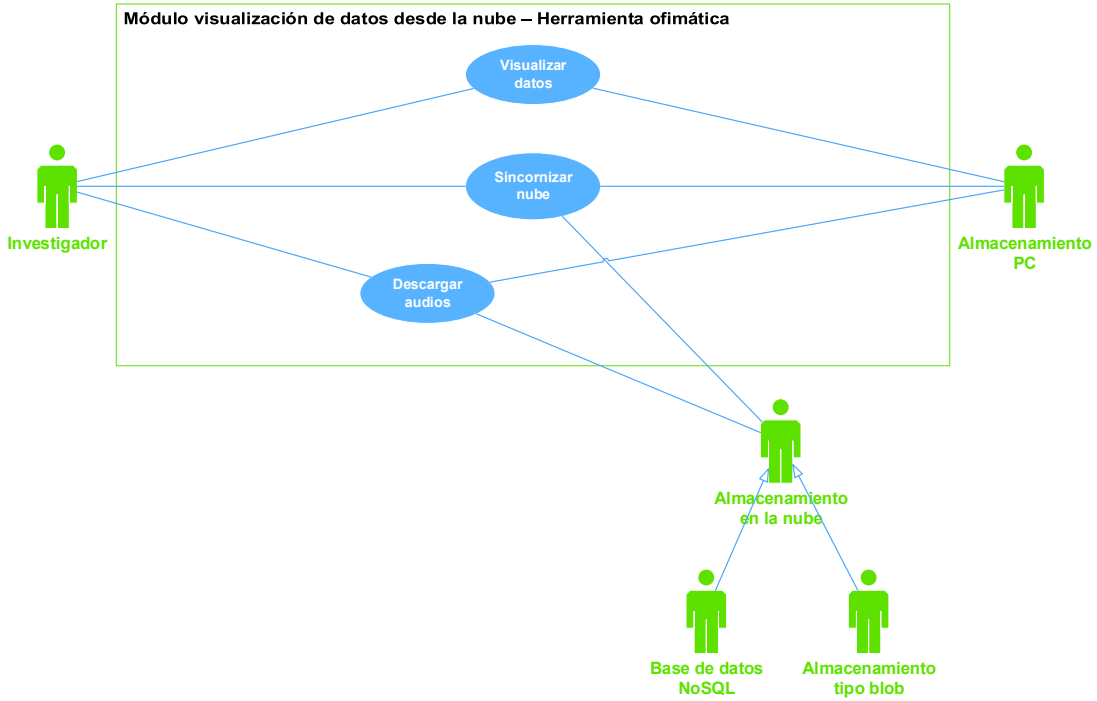


Figura 2.5 Diagrama de casos de usos del subsistema nube – Herramienta ofimática

2.2.2 Diagrama de navegación de actividades

El diagrama de la Figura 2.6 muestra la navegación que puede existir entre diferentes actividades de la aplicación Android, la elaboración del flujo de interacción cumple con los requerimientos del sistema.

Figura 2.6 Diagrama de navegación de actividades

2.2.3 Diagrama de clases

La Figura 2.7 muestra el diagrama de clases de la aplicación Android. Las clases se ven simplificadas, en la implementación del sistema se dará mayor detalle de cada una.

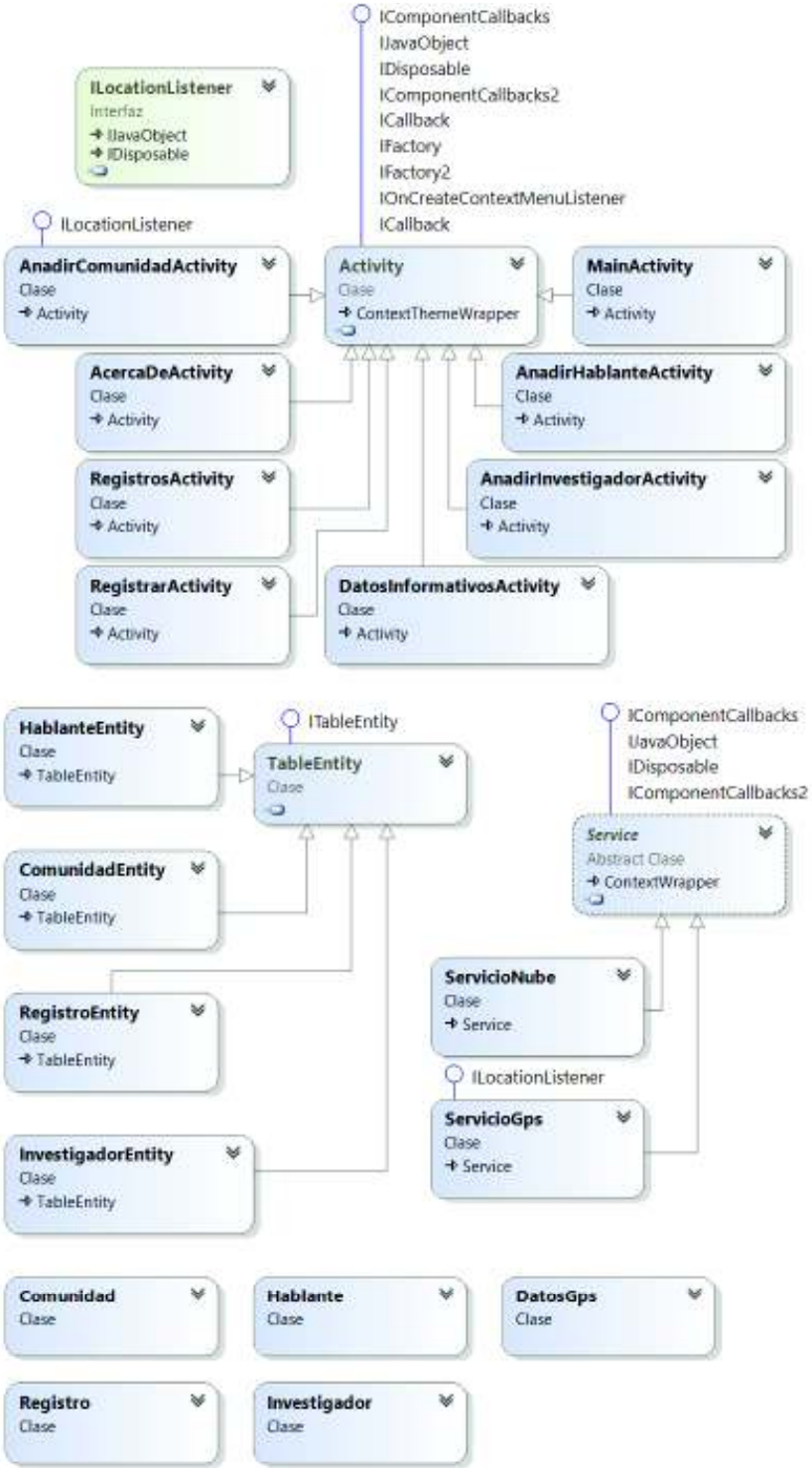


Figura 2.7 Diagrama de clases de la aplicación Android

2.2.4 Bosquejos (*sketches*) de la interfaz gráfica

Posteriormente se presentan los diseños de pantalla para cada actividad de la aplicación Android, también se indica el nombre de los componentes IU y una breve descripción del funcionamiento de estos.

2.2.4.1 Bosquejo de la actividad Principal

En la Figura 2.8 se muestra el bosquejo de la interfaz de usuario correspondiente a la actividad principal.

Figura 2.8 Bosquejo de la actividad principal

La Tabla 2.1 contiene una breve descripción de todos los componentes IU presentes en el bosquejo (*sketch*) de la Figura 2.8 correspondiente a la actividad principal.

Tabla 2.1 Descripción de los componentes IU de la actividad principal

No.	Tipo de control	Descripción
1	Botón	Accede a la actividad Datos informativos
2	Botón	Accede a la actividad Añadir Registros
3	Botón	Accede a la actividad Registros
4	Vista de texto	Muestra el texto: Sincronizar
5	Vista de texto	Muestra el texto “Registros en la nube:”
6	Vista de texto	Muestra el texto “Registros locales”
7	Botón	Accede a la actividad Acerca de
8	Interruptor	Permite iniciar o parar el envío hacia la nube
9	Vista de texto	Indica el número de registros en la nube
10	Vista de texto	Indica el número de registros locales

2.2.4.2 Bosquejo de la actividad Datos informativos

En la Figura 2.9 se observa el diseño gráfico de la interfaz de usuario correspondiente a la actividad Datos informativos.

Figura 2.9 Bosquejo de la actividad Datos informativos

La Tabla 2.2 contiene una descripción del funcionamiento de cada componente IU dentro de la actividad Datos informativos.

Tabla 2.2 Descripción de los componentes IU de la actividad Datos informativos

No.	Tipo de control	Descripción
1	Vista de texto	Muestra el texto “Campos”
2	Botón	Despliega la lista de comunidades
3	Botón	Despliega la lista de hablantes
4	Botón	Despliega la lista de investigadores
5	Vista de texto	Muestra el texto “Seleccione un campo”
6	Vista de lista	Muestra un listado del campo seleccionado
7	Vista de texto	Muestra el texto “Añadir”
8	Botón	Accede a la actividad Añadir comunidad
9	Botón	Accede a la actividad Añadir hablante
10	Botón	Accede a la actividad Añadir investigador

2.2.4.3 Bosquejo de la actividad Añadir comunidad

En la Figura 2.10 se observa la interfaz de usuario correspondiente a la actividad Añadir comunidad y en la Tabla 2.3 se encuentra la descripción de funcionamiento de los componentes IU.

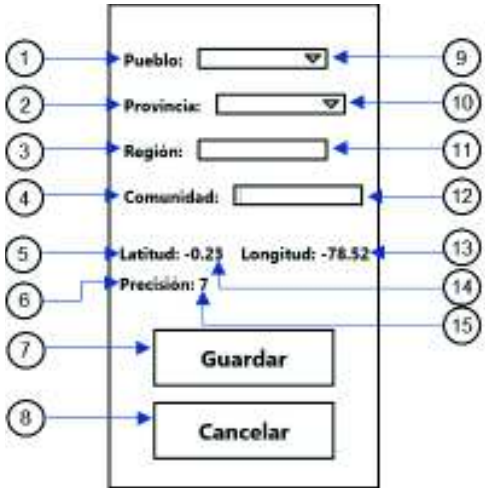


Figura 2.10 Bosquejo de la actividad Añadir comunidad

Tabla 2.3 Descripción de los componentes IU de la actividad Añadir comunidad

No.	Tipo de control	Descripción
1	Vista de texto	Muestra el texto “Pueblo:”
2	Vista de texto	Muestra el texto “Provincia”
3	Vista de texto	Muestra el texto “Región”
4	Vista de texto	Muestra el texto “Comunidad”
5	Vista de texto	Muestra el texto “Latitud”
6	Vista de texto	Muestra el texto “Precisión”
7	Botón	Permite guardar la comunidad en el almacenamiento interno
8	Botón	Cancela la acción de añadir una nueva comunidad
9	Control de número	Permite seleccionar el pueblo indígena
10	Control de número	Permite seleccionar la provincia[10]
11	Vista de texto	Muestra la región a la que pertenece la comunidad
12	Editar texto	Permite ingresar el nombre de la comunidad
13	Vista de texto	Muestra el valor de latitud de la comunidad
14	Vista de texto	Muestra el valor de longitud de la comunidad
15	Vista de texto	Muestra la precisión de los datos adquiridos por el GPS

[10] El control No. 10 es aplicable para comunidades indígenas que estén presentes en 2 o más provincias.

2.2.4.4 Bosquejo de la actividad Añadir hablante

La interfaz de usuario que se observa en la Figura 2.11 pertenece a la actividad Añadir hablante, le permitirá al usuario de la aplicación ingresar nuevos hablantes con sus respectivos metadatos al sistema.

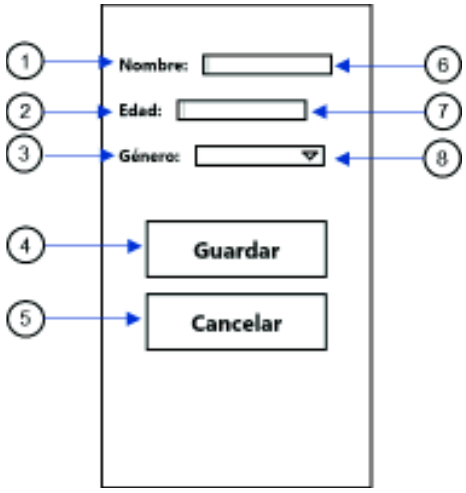


Figura 2.11 Bosquejo de la actividad Añadir hablante

Todos los componentes de interfaz gráfica de la actividad Añadir hablante y su respectiva funcionalidad se detallan en la Tabla 2.4.

Tabla 2.4 Descripción de los componentes IU de la actividad Añadir hablante

No.	Tipo de control	Descripción
1	Vista de texto	Muestra el texto “Nombre:”
2	Vista de texto	Muestra el texto “Edad:”
3	Vista de texto	Muestra el texto “Género:”
4	Botón	Permite guardar la información del hablante en el almacenamiento interno
5	Botón	Cancela la acción de añadir un nuevo hablante
6	Editar texto	Permite ingresar el nombre del hablante
7	Editar texto	Permite ingresar la edad del hablante
8	Control de número	Permite seleccionar el género del hablante

2.2.4.5 Bosquejo de la actividad Añadir investigador

En la Figura 2.12 se observa la interfaz de usuario que pertenece a la actividad Añadir investigador, le permitirá al usuario de la aplicación móvil ingresar nuevos investigadores con sus respectivos metadatos al sistema.

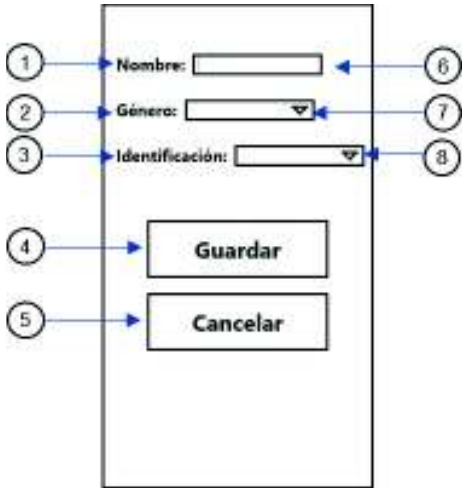


Figura 2.12 Bosquejo de la actividad Añadir investigador

Se detallan en la Tabla 2.5 todos los componentes IU que pertenecen a la interfaz de la actividad Añadir investigador, también se incluyen los detalles de funcionalidad.

Tabla 2.5 Descripción de los componentes IU de la actividad Añadir investigador

No.	Tipo de control	Descripción
1	Vista de texto	Muestra el texto “Nombre:”
2	Vista de texto	Muestra el texto “Género:”
3	Vista de texto	Muestra el texto “Identificación:”
4	Botón	Permite guardar la información del investigador en el almacenamiento interno
5	Botón	Cancela la acción de añadir un nuevo investigador
6	Editar texto	Permite ingresar el nombre del investigador
7	Control de número	Permite seleccionar el género del investigador
8	Control de número	Permite seleccionar la identificación del investigador

2.2.4.6 Bosquejo de la actividad Registrar

En la Figura 2.13 se observa el bosquejo de la actividad Registrar, todos los componentes IU incluidos en está interfaz se detallan en la Tabla 2.6, también se incluyen los detalles de funcionalidad de dichos elementos.

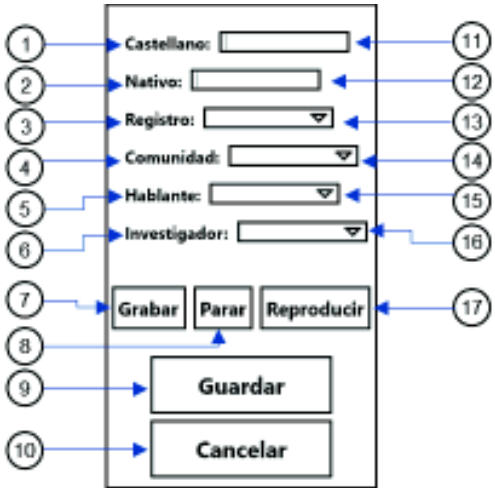


Figura 2.13 Bosquejo de la actividad Registrar

Tabla 2.6 Descripción de los componentes IU de la actividad Registrar

No.	Tipo de control	Descripción
1	Vista de texto	Muestra el texto “Castellano:”
2	Vista de texto	Muestra el texto “Nativo:”
3	Vista de texto	Muestra el texto “Registro:”
4	Vista de texto	Muestra el texto “Comunidad:”
5	Vista de texto	Muestra el texto “Hablante:”
6	Vista de texto	Muestra el texto “Investigador:”
7	Botón	Permite iniciar la grabación de audio
8	Botón	Permite finalizar la grabación de audio
9	Botón	Permite guardar el registro en el almacenamiento interno
10	Botón	Permite cancelar la acción de añadir un nuevo registro
11	Editar texto	Permite ingresar la traducción al castellano del nuevo registro
12	Editar texto	Permite ingresar la escritura en el lenguaje nativo del registro
13	Control de número	Permite seleccionar el tipo de registro
14	Control de número	Permite seleccionar la comunidad
15	Control de número	Permite seleccionar el hablante
16	Control de número	Permite seleccionar el investigador
17	Botón	Permite reproducir el registro

2.2.4.7 Bosquejo de la actividad Registros

El bosquejo de la interfaz gráfica de la actividad Registro se visualiza en la Figura 2.14.

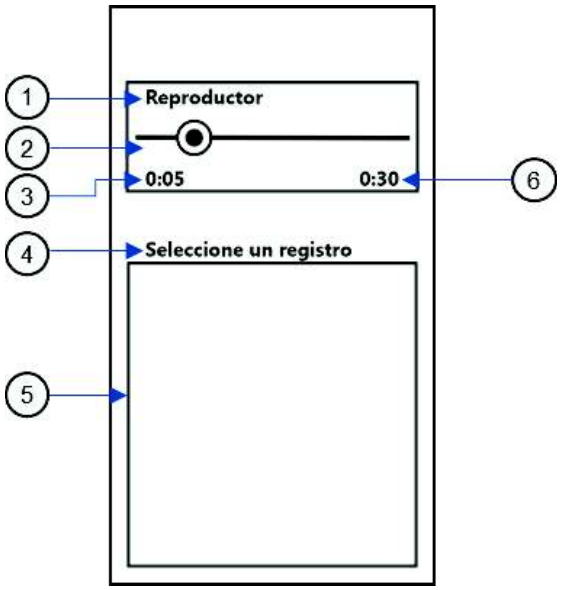


Figura 2.14 Bosquejo de la actividad Registros

En la Tabla 2.7 se detalla la funcionalidad de cada control IU correspondiente a la interfaz de usuario de la actividad Registros.

Tabla 2.7 Descripción de los componentes IU de la actividad Registros

No.	Tipo de control	Descripción
1	Vista de texto	Muestra texto “Reproductor”
2	Barra de búsqueda	Permite visualizar el avance de la reproducción del registro seleccionado
3	Vista de texto	Muestra el valor de tiempo de reproducción actual
4	Vista de texto	Muestra texto “Seleccione un registro”
5	Vista de lista	Muestra un listado de los registros
6	Vista de texto	Muestra el tiempo total que dura el registro seleccionado

2.2.4.8 Bosquejo de la actividad Acerca de

La Figura 2.15 muestra el bosquejo de la actividad Acerca de, contiene un único componte de interfaz gráfica, la funcionalidad de este se detalla en la Tabla 2.8. La finalidad de la actividad Acerca de, no se enfoca en el cumplimiento de los requerimientos del sistema, su función es proporcionar información relevante sobre el presente proyecto.

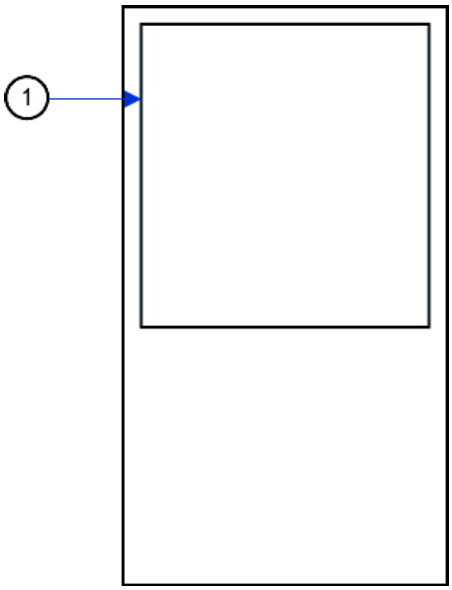


Figura 2.15 Bosquejo de la actividad Acerca de

Tabla 2.8 Descripción del componente IU de la actividad Acerca de

No.	Tipo de control	Descripción
1	Vista de texto	Muestra texto con información relevante acerca del proyecto

2.2.5 Diagramas de secuencias

Un diagrama de secuencia permite conocer la forma en la que se comunican las partes del sistema a través de mensajes, indicando el orden de sucesión de estas interacciones. Es un diagrama de tipo UML, que permite modelar la interacción entre las partes del sistema a lo largo del tiempo, el paso de este se visualiza en los ejes verticales de arriba hacia abajo. Los participantes, objetos o partes del sistema se los representa dentro de un rectángulo, para la representación de mensajes se utiliza flechas. Los objetos pueden ser instancias de clase o partes del sistema.

Si bien los diagramas de secuencia dan una idea del comportamiento del sistema, los detalles de implementación del proyecto serán revisados en la sección 2.3.

2.2.5.1 Diagrama de secuencias Ingreso de metadatos

La visualización o eliminación de metadatos es similar tanto para el caso de hablantes, comunidades e investigadores. En la Figura 2.16 se detalla la interacción del usuario a lo largo del tiempo cuando desea ver o añadir metadatos de una comunidad.

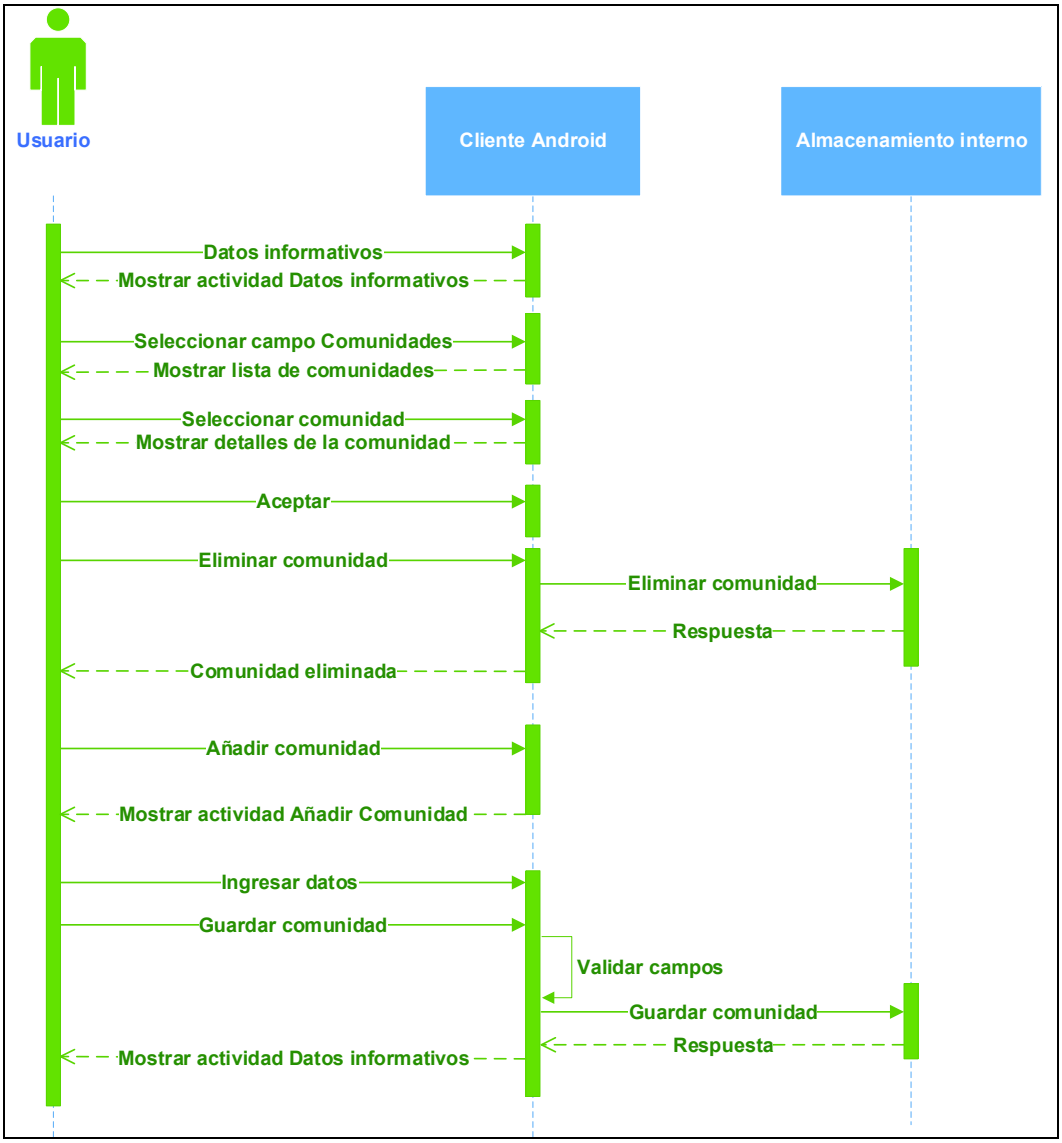


Figura 2.16 Diagrama de secuencia Ingreso de metadatos

2.2.5.2 Diagrama de secuencias Nuevo registro

La interacción de los diferentes componentes del sistema a lo largo del tiempo, cuando el usuario realiza un nuevo registro se detalla en el diagrama de secuencia de la Figura 2.17.

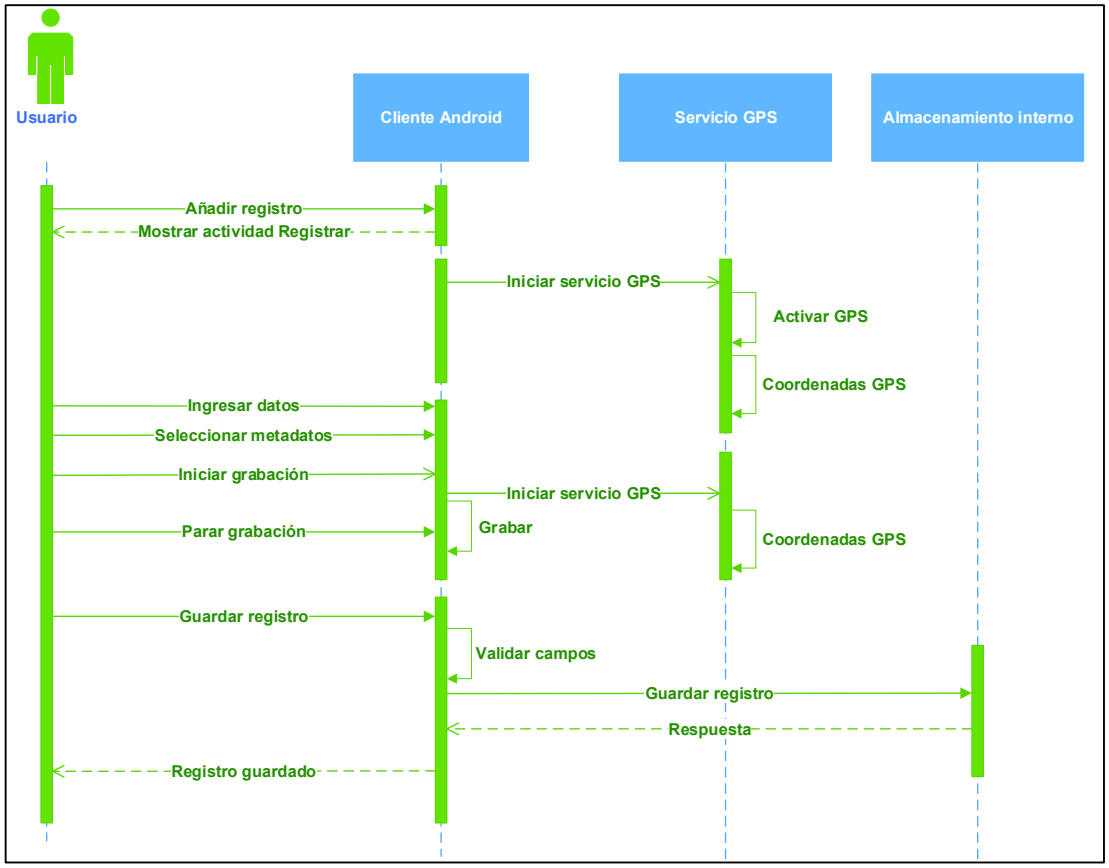


Figura 2.17 Diagrama de secuencias Nuevo registro

2.2.5.3 Diagrama de secuencias Reproducir registros

En la Figura 2.18, se presenta la interacción que existe entre las diferentes partes del sistema al realizar la reproducción de un registro de audio.

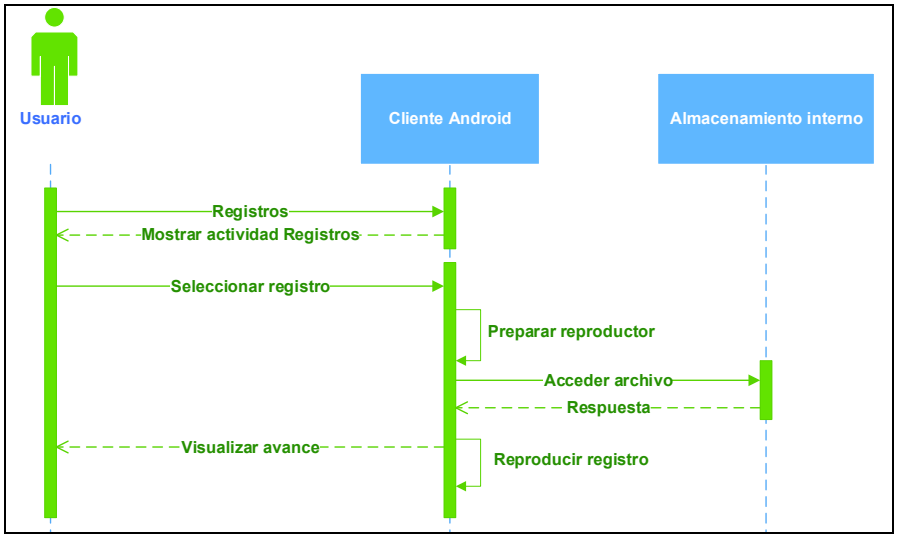


Figura 2.18 Diagrama de secuencias Reproducir registros

2.2.5.4 Diagrama de secuencias Envío de datos a la nube

Para que los datos y metadatos de cada registro puedan ser subidos a la nube, es necesario la ejecución de varios procesos, tanto síncronos como asíncronos.

En el diagrama de secuencia de la Figura 2.19 se tiene un esquema, en el que se describe el proceso de almacenamiento en la nube, tanto para registros como metadatos.

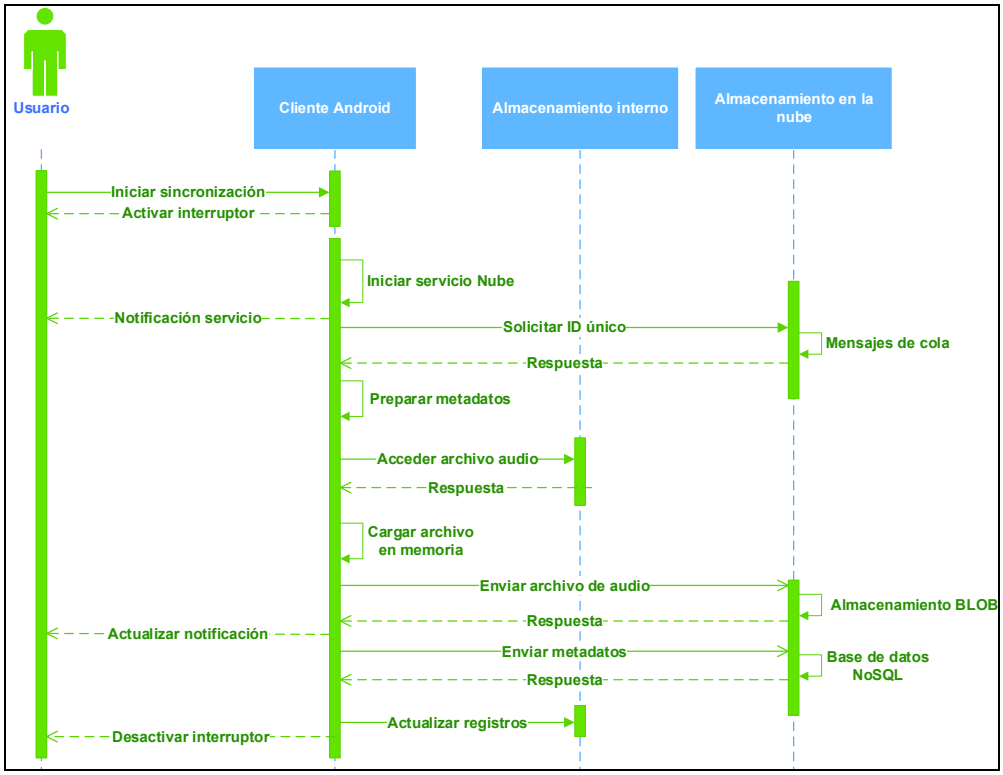


Figura 2.19 Diagrama de secuencia Envío de datos a la nube.

2.3 Implementación del prototipo

En esta sección se detallará el desarrollo del prototipo en base a los requerimientos y diseño del sistema. Para la implementación de la aplicación Android se usará la versión 15.8.6 de Microsoft Visual Studio, que incluye las herramientas y bibliotecas de Xamarin.Android, el lenguaje de programación que se usará es C#.

El proveedor de servicios en la nube será Microsoft Azure, se utilizará una cuenta de almacenamiento en la nube. Las cuentas de almacenamiento que provee Azure brindan cuatro tipos de servicios, de los cuales se usarán:

- Blob Storage.
- Table Storage.

- Queue Storage.

Y finalmente para la visualización de datos se usará el programa de ofimática Microsoft Excel.

2.3.1 Implementación aplicación Android

La implementación de la aplicación Android lleva como nombre Nativa, todas sus actividades se implementan según el diseño de la aplicación, los bocetos de la aplicación sirvieron de base para crear los diseños definitivos.

2.3.1.1 Implementación de la actividad Principal

La actividad Principal (*MainActivity*) es la primera pantalla que verá el usuario cuando inicie la aplicación por primera vez. En la Figura 2.20 se observa el diseño gráfico de esta actividad.

Figura 2.20 Interfaz de usuario de la actividad Principal.

En la Figura 2.21 se ven detalles de la clase correspondiente a la actividad Principal, que contiene: diez campos, una propiedad y 19 métodos.

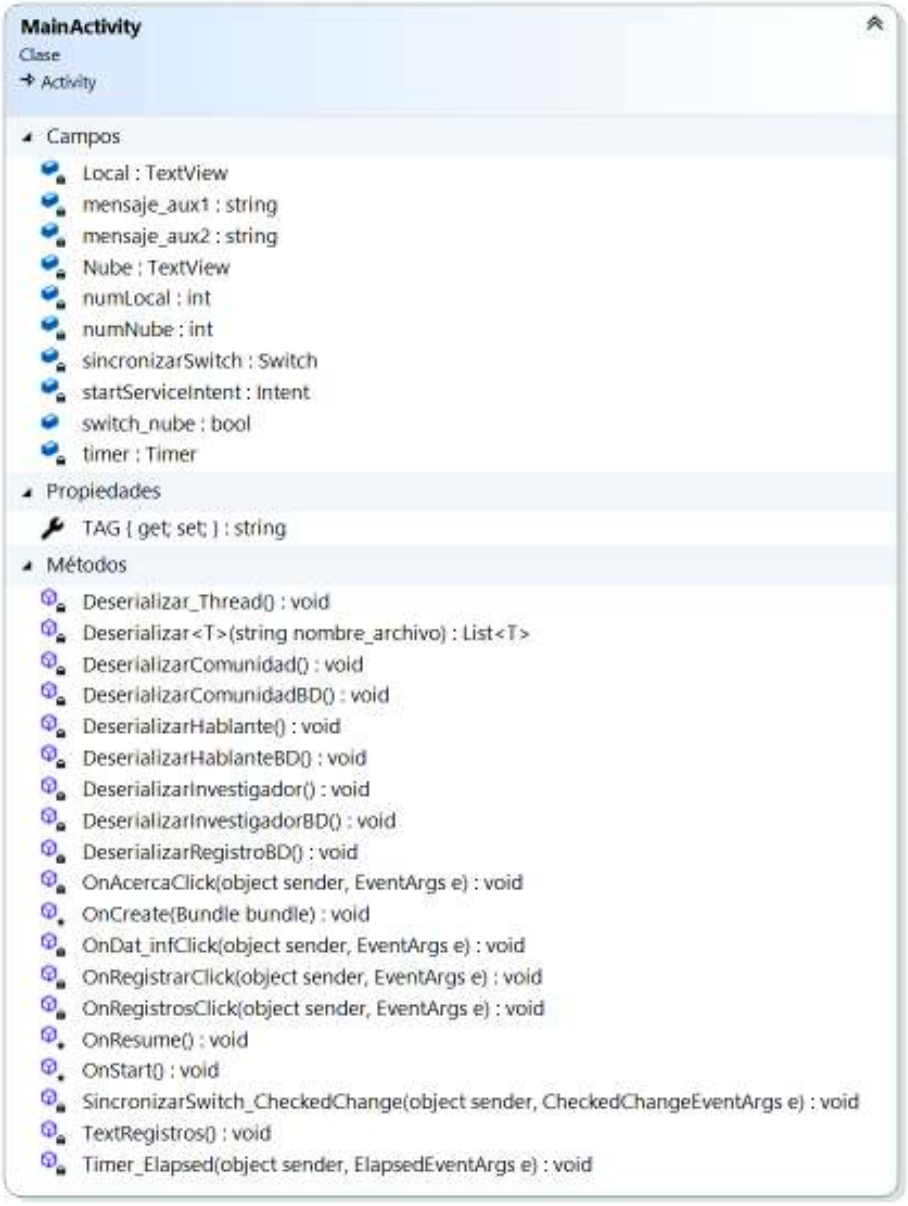


Figura 2.21 Clase correspondiente a la actividad Principal.

El código correspondiente a los métodos `OnDat_infClick`, `OnRegistrarClick`, `OnRegistrosClick` y `OnAcercaClick` se ejecuta cuando el usuario accione los botones de esta interfaz. El código de estos métodos es similar, realiza la transición a una nueva actividad. El Segmento de Código 2.1 muestra la programación correspondiente al método `OnDat_infClick`, permite acceder a la actividad `DatosInformativosActivity` a través del

método `StartActivity`, este necesita como parámetro un `Intent` de tipo igual al de la nueva actividad.

```
        private void OnDat_infClick(object sender,EventArgs e)
        {
            Intent intent = new Intent(this, typeof(DatosInformativosActivity));
            StartActivity(intent);
            OverridePendingTransition(Android.Resource.Animation.SlideInLeft,
                                      Android.Resource.Animation.SlideOutRight);
        }
```

Segmento de Código 2.1 Método `OnDat_infClick` de la clase `MainActivity`.

El método `OverridePendingTransition` es utilizado para definir la animación de transición entre actividades, recibe dos parámetros; el primero define la animación de la actividad saliente y el segundo de la entrante.

Todos los métodos `Deserializar` tipo `void`, invocan al método genérico `Deserializar<T>`, la codificación del método `DeserializarComunidad` se incluye en el Segmento de Código 2.2. El objetivo de estos métodos es traer información desde archivos guardados en el almacenamiento interno del dispositivo al contexto actual de la aplicación para poder interactuar con dichos datos.

```
    private static void DeserializarComunidad()
    {
        List<Comunidad> comunidades_des = Deserializar<Comunidad>("comunidades.xml");
        if (comunidades_des!=null)
        {
            DatosInformativosActivity.Comunidades = comunidades_des;
        }
    }
```

Segmento de Código 2.2 Método `DeserializarComunidad`

El método genérico `Deserializar<T>` recibe como parámetros una colección genérica de tipo `List<T>` y una cadena de texto que corresponde al nombre del archivo que se necesita deserializar[11]. Para este ejemplo, la colección genérica es del tipo `<Comunidad>` y el nombre del archivo a deserializar es "comunidades.xml".

El método genérico `Deserializar<T>` luego de su ejecución regresa una colección genérica de tipo `List<T>`, en el ejemplo, el nombre de dicha colección es `comunidades_des`, ésta contiene los datos del archivo "comunidades.xml" deserializado. Esta información es cargada a la lista comunidades de la actividad `DatosInformativosActivity`.

El código de programación del método genérico `Deserializar<T>` se muestra en el Segmento de Código 2.3.

[11] Proceso inverso a la serialización de objetos

```
private static List<T> Deserializar<T>(string nombre_archivo)
        {
            string path = Path.Combine(System.Environment.GetFolderPath(
                          System.Environment.SpecialFolder.Personal), nombre_archivo);
            if (File.Exists(path))
            {
                XmlSerializer serializador = new XmlSerializer(typeof(List<T>));
                using (FileStream fs2 = File.OpenRead(path))
                {
                    List<T> Resultado = (List<T>)serializador.Deserialize(fs2);
                    return Resultado;
                }
            }
            else
            {
                return null;
            }
        }
```

Segmento de Código 2.3 Método `Deserializar<T>`

El método `Deserializar<T>` crea la ruta a la que se quiere acceder a través de `Path.Combine`, si el archivo existe en el almacenamiento interno, se procede con la deserialización, el método devuelve el contenido del archivo.

En la Figura 2.20 se ve un interruptor, este puede iniciar o parar la sincronización de registros a la nube, cuando este control cambia de posición, se ejecuta el código del método `SincronizarSwitch_CheckedChange`. Cuando se inicia la sincronización, el código de este método invoca un servicio en segundo plano llamado `ServicioNube`. Al finalizar o parar la sincronización, el código detiene al servicio.

```
startServiceIntent = new Intent(this, typeof(Servicios.ServicioNube));
                    if (Android.OS.Build.VERSION.SdkInt >= Android.OS.BuildVersionCodes.O)
                    {
                        StartForegroundService(startServiceIntent);
                        timer.Start();
                    }
                    else
                    {
                        StartService(startServiceIntent);
                        timer.Start();
                    }
```

Segmento de Código 2.4 Extracto del método `SincronizarSwitch_CheckedChange`.

El Segmento de Código 2.4 contiene la programación que invoca el inicio del servicio `ServicioNube`. Cuando el software del dispositivo tiene un nivel de API mayor o igual a 26 (Oreo 8.0), el servicio inicia con el método `StartForegroundService`, para niveles de API inferiores, es invocado a través de `StartService`. El código del método `StopService` finalizará el servicio para cualquier nivel de API. Todos estos métodos usan como parámetro un `Intent` de tipo igual al servicio que desean acceder o finalizar.

Cuando se realiza la invocación del servicio de sincronización, empieza el inicio del temporizador `timer`, la función de este contador es ejecutar el método `Timer_Elapsed` cada vez que se desborde, para empezar nuevamente en cero. La programación del método `Timer_Elapsed` se observa en el Segmento de Código 2.5.

```
        private void Timer_Elapsed(object sender, ElapsedEventArgs e)
        {
            RunOnUiThread(() =>
            {
                if (switch_nube == false)
                {
                    sincronizarSwitch.Checked = false;
                    timer.Stop();
                }
                TextRegistros();
            });
        }
```

Segmento de Código 2.5 Método `Timer_Elapsed`.

El temporizador se ejecuta en un hilo de programación diferente al del programa principal, el código encapsulado en `RunOnUiThread` se ejecutará en el hilo principal, permitiendo cambiar desde código la posición del interruptor `sincronizarSwitch`, siempre y cuando la sincronización haya finalizado, se detendrá el temporizador. La configuración del temporizador se encuentra en el método `OnCreate`; en el Segmento de Código 2.6 se observa lo mencionado.

```
            timer = new System.Timers.Timer
            {
                Interval = 1000
            };
            timer.Elapsed += Timer_Elapsed;
```

Segmento de Código 2.6 Extracto del método `OnCreate`.

En la configuración se asigna el nombre `timer` al temporizador, el intervalo de tiempo es de 1000 ms y el evento de desborde está asociado al método `Timer_Elapsed`.

La programación del método `TextRegistros` permite actualizar la visualización del número de registros locales y en la nube. Es invocado cuando se ejecutan los métodos: `OnStart`, `Timer_Elapsed` y `DeserializarRegistroBD`. Así se garantiza que el usuario visualice información actualizada.

En esta actividad se implementaron los métodos de devolución de llamada (*callbacks*) `OnStart` y `OnResume`. Puede darse el caso que el servicio de sincronización termine cuando la aplicación esté pausada, de ocurrir este escenario, la codificación dentro del método `OnResume` permite actualizar a la posición correcta del interruptor de sincronización.

La programación del método `OnStart` garantiza que los datos que verá el usuario al iniciar la aplicación sean los correctos, también iniciará un nuevo hilo (*thread*) de programación, que permitirá ejecutar la deserialización de los archivos tipo XML, sin afectar a una fluidez en la interfaz gráfica. Lo descrito se detalla en el Segmento de Código 2.7.

```
ThreadStart threadDelegate = new ThreadStart(Deserializar_Thread);
Thread thread = new Thread(threadDelegate);
thread.Start();
```

Segmento de Código 2.7 Extracto del método `OnStart`

La creación del delegado `threadDelegate` usa como parámetro el nombre del método a ejecutarse en el nuevo hilo de programación, así mismo, la creación del nuevo hilo lleva como parámetro el delegado creado con anterioridad. Cuando se invoque al método de tipo `Thread.Start`, iniciará la ejecución del código `Deserializar_Thread`, dentro del nuevo hilo de programación.

2.3.1.2 Implementación de la actividad Datos informativos

En la Figura 2.22 se observa el diseño gráfico de la actividad Datos informativos.

Figura 2.22 Interfaz de usuario de la actividad Datos informativos.

En la Figura 2.23 se detalla la clase correspondiente a la actividad Datos informativos, contiene diez campos y 19 métodos.

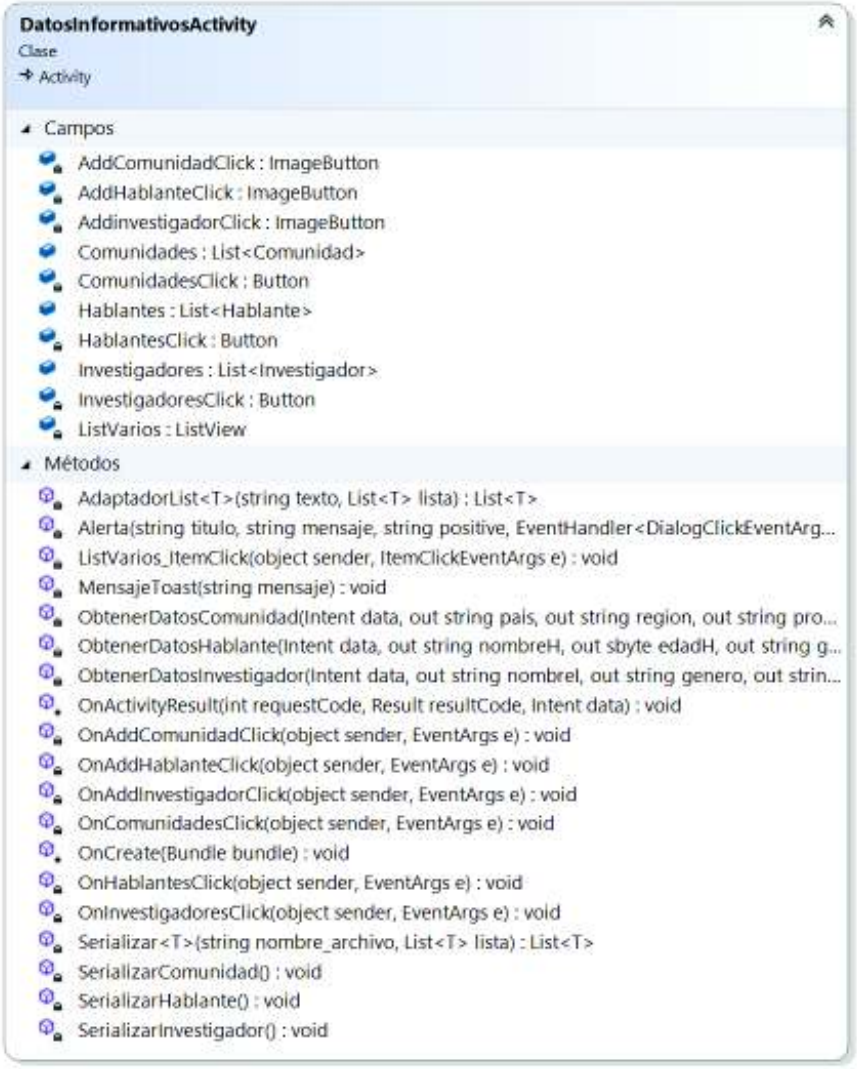


Figura 2.23 Clase correspondiente a la actividad Datos Informativos

Los métodos `OnAddComunidadClick`, `OnAddHablanteClick` y `OnAddInvestigadoreClick` se ejecutan cuando el usuario active cualquier botón del tipo añadir, su codificación es similar. En el Segmento de Código 2.8 se detalla el código de `OnAddComunidadClick`. La programación de estos métodos desplegará las actividades `AnadirComunidadActivity`, `AnadirHablanteActivity` y `AnadirInvestigadorActivity` respectivamente. Estas actividades facilitarán el ingreso de metadatos de comunidad, hablante o investigador según el caso. Estos metadatos deberán ser enviados a la actividad `DatosInformativosActivity`, para este fin se puede usar variables globales o almacenamiento interno. Como los datos a compartir son pocos, se recomienda invocar la nueva actividad con el método

`StartActivityForResult`, a través de dos parámetros, el primero un `Intent` de tipo igual a la actividad invocada, el segundo un código de solicitud de tipo entero, este número debe ser único en el contexto de la actividad.

```
        private void OnAddComunidadClick(object sender, EventArgs e)
        {
            var intent = new Intent(this, typeof(AnadirComunidadActivity));
            StartActivityForResult(intent,100);

        }
```

Segmento de Código 2.8 Método `OnAddComunidadClick`

También se considera buena práctica usar este método para compartir información, siempre y cuando los datos solo se utilicen entre las actividades que interactúan. En caso de que varias actividades necesiten usar estos datos, se recomienda usar variables globales.

En la actividad Datos informativos se crean tres colecciones genéricas `List<T>`, `T` es el tipo de dato que se almacenará en dicha colección, puede ser inclusive una clase previamente definida. Los nombres de las colecciones genéricas definidas en esta actividad son `Comunidades`, `Hablantes` e `Investigadores`. En el Segmento de Código 2.9 se observa la creación de estas listas genéricas, son del tipo de clase `Comunidad`, `Hablante` o `Investigador`, según sea el caso.

```
        public static List<Comunidad> Comunidades = new List<Comunidad>();
        public static List<Hablante> Hablantes = new List<Hablante>();
        public static List<Investigador> Investigadores = new List<Investigador>();
```

Segmento de Código 2.9 Definición campos públicos de la actividad Datos informativos

En las Figuras 2.24, 2.25 y 2.26 se ve la estructura de las clases `Comunidad`, `Hablante` e `Investigador`.

Figura 2.24 Clase `Comunidad`

Figura 2.25 Clase `Hablante`

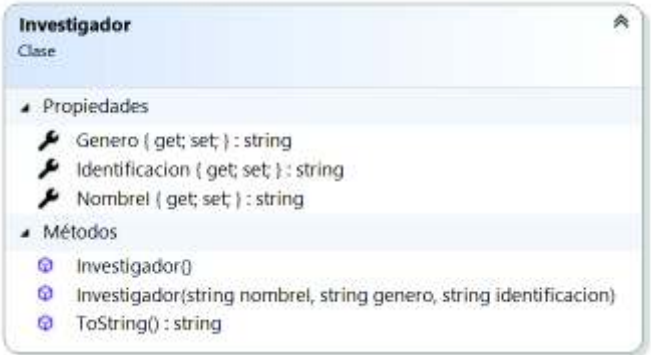


Figura 2.26 Clase `Investigador`

Las tres colecciones genéricas `List<T>` contendrán un listado de las comunidades, hablantes e investigadores añadidos, incluido los datos de cada una. Cuando `DatosInformativosActivity` detecta que fue invocada desde una actividad previamente lanzada con código de solicitud, ejecuta la programación del método `OnActivityResult`, se verifica el número de solicitud para enviar los datos recibidos a la colección genérica correspondiente.

En el presente proyecto se utilizan tres códigos de solicitud para gestionar el intercambio de datos entre actividades. En la Figura 2.27 se visualizan los códigos utilizados.

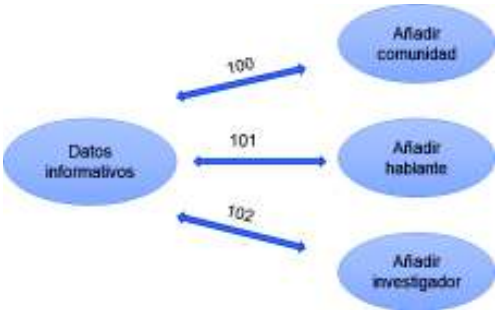


Figura 2.27 Códigos de solicitud entre actividades

En la Figura 2.28 se muestra una parte de la programación del método `OnActivityResult`, recibe tres parámetros que son: código de solicitud, estado y datos a compartir. El estado de la solicitud indica si se completó exitosamente la transferencia de datos.

```
else if (requestCode == 101 && resultCode == Result.Ok)
{
    ObtenerDatosHablante(data, out string nombreH, out sbyte edadH, out string generoH);
    Hablantes.Add(new Hablante(nombreH, edadH, generoH));
    AdaptadorList("Hablantes: ", Hablantes);
    SerializarHablante();
}
else if (requestCode == 102 && resultCode == Result.Ok)
{
    ObtenerDatosInvestigador(data, out string nombreI, out string genero, out string identificacion);
    Investigadores.Add(new Investigador(nombreI, genero, identificacion));
    AdaptadorList("Investigadores: ", Investigadores);
    SerializarInvestigador();
}
```

Figura 2.28 Extracto del método `OnActivityResult`

A continuación, se explica la codificación del método, suponiendo que se recibieron datos desde la actividad `AnadirHablanteActivity`, primero se verifica que el código de solicitud sea 101 y que la transferencia de datos fue exitosa. Se invoca al código `ObtenerDatosHablante`, permite guardar los datos recibidos en variables locales, con esta información se añade un nuevo hablante a la colección genérica `Hablantes`. Finalmente se invoca a los métodos `AdaptadorList` y `SerializarHablante`.

El código del método `ObtenerDatosHablante` se incluye en el Segmento de Código 2.10

```
private static void ObtenerDatosHablante(Intent data, out string nombreH, out sbyte edadH, out string generoH)
{
    nombreH = data.GetStringExtra("NombreHablante");
    edadH = data.GetByteExtra("EdadHablante", -1);
    generoH = data.GetStringExtra("GeneroHablante");
}
```

Segmento de Código 2.10 Método `ObtenerDatosHablante`

El código del método genérico `AdaptadorList` permite cambiar el contenido del controlador tipo `ListView` de la interfaz gráfica de la actividad Datos informativos. Se utiliza el mismo `ListView` para poder visualizar datos de las listas de comunidades, hablantes e investigadores añadidos. La programación de este método se observa en el Segmento de Código 2.11.

```
private List<T> AdaptadorList<T>(string texto, List<T> lista)
{
    FindViewById<TextView>(Resource.Id.textView1).Text = texto;
    ListVarios.Adapter = new ArrayAdapter<T>(this,
          Android.Resource.Layout.SimpleListItem1,Android.Resource.Id.Text1, lista);
    return null;
}
```

Segmento de Código 2.11 Método genérico `AdaptadorList`

El código de SerializarHablante invoca al método genérico Serializar<T>, la programación de SerializarHablante se observa en el Segmento de Código 2.12.

```
private static void SerializarHablante()
{
    Serializar("hablantes.xml", Hablantes);
}
```

Segmento de Código 2.12 Método SerializarHablante

El método Serializar<T> usa dos parámetros, el primero es de tipo string, proporciona el nombre del archivo que guardará la información serializada y el segundo de tipo List<T>, contiene la colección genérica a serializar[12]. En el Segmento de Código 2.13 se incluye la programación del método Serializar<T>.

```
private static List<T> Serializar<T>(string nombre_archivo,List<T> lista)
{
    string path = Path.Combine(System.Environment.GetFolderPath
                    (System.Environment.SpecialFolder.Personal), nombre_archivo);
    using (Stream fs = new FileStream(path, FileMode.Create, FileAccess.Write, FileShare.None))
    {
        XmlSerializer serializador = new XmlSerializer(typeof(List<T>));
        serializador.Serialize(fs, lista);
    }
    return null;
}
```

Segmento de Código 2.13 Método Serializar<T>

Las colecciones genéricas serializadas por el método Serializar<T> se guardan en el almacenamiento interno, con el fin de poder acceder a estos datos después, inclusive cuando se haya cerrado la aplicación por completo.

En la interfaz de la actividad Datos informativos de la Figura 2.22 se observan tres botones ubicados a la izquierda con los nombres: Comunidades, Hablantes e Investigadores. Cuando el usuario accione dichos botones, el sistema ejecutará la codificación de los métodos OnComunidadesClick, OnHablantesClick y OnInvestigadoresClick; según sea el caso. Estos métodos invocan al método AdaptadorList, esto permite visualizar en la misma pantalla información de comunidades, hablantes e investigadores; dependiendo de la selección realizada.

```
private void OnComunidadesClick(object sender, EventArgs e)
{
    AdaptadorList("Comunidades: ", Comunidades);
}
```

Segmento de Código 2.14 Método OnComunidadesClick.

[12] Conversión de un objeto a una cadena de bytes, con fines de almacenamiento o transmisión, guardando el estado actual del mismo.

En la Figura 2.29 se muestra la lista de comunidades, previamente seleccionado el botón Comunidades.

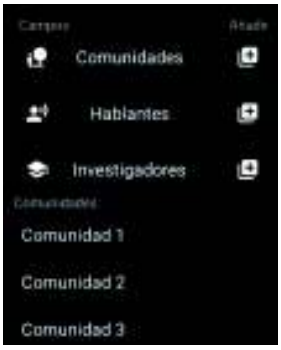


Figura 2.29 Selección del botón Comunidades

Cuando el usuario seleccione un elemento de la lista, la programación del método `ListVarios_ItemClick` se ejecutará, verifica qué ítem fue seleccionado y el tipo de lista a la que pertenece, con estos parámetros invoca al método `Alerta` para desplegar los detalles del elemento. En la Figura 2.30 se observa el diálogo de alerta que ejecutará el método `Alerta` cuando se seleccione alguna comunidad de la lista.

Figura 2.30 Detalles de la comunidad seleccionada

El método `Alerta` tiene tres parámetros tipo `string` y uno de tipo `EventHandler`. Los parámetros `string` especifican: título, contenido y etiqueta del botón derecho; `EventHandler` es un controlador de eventos de tipo delegado. En el Segmento de Código 2.15 se observa el código del método `Alerta`.

```
private void Alerta(string titulo, string mensaje, string positive,
                    EventHandler<DialogClickEventArgs> eventHandler)
{
    var dialog = new AlertDialog.Builder(this);
    dialog.SetTitle(titulo);
    dialog.SetMessage(mensaje);
    dialog.SetNeutralButton("Ok", delegate { });
    dialog.SetPositiveButton(positive, eventHandler);
    dialog.Show();
}
```

Segmento de Código 2.15 Método `Alerta`.

Cuando el usuario elija la opción eliminar comunidad, se ejecutará el delegado[13], incluido como cuarto parámetro del método `Alerta`, que apunta a `BorrarComunidad`. La programación de `BorrarComunidad` se detalla en el Segmento de Código 2.16.

```
void BorrarComunidad(object sender1, DialogClickEventArgs e1)
{
    Comunidad comunidad1 = Comunidades[position];
    string toast = string.Format("Comunidad " + comunidad1.NombreC+" eliminada");
    MensajeToast(toast);
    Comunidades.Remove(Comunidades[position]);
    AdaptadorList("Comunidades: ", Comunidades);
    SerializarComunidad();
}
```

Segmento de Código 2.16 Método `BorrarComunidad`.

El código del método `BorrarComunidad` elimina la comunidad seleccionada, despliega un mensaje de notificación que indica el nombre del elemento eliminado, invoca a `AdaptadorList` para actualizar los datos de pantalla, evitando la visualización de espacios vacíos en el `ListView`. Luego el método `SerializarComunidad` actualiza el archivo XML que contiene la lista de comunidades en el almacenamiento interno del dispositivo.

2.3.1.3 Implementación de la actividad Añadir comunidad

Esta actividad permite ingresar datos correspondientes a una nueva comunidad, en la Figura 2.31 se observa el diseño gráfico de su pantalla.

Figura 2.31 Interfaz de usuario de la actividad Añadir comunidad.

[13] Un delegado permite almacenar direcciones de métodos.

La clase `AnadirComunidadActivity` hereda de `Activity`, implementa a la interfaz `ILocationListener`, que se usa para recibir notificaciones de `LocationManager` cuando cambia la ubicación del dispositivo, estos avisos se gestionan a través de los métodos propios de `ILocationListener`.

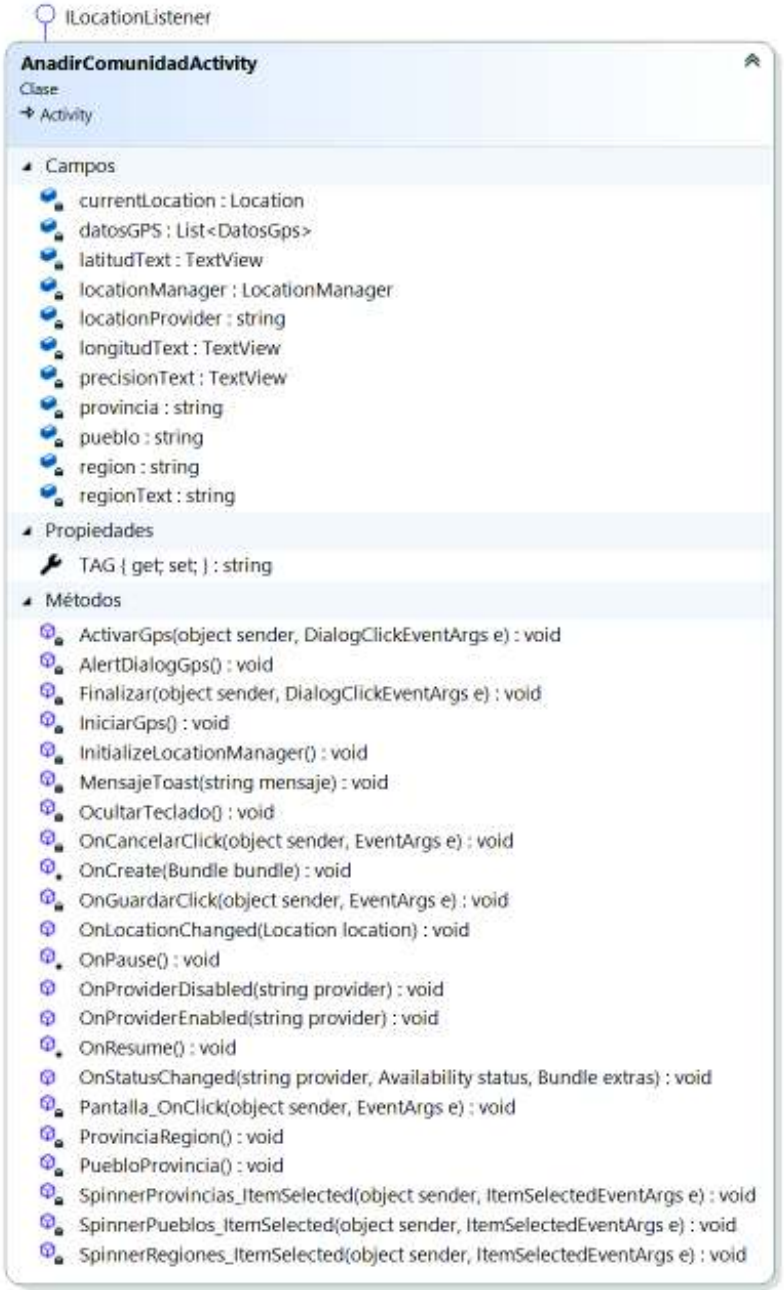


Figura 2.32 Clase correspondiente a la actividad Añadir comunidad.

Los métodos correspondientes a la implementación de la interfaz `ILocationListener` son `OnLocationChanged, OnProviderDisabled, OnProviderEnabled` y `OnStatusChanged.`

El método `OnLocationChanged` es invocado cuando la ubicación del dispositivo ha cambiado, recibe la posición actual como parámetro.

La codificación de los métodos `OnProviderDisabled` y `OnProviderEnabled` se ejecuta cuando el usuario habilite o deshabilite al proveedor de ubicación respectivamente. Estos métodos reciben como parámetro el nombre del proveedor asociado al cambio de estado.

El método `OnStatusChanged` es llamado cuando existen cambios en el estado de la fuente de ubicación, recibe como parámetro el nombre del proveedor asociado. Se tienen como posibles estados: fuera de servicio, disponible y temporalmente inaccesible.

Esos son los escenarios ante los cuales se ejecutan los códigos de programación de los métodos pertenecientes a la interfaz `ILocationListener`. A continuación, se describen las partes más importantes de la codificación incluida en estos métodos.

El código del método `OnLocationChanged` guarda las últimas ubicaciones en una colección genérica de tipo clase `DatosGps`, la estructura de esta se observa en la Figura 2.33.

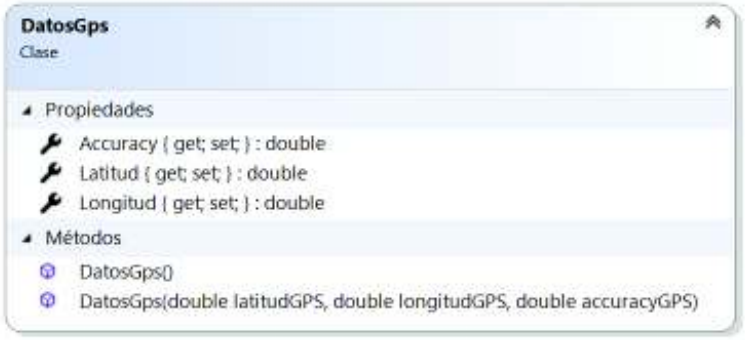


Figura 2.33 Clase `DatosGps`

Un extracto del código de programación del método `OnLocationChanged` se observa en el Segmento de Código 2.17.

```
double minimo = datosGPS.Min(x => x.Accuracy);
DatosGps min = datosGPS.Find(x => x.Accuracy == minimo);
string latitud = min.Latitud.ToString();
string longitud = min.Longitud.ToString();
string precision ="Precisión: "+ min.Accuracy.ToString();
if (datosGPS.Count >= 4)
{
    datosGPS.Remove(datosGPS[0]);
}
latitudText.Text = latitud;
longitudText.Text = longitud;
precisionText.Text = precision;
```

Segmento de Código 2.17 Extracto del método `OnLocationChanged`

La variable `minimo` guarda la mejor precisión de las ubicaciones contenidas en la colección genérica `List<DatosGps>`, con ese valor se busca la posición correspondiente, se extraen datos de longitud, latitud y precisión, esta información será desplegada en pantalla. La colección genérica contendrá siempre las últimas cuatro mediciones de ubicación.

La programación de `OnProviderDisabled` envía un mensaje de notificación solicitando al usuario que active el GPS, luego invoca al método `AlertDialogGps`, este despliega en pantalla el diálogo de alerta de la Figura 2.34.

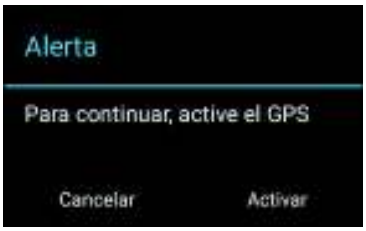


Figura 2.34 Diálogo de alerta solicitando la activación del GPS.

El diálogo de alerta presenta al usuario dos opciones: Cancelar que cierra la actividad `AnadirComunidadActivity` y Activar que invoca al método `ActivarGps`. La codificación de `ActivarGps` abre una nueva pantalla con las configuraciones de ubicación del dispositivo, el código de este método se encuentra en el Segmento de Código 2.18.

```
        private void ActivarGps(object sender, DialogClickEventArgs e)
        {
            Intent intent = new Intent(Android.Provider.Settings.ActionLocationSourceSettings);
            StartActivity(intent);
        }
```

Segmento de Código 2.18 Método `ActivarGps`

El usuario observará una pantalla similar a la Figura 2.35.

Figura 2.35 Interfaz de las configuraciones de ubicación

El código de `OnProviderEnabled` invoca al método `InitializeLocationManager`, éste realiza la elección del proveedor de ubicación, según el criterio de selección especificados en el mismo método. Parte del código de programación del método `InitializeLocationManager` se observa en el Segmento de Código 2.19.

```
Criteria criteriaLocation = new Criteria
{
    Accuracy = Accuracy.Fine
};
IList<string> listProviders = locationManager.GetProviders(criteriaLocation, true);
if (listProviders.Any())
{
    locationProvider = listProviders.First();
}
```

Segmento de Código 2.19 Extracto del método `InitializeLocationManager`.

El código del método `InitializeLocationManager` establece precisión fina como criterio de ubicación, también a través de `locationManager.GetProviders` se crea una colección genérica de proveedores de posición que cumplan con el criterio establecido y finalmente se elige al mejor de la lista que cumpla estos parámetros.

La programación del código `OnStatusChanged` despliega un mensaje de notificación en pantalla con el contenido “Recuperando datos del GPS”.

A continuación, se describen los métodos de la clase `AnadirComunidadActivity` que no pertenecen a la interfaz `ILocationListener`. Primero se describen los métodos de devolución de llamada `OnResume` y `OnPause` que gestionarán el ciclo de vida de la actividad.

El método `OnResume` se ejecutará cuando la pantalla de la actividad se despliegue por primera vez o si se regresa de segundo plano. Cuando el usuario vea las configuraciones de ubicación de la Figura 2.35, se ejecutará el código del método `OnPause` y al regresar a la aplicación se ejecutará la programación de `OnResume`.

Parte de la codificación del método `OnResume` se muestra en el Segmento de Código 2.20.

```
locationManager = GetSystemService(Context.LocationService) as LocationManager;
if (locationManager.IsProviderEnabled(LocationManager.GpsProvider))
{
    InitializeLocationManager();
    locationManager.RequestLocationUpdates(locationProvider, 0, 1, this);
}
else
{
    IniciarGps();
}
```

Segmento de Código 2.20 Extracto del método `OnResume`.

El código del método `OnResume` permite acceder a los servicios de ubicación del sistema Android a través de `GetSystemService`. Si el GPS se encuentra activado invoca a `InitializeLocationManager`, después se configuran las actualizaciones de ubicación para que se produzcan ante cambios de posición de al menos un metro. Cuando el GPS no está habilitado se invoca a `IniciarGps` que llama al método `AlertDialogGps`.

Como se observa en la Figura 2.31, el usuario debe seleccionar el pueblo al que pertenece la comunidad. En la Tabla 1.1 se observa un listado de pueblos pertenecientes a la nacionalidad Kichwa con sus respectivas provincias. Cuando el usuario seleccione el pueblo, las provincias desplegadas estarán de acuerdo con la tabla mencionada. Si algún se ubica en una sola provincia, automáticamente el sistema desplegará la misma. Todas estas acciones son desarrolladas por el método `PuebloProvincia`. La región del país a la que pertenece la comunidad será vista en pantalla automáticamente, previa selección de provincia, a través de la codificación del método `ProvinciaRegion`.

Parte de la programación del método `PuebloProvincia` se observa en el Segmento de Código 2.21.

```
string[] provinciasString = Resources.GetStringArray(Resource.Array.provincias_ecuador);
List<string> provinciasArray = new List<string>();
switch (pueblo)
{
    case "Chibuleo":
        provinciasArray.Add(provinciasString[23]);
        break;
    case "Kañari":
        provinciasArray.Add(provinciasString[0]);
        provinciasArray.Add(provinciasString[1]);
        provinciasArray.Add(provinciasString[3]);
        break;
```

Segmento de Código 2.21 Extracto del método `PuebloProvincia`.

Fragmento de la programación del método `ProvinciaRegion` se detalla en el Segmento de Código 2.22.

```
string[] regionesString = Resources.GetStringArray(Resource.Array.regiones_ecuador);
switch (provincia)
{
    case "El Oro":
        regionText = regionesString[1];
        break;
    case "Esmeraldas":
        regionText = regionesString[1];
        break;
```

Segmento de Código 2.22 Extracto del método `ProvinciaRegion`

Las listas de pueblos indígenas, provincias del ecuador y regiones se encuentran dentro del archivo `Strings.xml` en forma de arreglos tipo `string`, ubicado en el directorio Resources\values del proyecto. Un extracto del código XML de este archivo se encuentra en el Segmento de Código 2.23.

```
<string-array name="regiones_ecuador">
  <item>-- Regiones --</item>
  <item>Costa</item>
  <item>Insular</item>
  <item>Oriental</item>
  <item>Sierra</item>
</string-array>
<string-array name="provincias_ecuador">
  <item>-- Provincias --</item>
  <item>Azuay</item>
  <item>Bolívar</item>
  <item>Cañar</item>
```

Segmento de Código 2.23 Extracto del archivo `Strings.xml`

Finalmente, el método `OnGuardarClick` se ejecuta cuando el usuario accione el botón Guardar, verifica todos los campos, comprueba la adquisición de coordenadas del GPS y a través de un `Intent` regresa a la actividad `DatosInformativosActivity`, agregando datos de la comunidad.

```
Intent intent = new Intent();
intent.PutExtra("RegionComunidad", region1);
intent.PutExtra("ProvinciaComunidad", provincia1);
intent.PutExtra("PuebloComunidad", pueblo1);
intent.PutExtra("NombreComunidad", nombreC);
intent.PutExtra("LatitudComunidad", latitud1);
intent.PutExtra("LongitudComunidad", longitud1);
SetResult(Result.Ok, intent);
Finish();
```

Segmento de Código 2.24 Extracto del método `OnGuardarClick`

Si el usuario acciona el botón cancelar, se ejecuta el código del método `OnCancelarClick`, éste cierra la actividad sin guardar ningún dato.

De manera adicional se incluye el método `OcultarTeclado`, permite ocultar el teclado del dispositivo móvil cuando se toque cualquier parte de la pantalla, con el propósito de mejorar la experiencia de interacción del usuario. La codificación de este método se encuentre en el Segmento de Código 2.25.

```
InputMethodManager inputMethodManager = (InputMethodManager)GetSystemService(InputMethodService);
inputMethodManager.HideSoftInputFromWindow(CurrentFocus.WindowToken, 0);
```

Segmento de Código 2.25 Extracto del método `OcultarTeclado`.

`GetSystemService` permite acceder al servicio `InputMethodService`, éste oculta el teclado de pantalla a través del método `HideSoftInputFromWindow` que recibe como parámetro el token de ventana que realizó la solicitud de ocultamiento.

2.3.1.4 Implementación de la actividad Añadir hablante

La actividad Añadir hablante facilita el ingreso de un nuevo hablante, incluidos sus datos. El diseño de pantalla de esta actividad se observa en la Figura 2.36.

Figura 2.36 Interfaz de usuario de la actividad Añadir hablante.

En la Figura 2.37 se ven los detalles de la clase `AnadirHablanteActivity`.

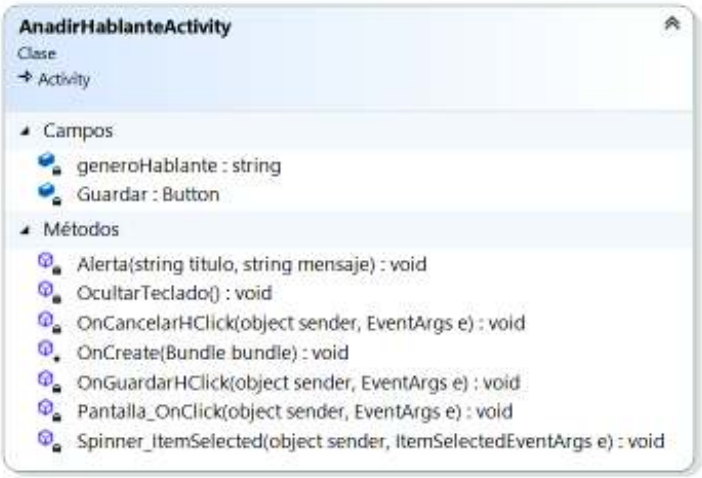


Figura 2.37 Clase correspondiente a la actividad Añadir hablante

La clase `AnadirHablanteActivity` hereda de `Activity`, tiene un funcionamiento similar a `AnadirComunidadActivity`, con la diferencia que ésta no implementa la interfaz `ILocationListener`, razón por la cual en la siguiente sección se realizará la descripción de los métodos `OnCreate` y `Spinner_ItemSelected`.

Todas las clases que heredan de `Activity` deben implementar el método `OnCreate`. En la sección 1.3.6.3 se indica que es un método de tipo devolución de llamada y que está relacionado con la carga de los elementos de IU. En el Segmento de Código 2.26 se observa un fragmento de la programación del método `OnCreate`.

```
SetContentView(Resource.Layout.AnadirHablante);
Button Guardar = FindViewById<Button>(Resource.Id.guardarButtonH);
Guardar.Click += OnGuardarHClick;
FindViewById<Button>(Resource.Id.cancelarButtoH).Click += OnCancelarHClick;
Spinner spinner = FindViewById<Spinner>(Resource.Id.generoSpinner);
spinner.ItemSelected += new  EventHandler<AdapterView.ItemSelectedEventArgs>(Spinner_ItemSelected);
```

Segmento de Código 2.26 Extracto del método `OnCreate`

Todos los diseños de pantalla (*layout*) del sistema Android se guardan en archivos de formato XML, `SetContentView` relaciona la actividad con el nombre del archivo correspondiente. Los diseños de pantalla en formato XML se ubican dentro del directorio del proyecto “Resources\layout”, en el Segmento de Código 2.27 se observa una parte de la codificación XML correspondiente al archivo de diseño `AnadirHablante.axml`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/pantallaAH">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Nombre:"
            android:layout_marginRight="10dp"
            android:layout_marginBottom="6.6dp"
            android:layout_marginTop="14.6dp" />
        <EditText
            android:id="@+id/hablanteInput"
            android:layout_gravity="center"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:inputType="textPersonName"
            android:maxLength="20" />
    </LinearLayout>
```

Segmento de Código 2.27 Extracto del archivo `AnadirHablante.axml`

Todos los elementos IU se encuentran dentro de archivos XML, puede darse el caso que en diferentes archivos se encuentren elementos con el mismo nombre, para evitar

problemas Android genera un ID único para cada elemento IU, `FindViewById` relaciona el ID del elemento con una etiqueta que podrá ser utilizada en código para facilitar la manipulación de estos elementos.

En `OnCreate` también se generan las etiquetas de métodos a ejecutarse cuando ocurra algún evento con relación a los controles para poder realizar alguna acción. Por ejemplo, cuando se toque el botón `Guardar` se llamará al método `OnGuardarHClick`, o al seleccionar un elemento del control de número (*spinner*) se invocará a `Spinner_ItemSelected`.

El método `Spinner_ItemSelected` permite identificar que elemento de la lista se escogió, un fragmento del código de este método se observa en el Segmento de Código 2.28.

```
Spinner spinner = (Spinner)sender;
string generoHablante = spinner.GetItemAtPosition(e.Position).ToString();
```

Segmento de Código 2.28 Extracto del método `Spinner_ItemSelected`.

La variable `generoHablante` es de tipo `string`, la posición del elemento selecciona dentro de la lista se encuentra en `e.Position`, con ésta se obtiene el ítem seleccionado en formato `string` para luego guardarlo en la variable `generoHablante`.

2.3.1.5 Implementación de la actividad Añadir investigador

La implementación de la actividad `AnadirInvestigadorActivity` es similar a `AnadirComunidadActivity`, razón por la cual en esta sección solo se incluye el diseño de pantalla y la estructura de la clase `AnadirInvestigadorActivity`. Toda la codificación de este proyecto se incluye en el Anexo I. En la Figura 2.38 se incluyen los detalles de la clase `AnadirInvestigadorActivity`.

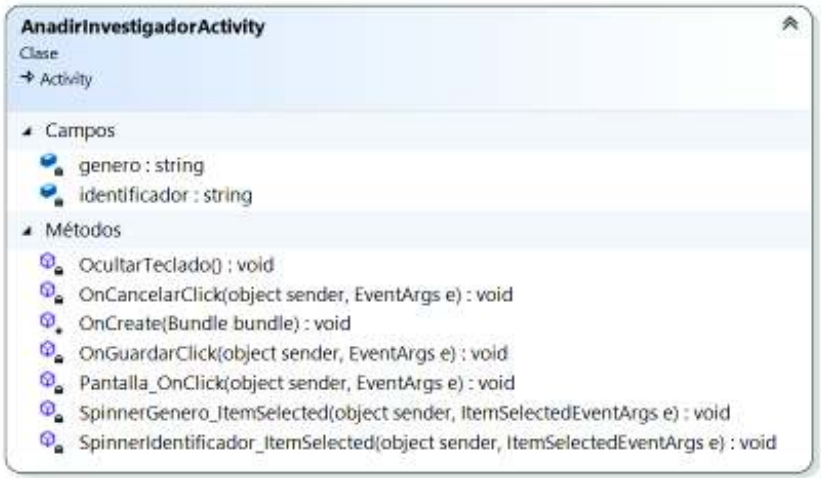


Figura 2.38 Clase `AnadirInvestigadorActivity`

En la Figura 2.39 se observa la pantalla que se despliega para agregar un nuevo investigador.

Figura 2.39 Interfaz de usuario de la actividad Añadir investigador

2.3.1.6 Implementación de la actividad Registrar

Esta actividad permite realizar registros orales de la lengua nativa, muchos de sus métodos tienen codificación similar a otros ya descritos en anteriores secciones. Se dará una breve descripción de su funcionalidad, sin realizar análisis de código. En aquellos métodos que no guarden similitud, se realizará una descripción de las partes más importantes del código.

Las colecciones genéricas de esta clase varían con las creadas en la actividad `DatosInformativosActivity`. En todo el proyecto se tienen seis colecciones genéricas que serán guardadas en el almacenamiento interno. Para esclarecer o evitar cualquier tipo de confusión se resumen estas colecciones genéricas en la Tabla 2.9, incluyendo el tipo de colección, actividad que la crea y nombre del archivo XML que la contiene.

Tabla 2.9 Colecciones genéricas dentro del almacenamiento interno

Nombre	Tipo	Actividad	Archivo XML
`Comunidades`	`List<Comunidad>`	`DatosInformativosActivity`	`comunidades.xml`
`Hablantes`	`List<Hablante>`	`DatosInformativosActivity`	`hablantes.xml`
`Investigadores`	`List<Investigador>`	`DatosInformativosActivity`	`investigadores.xml`
`ComunidadesBaseDatos`	`List<Comunidad>`	`RegistrarActivity`	`comunidadesBD.xml`
`HablantesBaseDatos`	`List<Hablante>`	`RegistrarActivity`	`hablantesBD.xml`
`Investigadores`	`List<Investigador>`	`RegistrarActivity`	`investigadoresBD.xml`
`RegistrosBaseDatos`	`List<Registro>`	`RegistrarActivity`	`registrosBD.xml`

Hay dos colecciones genéricas de tipo `List<Comunidad>`, `Comunidades` almacena la lista de comunidades, ingresadas por el usuario al dispositivo, en cambio `ComunidadesBaseDatos` guarda las que fueron utilizadas en la toma de registros orales. El mismo análisis se tiene para los pares de tipo `List<Hablante>` y `List<Investigador>`.

La colección genérica `RegistrosBaseDatos` guarda datos adicionales a cada registro como fecha y hora de grabación del registro, así como identificadores de comunidad, hablante e investigadores; que serán de utilidad en el servicio de sincronización a la nube. La estructura de la clase `Registro` se observa en la Figura 2.40.

Figura 2.40 Clase `Registro`.

La estructura de la clase `RegistrarActivity` se observa en la Figura 2.41 y el diseño de pantalla de la actividad se encuentra en la Figura 2.42.

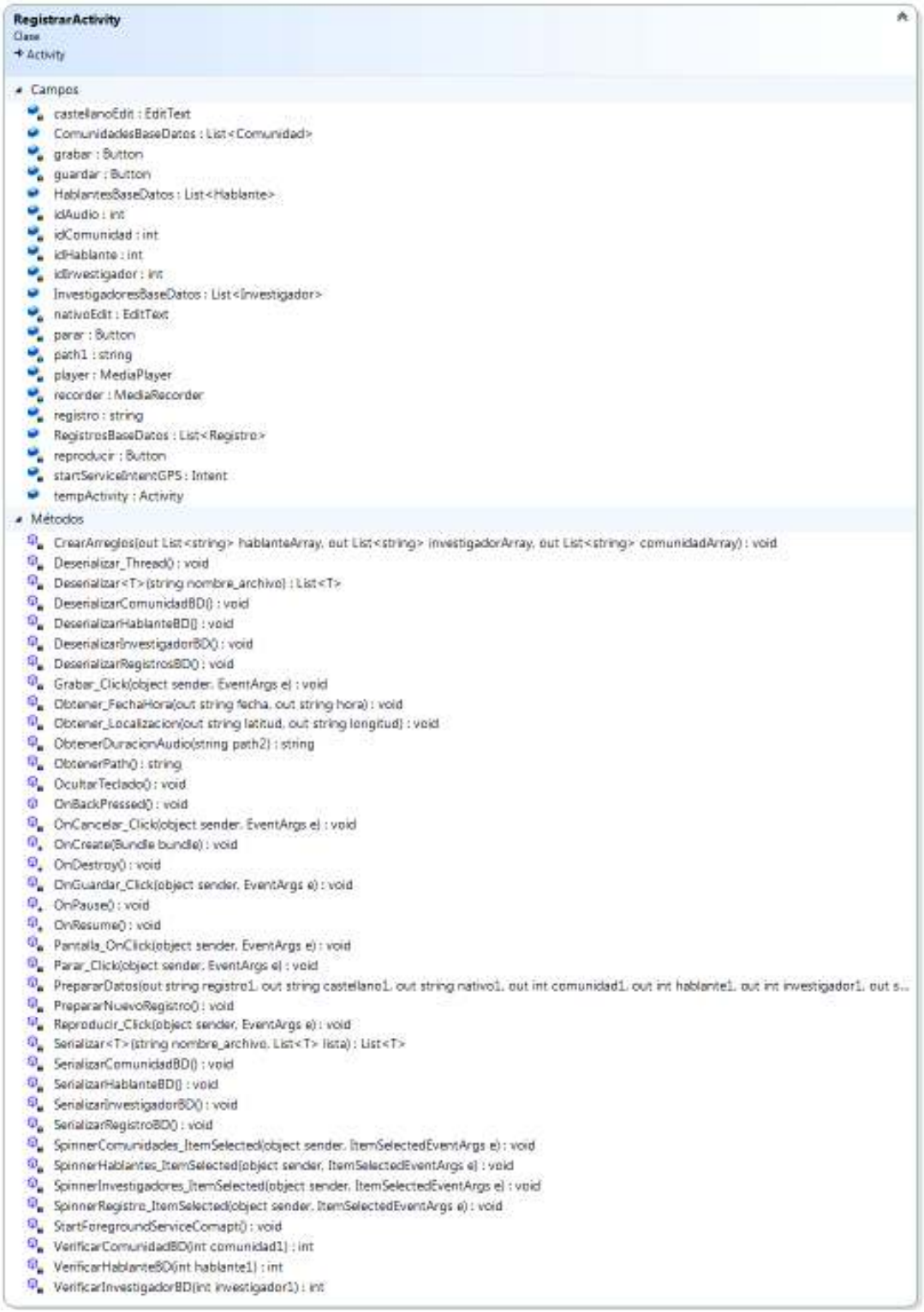


Figura 2.41 Clase `RegistrarActivity`.

Figura 2.42 Interfaz de usuario de la actividad Registrar

El código del método `CrearArreglos` facilita la creación de arreglos que verá el usuario cuando despliegue los controles de número (*spinner*), las listas de comunidades, hablantes e investigadores visualizadas corresponden a las almacenadas en el dispositivo. En el Segmento de Código 2.29 se ve la creación del arreglo tipo `string` que posteriormente se cargará al control de número de hablantes.

```
            hablanteArray = new List<string>
            {
                "-- Hablantes --"
            };
            foreach (var item in DatosInformativosActivity.Hablantes)
                hablanteArray.Add(item.NombreH);
```

Segmento de Código 2.29 Extracto del método `CrearArreglos`

El código de programación del método `Grabar_Click` se invoca con el accionar del botón Grabar, `Grabar_Click` llama a `StartForegroundServiceComapt`, `ObtenerPath` y al servicio en segundo plano `ServicioGps`, el cual permite obtener la ubicación del dispositivo. Las funcionalidades de estos métodos son invocar al servicio `ServicioGps` y obtener el directorio del nuevo archivo de audio a grabarse con el nombre `path`, respectivamente.

El servicio `ServicioGps` se ejecuta en segundo plano, no interfiere con la interfaz gráfica ni con la adquisición de audio. Luego, la codificación de `Grabar_Click` crea a `recorder`, una nueva instancia de la clase `MediaRecorder`, invoca a varios métodos de la misma, cuya finalidad es establecer los parámetros de la grabación a realizarse, en el Segmento de Código 2.30 se observan estas configuraciones.

```
recorder.Reset();
recorder.SetAudioSource(AudioSource.Mic);
recorder.SetAudioSamplingRate(44100);
recorder.SetAudioEncodingBitRate(96000);
recorder.SetOutputFormat(OutputFormat.Mpeg4);
recorder.SetAudioEncoder(AudioEncoder.Aac);
```

Segmento de Código 2.30 Extracto del método `Grabar_Click`

Luego de establecer las configuraciones necesarias, la programación del método `Grabar_Click` empieza el proceso de grabación como se observa en el Segmento de Código 2.31.

```
recorder.SetOutputFile(path);
recorder.Prepare();
recorder.Start();
grabar.Enabled = !grabar.Enabled; //Inhabilita el botón grabar
parar.Enabled = !parar.Enabled; //Habilita el botón parar
```

Segmento de Código 2.31 Extracto del método `Grabar_Click`

La programación del método `OnGuardar_Click` se ejecuta cuando el usuario acciona el botón Guardar, verifica que todos los campos de los metadatos estén llenos y crea un nuevo elemento de tipo `Registro` en la colección genérica `RegistrosBaseDatos`. Antes de crear el nuevo ítem de la lista de registros, se deben ejecutar algunos métodos para la creación de metadatos adicionales. El método `Obtener_FechaHora` permite guardar la fecha y hora en la que se realizó el registro, su codificación se encuentra en el Segmento de Código 2.32.

```
private static void Obtener_FechaHora(out string fecha, out string hora)
{
    DateTime currentTime = DateTime.Now;
    fecha = currentTime.ToString("yyyy-MM-dd");
    hora = currentTime.ToString("HH:mm:ss");
}
```

Segmento de Código 2.32 Método `Obtener_FechaHora`

El código del método `ObtenerDuracionAudio` permite medir la duración del archivo de audio recién generado y la programación de `Obtener_Localizacion` adquiere las coordenadas de latitud y longitud otorgadas por el servicio en segundo plano `ServicioGps`. Los métodos `VerificarHablanteBD`, `VerificarInvestigadorBD` y `VerificarComunidadBD` tienen como función determinar si el hablante, investigador y comunidad asociadas al registro, ya participaron en anteriores grabaciones; estas verificaciones se realizan a través de

expresiones lambda[14], un extracto del método `VerificarHablanteBD` se observa en el Segmento de Código 2.33.

```
private static int VerificarHablanteBD(int hablante1)
{
    Hablante y = DatosInformativosActivity.Hablantes[hablante1];
    int hablante2 = HablantesBaseDatos.FindIndex(a => a.NombreH == y.NombreH &
    a.GeneroH == y.GeneroH & a.EdadH == y.EdadH);
```

Segmento de Código 2.33 Extracto del método `VerificarHablanteBD`.

Finalmente se guarda el registro, usando un nuevo hilo de programación a través del método `serializar_Thread` que guarda los registros en el almacenamiento interno del dispositivo. En la Figura 2.43 se observa el mensaje de notificación que será desplegado después de guardar el registro.

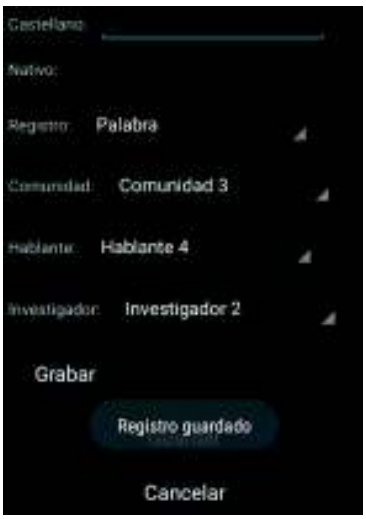


Figura 2.43 Mensaje de notificación al guardar un registro

La programación del método `PrepararNuevoRegistro` tiene como finalidad permitir al recolector de datos realizar nuevas grabaciones al mismo hablante, evitando elegir ciertos campos como se ve en la Figura 2.43, esto agiliza el proceso de adquisición de registros.

2.3.1.7 Implementación de la actividad Registros

Esta actividad permite visualizar los registros que se encuentran en el almacenamiento interno del dispositivo, reproducirlos y a través de una barra de búsqueda verificar visualmente el avance del audio. En la Figura 2.44 se ve el diseño de pantalla correspondiente a esta actividad, en la lista de registros se observa una variable que determinará si el registro ya fue subido a la nube.

[14] Funciones anónimas que pueden ser usadas para crear delegados.

Figura 2.44 Interfaz de usuario de la actividad Registros.

Los detalles de la clase correspondiente a la actividad Registros se ven en la Figura 2.45, ésta tiene tres métodos privados y cuatro de devolución de llamada.

La codificación del método `ListRegistros_ItemClick` se ejecutará cuando el usuario seleccione algún registro de la lista, el código permite la reproducción del audio asociado al registro, también carga en pantalla la duración de este. Para la actualización del contenido de cualquier elemento gráfico, se recomienda usar el hilo principal del programada, mediante `RunOnUiThread`. Un extracto del código descrito se ve en el Segmento de Código 2.34.

```
            player = new MediaPlayer();
            string path = RegistrarActivity.RegistrosBaseDatos[e.Position].Path;
            RunOnUiThread(() =>
            {
                total.Text = RegistrarActivity.RegistrosBaseDatos[e.Position].Duracion;
            });
            Reproducir(path);
```

Segmento de Código 2.34 Extracto del método `ListRegistros_ItemClick`

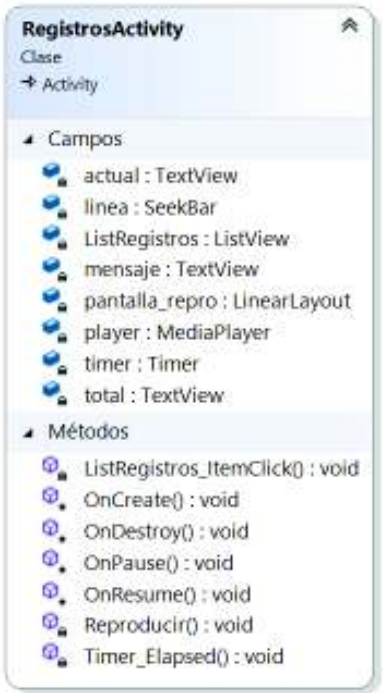


Figura 2.45 Clase correspondiente a la actividad Registros

El método `Reproducir` recibe como parámetro la ruta del archivo de audio a reproducir, luego inicia al temporizador `timer` que se desbordará cada 20 milisegundos para ejecutar el código de `Timer_Elapsed`, este permite actualizar la posición de la barra de búsqueda conforme al avance de la reproducción. Una parte del código del método `Timer_Elapsed` se observa en el Segmento de Código 2.35.

```
if (player.CurrentPosition <= player.Duration)
{
    TimeSpan tiempo = TimeSpan.FromMilliseconds(player.CurrentPosition);
    string duracion = tiempo.ToString(@"mm\:ss\.ff");
    RunOnUiThread(() =>
    {
        try
        {
            linea.Progress = player.CurrentPosition;
            actual.Text = duracion;
```

Segmento de Código 2.35 Extracto del método `Timer_Elapsed`

Cuando el avance de la reproducción alcance la duración total del archivo de audio, el temporizador será parado, así la ejecución del método `Timer_Elapsed` se dará solo cuando exista alguna reproducción en curso.

2.3.1.8 Implementación de la actividad Acerca de

Esta actividad permite desplegar en pantalla información relevante sobre la aplicación, el diseño gráfico de su pantalla se observa en Figura 2.46.

Figura 2.46 Interfaz de usuario de la actividad Acerca de.

En la Figura 2.47 se observan los detalles de la clase `AcercaDeActivity`, el método de devolución de llamada `OnCreate` siempre se ejecuta en todas las actividades, es responsable de cargar todos los elementos de la interfaz gráfica.

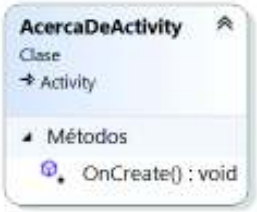


Figura 2.47 Clase correspondiente a la actividad Acerca de.

2.3.1.9 Implementación del servicio nube

El servicio nube se ejecutará en segundo plano, su principal función es subir todos los archivos de audio a un contenedor tipo BLOB alojado en la nube. Se encargará de crear las tablas que serán usadas en la base de datos de tipo no relacional. En estas tablas se almacenarán todos los datos y metadatos de los archivos.

Los servicios en segundo plano no tienen interfaz gráfica, pero usan notificaciones locales para informar al usuario de su ejecución. El código del servicio `ServicioNube` correrá cuando se active el interruptor de sincronización de la actividad principal, luego se mostrará en la barra de notificaciones que ha iniciado el servicio.

En la parte superior izquierda de la Figura 2.48 se observa el icono de la notificación local, el usuario puede desplegar hacia abajo la barra para ver más detalles.

Figura 2.48 Notificación local del servicio Nube

Cuando inicie la sincronización de archivos a la nube, el usuario verá los detalles de la notificación local de la Figura 2.49.

Figura 2.49 Notificación local al iniciar el servicio Nube.

Una vez establecida la conexión con la nube, el contenido de la notificación local cambiará, el usuario podrá verificar el avance de la sincronización, inclusive cuando la aplicación este en pausa, el aviso no desaparecerá hasta que haya finalizado el servicio. Cada actualización de la notificación local será similar a la Figura 2.50.

Figura 2.50 Actualización de la notificación local del servicio Nube.

El servicio nube puede iniciar o finalizar en cualquier instante, a criterio del usuario. Los archivos se sincronizarán a la nube mediante conexión wifi o si el usuario lo prefiere usando los datos móviles de su dispositivo.

Los detalles de la clase `ServicioNube` se observan en la Figura 2.51.

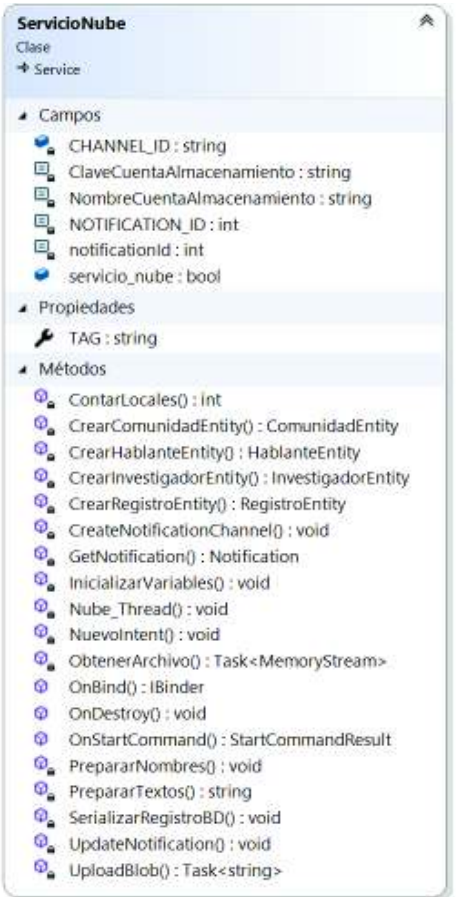


Figura 2.51 Clase `ServicioNube`

Desde el nivel de API 26 (Android Oreo) o superiores, las notificaciones locales de los servicios en segundo plano serán gestionadas a través de canales de notificación, generados por el código del método `CrearCanalNotificacion`. Si el nivel de API es menor a 26, se prescinde de crear estos canales. En el Segmento de Código 2.36 se observa la creación de un canal de notificación local.

```
NotificationChannel channel = new NotificationChannel(ID_Canal, nombreCanal, NotificationImportance.Default)
{
Description = descripcionCanal
};
NotificationManager notificationManager = (NotificationManager)GetSystemService(NotificationService);
notificationManager.CreateNotificationChannel(channel);
```

Segmento de Código 2.36 Extracto del método `CrearCanalNotificacion`.

Después de generar el canal, se crea la notificación local correspondiente al servicio `ServicioNube`, aquí se establecen las propiedades de título, contenido e icono. En el Segmento de Código 2.37 se establecen estas propiedades.

```
Notification notificacion = new NotificationCompat.Builder(this,ID_Canal)
        .SetSmallIcon(Resource.Drawable.ic_cloud_upload_white_24dp)
        .SetContentTitle("Sincronización Nativa")
        .SetContentText("Iniciando...")
        .SetOngoing(true)
        .Build();
StartForeground(ID_Notificacion, notificacion);
```

Segmento de Código 2.37 Creación notificación local para el servicio `ServicioNube`.

En la definición de campos de la clase `ServicioNube`, se establecen dos constantes de tipo `string` que servirán para establecer conexión con los servicios en la nube, estas constantes pertenecen al nombre de la cuenta de almacenamiento de Azure y clave de acceso. El procedimiento para la creación de la cuenta de almacenamiento se adjunta en el Anexo III.

En el Segmento de Código 2.38 se ve un extracto de la definición de campos de la clase, por cuestiones de privacidad la clave de la cuenta de almacenamiento ha sido ocultado de la imagen.

```
const string NombreCuentaAlmacenamiento = "nativapruebas";
const string ClaveCuentaAlmacenamiento = "**********************************************";
```

Segmento de Código 2.38 Campos de acceso a los servicios en la nube de Azure.

Estos campos serán útiles en la creación de las credenciales de almacenamiento, con las cuales se puede crear un objeto de tipo `CloudStorageAccount`, este permite conectar a la cuenta de almacenamiento Azure mediante código. En el Segmento de Código 2.39 se incluye la creación del objeto, es importante establecer el segundo parámetro en `true`, ya que esto permite que la comunicación con los servicios en la nube se dé a través del protocolo de transferencia segura HTTPS.

```
StorageCredentials storageCredentials = new StorageCredentials(NombreCuentaAlmacenamiento,
                                                               ClaveCuentaAlmacenamiento);
CloudStorageAccount storageAccount = new CloudStorageAccount(storageCredentials, true);
```

Segmento de Código 2.39 Extracto del método `Nube_Thread`

Con el objeto `storageAccount` de tipo `CloudStorageAccount` se generan los clientes que se conectarán a los diferentes servicios de la cuenta de almacenamiento. En este proyecto se usan tres clientes para conectarse a los servicios de contenedores, tablas y colas respectivamente. La generación de estos clientes se ve en el Segmento de Código 2.40.

```
CloudQueueClient clientQueue = storageAccount.CreateCloudQueueClient();
CloudBlobClient clientBlob = storageAccount.CreateCloudBlobClient();
CloudTableClient clientTable = storageAccount.CreateCloudTableClient();
```

Segmento de Código 2.40 Extracto del método `Nube_Thread`.

Mediante el `CloudBlobClient` se accede a la lista completa de todos los contenedores de la cuanta de almacenamiento, para acceder a un ítem en específico se necesita de la creación de un `CloudBlobContainer`. De la misma forma `CloudTableClient` accede a todas las tablas de la cuenta de almacenamiento y a través de `CloudTable` se apunta al elemento de tipo NoSQL en específico. Similar comportamiento para `CloudQueueClient` que tendrá acceso a todos los servicios de cola de mensajes y a través de `CloudQueue` se apunta a un ítem en específico.

Los archivos de audio de los registros se guardarán en un contenedor tipo BLOB, la información de datos y metadatos en tres tablas NoSQL, los nombres y el tipo de objeto del servicio en la nube para estos elementos se especifican en la Tabla 2.10.

Tabla 2.10 Contenedor y tablas alojadas en la nube.

Descripción	Tipo	Nombre
Contenedor	`CloudBlobContainer`	audios
Tabla NoSQL	`CloudTable`	Registros
Tabla NoSQL	`CloudTable`	Comunidades
Tabla NoSQL	`CloudTable`	Hablantes
Tabla NoSQL	`CloudTable`	Investigadores
Mensajes de cola	`CloudQueue`	id-registros

En el Segmento de Código 2.41 se especifica la creación del contenedor “audios” y de la tabla “Registros”, se utilizan métodos asincrónicos que permiten ejecutar en un hilo diferente la creación de los objetos sin interferir con el programa principal.

```
CloudBlobContainer contenedor = clientBlob.GetContainerReference("audios");
await contenedor.CreateIfNotExistsAsync();

CloudTable tablaRegistros = clientTable.GetTableReference("Registros");
await tablaRegistros.CreateIfNotExistsAsync();
```

Segmento de Código 2.41 Extracto del método `Nube_Thread`.

Para subir archivos a la nube, se realiza un barrido en todos los registros del almacenamiento interno. Los registros tienen una propiedad de tipo booleano `Nube`, si su contenido es verdadero significa que el registro se encuentra en la nube, caso contrario en el almacenamiento interno. Se suben los registros que tengan la propiedad `Nube` en falso.

Para tener identificadores únicos, se utiliza un sistema de mensajes de cola, proporcionado dentro de los servicios de las cuentas de almacenamiento de Azure. El cliente `CloudQueueClient` tiene acceso al servicio de colas `queue`. En la variable de tipo mensaje de colas `message`, se guarda el identificador único, luego esta variable es pasada como argumento al método `PrepararNombres`, este devolverá los nombres a utilizar en el blob de audio y registro de tablas NoSQL. Al finalizar este proceso, el mensaje de colas será borrado de la nube para evitar duplicidad en los identificadores. Lo mencionado se describe en el Segmento de Código 2.42.

```
CloudQueueMessage message = await queue.GetMessageAsync();
PrepararNombres(message, out string nombre_blob, out string nombre_registro);
await queue.DeleteMessageAsync(message);
```

Segmento de Código 2.42 Extracto del método `Nube_Thread`.

El Segmento de Código 2.43 incluye la programación necesaria para subir un archivo de audio a la nube y el proceso para guardar una nueva entidad[15] en las tablas NoSQL.

```
string path = registrosBD[i].Path;
MemoryStream archivo = await ObtenerArchivo(path);
string url = await UploadBlob(nombre_blob, archivo,contenedor);

RegistroEntity registroT = CrearRegistroEntity(registrosBD, i, nombre_registro, url);
TableOperation insertarR = TableOperation.Insert(registroT);
await tablaRegistros.ExecuteAsync(insertarR);
```

Segmento de Código 2.43 Extracto del método `Nube_Thread`.

La colección genérica `registrosBD` contiene los registros que serán subidos a la nube, en la variable `path`, se guarda la dirección que tiene el archivo de audio en el almacenamiento interno, ésta es usada como parámetro del método `ObtenerArchivo`, el cual regresa un arreglo de bytes con la información del archivo. Es importante que todos los arreglos de

[15] Representación de un objeto dentro de una base de datos

este tipo sean manejados en la memoria del dispositivo, evitando cualquier corrupción en el archivo original y agilizando la subida de audios a la nube.

El método asíncrono `UploadBlob` recibe tres parámetros que son: el nombre que tendrá el archivo dentro del contenedor en la nube, el flujo de memoria del archivo y el nombre del contendedor al que apunta. Con estos parámetros se envía el audio a la nube, luego se regresa el URI (*Uniform Resource Identifier*, Identificador Uniforme de Recurso) del archivo subido. Parte del código de este método se ve en el Segmento de Código 2.44.

```
private async Task<string> UploadBlob(string name,Stream stream,CloudBlobContainer contenedor)
{
    string url = "";
    try
    {
        CloudBlockBlob audioBlob = contenedor.GetBlockBlobReference(name);
        using (stream)
        {
            await audioBlob.UploadFromStreamAsync(stream);
            url = audioBlob.StorageUri.PrimaryUri.AbsoluteUri;
```

Segmento de Código 2.44 Extracto del método `UploadBlob`.

El método genérico `CrearRegistroEntity` es invocado recibiendo cuatro parámetros, que son: lista genérica, identificador, nombre retornado por el método `PrepararNombres` y el URI del blob de audio; almacenado ya en la nube. Con estos parámetros se crea un objeto de tipo `RegistroEntity`, esta clase es similar a `Registro`, a diferencia que hereda de `TableEntity`. Todas las inserciones en las tablas NoSQL deben heredar de `TableEntity` para poder usar sus métodos. En el Segmento de Código 2.43 se ve la inserción de un nuevo registro en la tabla NoSQL "Registros", a través del `CloudTable` llamado `tablaRegistros`.

La inserción de nuevos hablantes, comunidades e investigadores en sus respectivas tablas NoSQL, es muy similar a lo expuesto en la inserción de registros, los métodos genéricos `CrearHablanteEntity`, `CrearInvestigadorEntity` y `CrearComunidadEntity` cumplen la misma funcionalidad que `CrearRegistroEntity`. La codificación detalla de estos métodos se incluye en el ANEXO I.

Cuando se hayan ejecutado con éxito todos los métodos asíncronos que intervienen en la subida de audios a la nube e inserciones en las tablas, se cambia la propiedad booleana `Nube` del registro a verdadero para evitar duplicados en próximas sincronizaciones. También se serializan los nuevos registros, garantizando que todos los campos booleanos `Nube` estén actualizados en el almacenamiento interno.

2.3.1.10 Implementación del servicio GPS

El servicio GPS se ejecutará en segundo plano, se activa cuando el usuario ingresa a la actividad Registrar o si empieza a grabar un nuevo audio, cualquiera de estos eventos intentará acceder a la localización geográfica del dispositivo. La codificación de la creación de notificaciones locales es similar a lo explicado en la sección 2.3.1.9.

Los servicios heredan de la clase abstracta `Service` para poder acceder a los métodos que permiten el funcionamiento en segundo plano. El servicio `ServicioGps` implementa la interfaz `ILocationListener` para poder acceder a los datos del GPS. En la Figura 2.52 se ven los detalles de la clase `ServicioGps`.

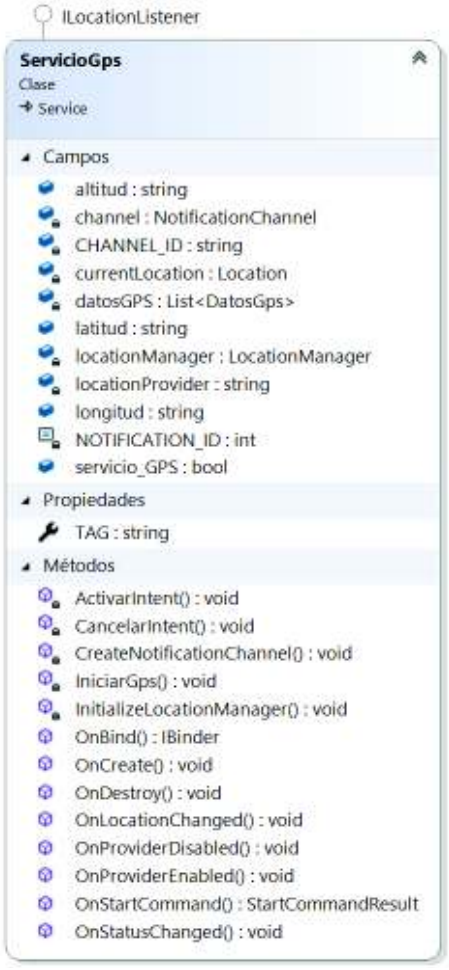


Figura 2.52 Clase `ServicioGps`.

El código de programación del servicio GPS se encuentra en el ANEXO I, es similar al explicado en la actividad Añadir comunidad (sección 2.3.1.3), con la diferencia que al ejecutarse en segundo plano no interfiere con la interfaz de usuario. En 2.3.1.3, al añadir

una nueva comunidad, el programa está diseñado para que permita guardar la comunidad, solo si se accedió a datos del GPS, por requerimientos del proyecto.

Podría darse el caso de no acceder a datos del GPS porque las mediciones fueron realizadas desde interiores de edificaciones con paredes gruesas y se estaba alejado de aberturas de puertas o ventanas, o también cuando se esté en lugares con condiciones irregulares y no exista línea de vista moderada hacia el firmamento, por ejemplo, en alguna quebrada o cañón. El servicio GPS permite guardar registros en estas condiciones, prescindiendo de los datos del GPS con la finalidad de dar fluidez al ingreso de información. Si al momento de realizarse la grabación de audio de un nuevo registro, no se accedió a los datos del GPS con éxito, se tomarán las localizaciones más cercanas y de no disponer de estos datos, se prescinde de estas ubicaciones, ya que el principal objetivo de la clase Registrar es guardar el registro de voz adquirido del hablante.

La notificación local del servicio GPS cuando se ejecute en segundo plano será similar a la Figura 2.53.

Figura 2.53 Notificación local del `ServicioGps`.

2.3.1.11 Implementación del programa Creador de mensajes de cola.

Los mensajes del servicio de cola serán utilizados para la creación de identificadores únicos, tanto para nombres de archivos de audio como para entradas de las tablas NoSQL. Es preferible automatizar la creación de estos mensajes de cola. Para esto se utiliza un proyecto de escritorio clásico de Windows aplicación de consola, usando el IDE de Visual Studio y como lenguaje de programación C#. La creación e instalación de librerías necesarias se adjunta en el ANEXO IV.

En el Segmento de Código 2.45 se observa la conexión a la cuenta de almacenamiento, a través del `CloudConfigurationManager`. Se usa un `string` llamado `StorageConnectionString`, que guarda una cadena de conexión, conformada por: nombre de la cuenta de almacenamiento, clave de acceso y habilitación de trasferencia segura a través del protocolo HTTPS. Estas configuraciones se guardan en el archivo `App.config` dentro del proyecto `CreadorMensajesCola`.

```
CloudStorageAccount storageAccount =
CloudStorageAccount.Parse(CloudConfigurationManager.GetSetting("StorageConnectionString"));
```

Segmento de Código 2.45 Conexión a la cuenta de almacenamiento

En el Segmento de Código 2.46 se ve la creación del cliente `CloudQueueClient` y de los mensajes de cola tipo `CloudQueue`, similar a los accesos a la nube creados en 2.3.1.9. Luego se invoca al método `GenerarColas`, recibe como parámetros el nombre del servicio de mensajería y dos variables de tipo `int`, que delimitarán los mensajes de cola.

```
CloudQueueClient cliente = storageAccount.CreateCloudQueueClient();
CloudQueue colaIdRegistros = cliente.GetQueueReference("id-registros");
colaIdRegistros.CreateIfNotExists();

GenerarColas(colaIdRegistros,01,99);
```

Segmento de Código 2.46 Extracto del programa principal `CreadorMensajesCola`.

El método `GenerarColas` añade mensajes al servicio de colas, si la nueva inserción ocurre con éxito, se desplegará en la consola la expresión “Completado:”, seguido del contenido del mensaje añadido a la nube. Lo descrito se ve en el Segmento de Código 2.47 y la correspondiente salida de consola se observa en la Figura 2.54.

```
private static void GenerarColas(CloudQueue cola, int min, int max)
{
    for (int i = min; i <= max; i++)
    {
        string mensaje = "-EPN-";
        mensaje = mensaje+i.ToString("D3");
        cola.AddMessage(new CloudQueueMessage(mensaje));
```

Segmento de Código 2.47 Extracto del método `GenerarColas`.

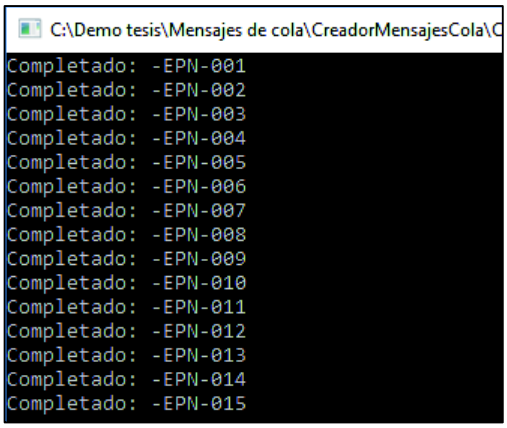


Figura 2.54 Salida de consola al agregar los mensajes en la nube.

2.3.1.12 Implementación de la visualización a través de Excel.

Para poder visualizar en Excel los datos de las tablas NoSQL, alojadas en la cuenta de almacenamiento en la nube, se deben seguir los siguientes pasos:

1. Seleccionar la pestaña Datos en la barra de herramientas de Excel.
2. Seleccionar Obtener datos/ Desde Azure/ Desde un almacenamiento de tabla de Azure.
3. Ingresar la cuenta de almacenamiento y la clave de acceso.
4. Elegir la tabla NoSQL de la cuenta de almacenamiento.

La ejecución de los dos primeros pasos se observa en la Figura 2.55.

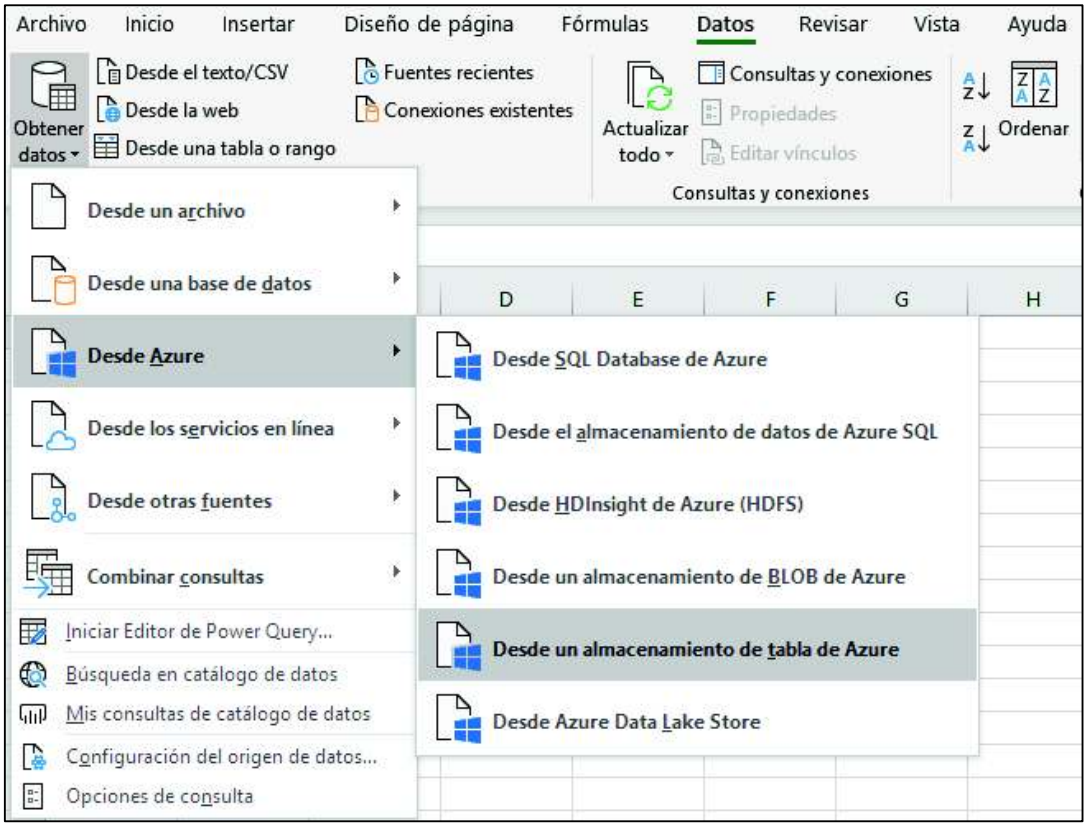


Figura 2.55 Conexión Excel con la cuenta de almacenamiento.

La Figura 2.56 muestra la ejecución de la tercera instrucción, mientras que el cuarto paso se indica en la Figura 2.57.

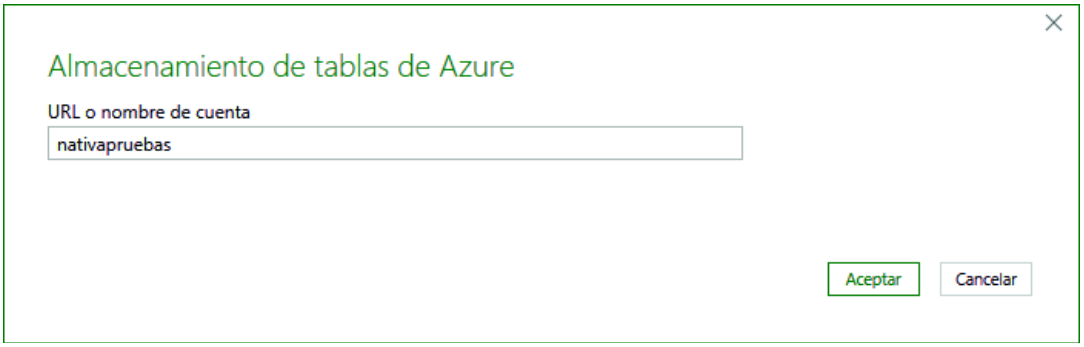


Figura 2.56 Ingreso de credenciales de autenticación.

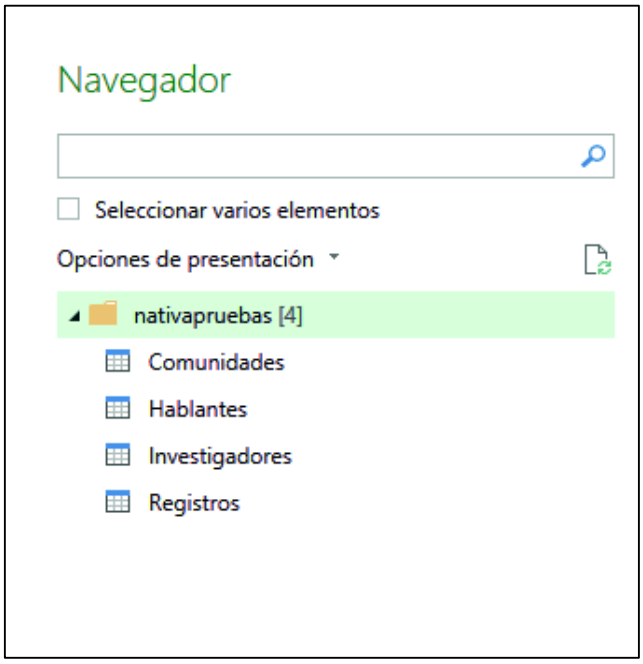


Figura 2.57 Selección de datos a visualizar.

Todos los pasos mencionados se realizan una sola vez, al establecer la primera conexión con la cuenta de almacenamiento. En adelante, no es necesario volver a establecer la conexión, si se desea actualizados los datos, se debe seleccionar la opción Actualizar, dentro de la pestaña Datos en la barra de herramientas Excel.

En la sección 3.2.1.1 se verán los datos de las diferentes tablas NoSQL, importados al archivo Excel.

3. RESULTADOS Y DISCUSIÓN

En este capítulo se detallan las pruebas de funcionamiento del prototipo multimedia, realizadas en los predios de la Escuela Politécnica Nacional[16] (EPN), se verificó la adquisición de datos del GPS, subida de archivos de audio a la nube e inserciones de datos en las diferentes tablas NoSQL. También se describe la puesta en marcha del proyecto, se realizaron registros de palabras quichuas a varios hablantes nativos pertenecientes a la comunidad San Francisco de Chibuleo.

Para las pruebas en los predios de la EPN y la puesta en marcha en la comunidad indígena, se seleccionó un grupo de 7 estudiantes de la carrera de Ingeniería en Electrónica y Telecomunicaciones, se realizaron encuestas para validar el funcionamiento de la aplicación y evaluar la realización del proyecto.

3.1 Pruebas de funcionamiento en predios de la EPN

Para probar el funcionamiento de la aplicación Android y la conectividad con la nube, se registraron 42 registros en diferentes puntos de las inmediaciones de la EPN, utilizando siete dispositivos móviles Android, cuyos detalles se observan en la Tabla 3.1.

Tabla 3.1 Información de los dispositivos móviles utilizados en las pruebas.

Nombre dispositivo	Modelo	Versión Android	Nivel API
Samsung Galaxy J5	SM-J500M	Marshmallow 6.0.1	23
Samsung Galaxy J7	SM-J700M	Marshmallow 6.0.1	23
Xiaomi	Redmi Note 4	Nougat 7.0	24
Huawei P10 lite	WAS-LX3	Nougat 7.0	24
Samsung Galaxy J5	SM-J510MN	Nougat 7.1.1	25
Samsung Galaxy S7 Edge	SM-G935FD	Oreo 8.0	26
Samsung Galaxy S8+	SM-G955FD	Oreo 8.0	26

En la prueba se realizó una analogía entre diferentes lugares de la EPN, simulando ser comunidades dentro del pueblo indígena Kitukara, los sitios se encuentran detallados en la Tabla 3.2 . En cada lugar se guardaron 6 registros, Se designó un sitio con su respectivo color a cada estudiante. La finalidad de estas pruebas es de verificar el almacenamiento de los datos en las tablas NoSQL y comprobar que todos los audios se guarden en el contenedor tipo BLOB, alojado en la nube.

[16] La Escuela Politécnica Nacional, es una Institución de Educación Superior, universidad pública ubicada en Quito, Ecuador. https://www.epn.edu.ec

Tabla 3.2 Lugares EPN con su respectivo color

Estudiante	Lugar EPN	Color
Alvarado Diego	CEC	
Cajas Kevin	Nivelación	
Gongora Kevin	Sistemas	
Guerrero Roberto	Eléctrica	
Palate Paul	Canchas	
Quishpe Alexis	Teatro	
Ríos Diego	Civil	

En el sector asignado para cada estudiante, se registraron 6 grabaciones de audio en los alrededores del sitio establecido. En la Figura 3.1, se observan las localizaciones geográficas de los lugares donde se realizaron los registros con su respectivo color.

Figura 3.1 Ubicación de las pruebas en las inmediaciones de la EPN

El funcionamiento de las inserciones a las tablas de datos y la subida de audios al contenedor se comprueba a través del portal de Azure; previa autenticación, se constata la creación de tablas y para verificar las inserciones realizadas se usa Microsoft Azure Storage Explorer[17].

[17] Permite la administración de cuentas de almacenamiento Azure, se puede crear, eliminar, editar y visualizar recursos de almacenamiento.

En la Figura 3.2, se presentan los primeros archivos de audio almacenados en la nube dentro del contenedor audios, esta verificación se realizó desde el portal de Azure[18].

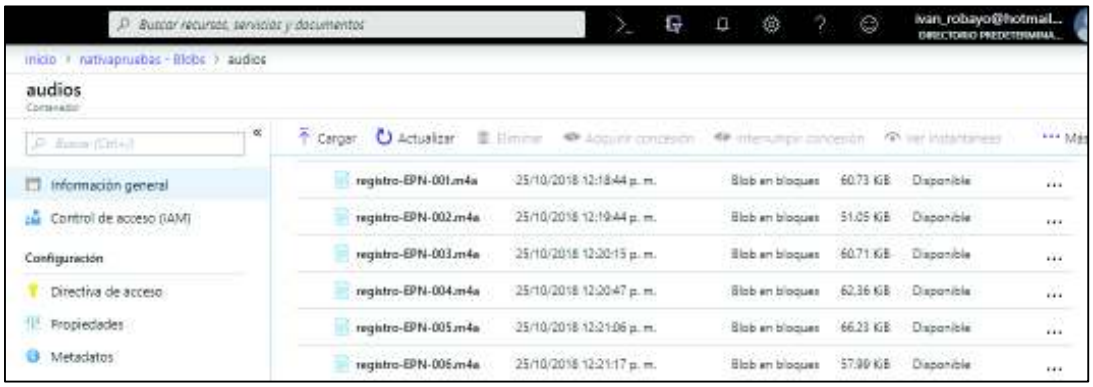


Figura 3.2 Verificación subida de archivos de audio a la nube.

En la Figura 3.3 se comprueba la creación de las cuatro tablas NoSQL, ubicadas en la cuenta de almacenamiento nativapruebas, cada una con su respectivo nombre y URL (*Uniform Resource Locator*, Localizador Uniforme de Recursos).

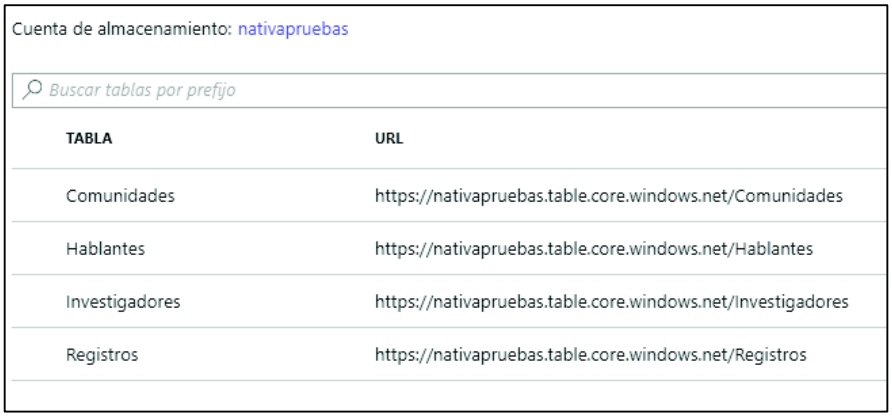


Figura 3.3 Verificación creación de tablas NoSQL.

Para observar el contenido de las tablas, es necesario abrir el programa de escritorio Microsoft Azure Storage Explorer. En la Figura 3.4 y Figura 3.5 se observan las propiedades correspondientes a dos entidades de la tabla Registros, se divide la captura de pantalla en dos imágenes por la cantidad de propiedades.

PartitionKey	RowKey	Timestamp	TipoRegistro	Castellano	Nativo	Duracion	Fecha
Registro-EPN-001	registro	2018-10-25T17:18:45.348Z	Palabra	P1-Eléctrica	T1-Eléctrica	00:05.03	2018-10-25
Registro-EPN-002	registro	2018-10-25T17:19:46.790Z	Palabra	P2-Eléctrica	T2-Eléctrica	00:04.22	2018-10-25

Figura 3.4 Verificación de inserción de registros en la base de datos parte 1.

[18] https://portal.azure.com

Hora	Latitud	Longitud	Link
12:12:39	-0.209043410771671	-78.4896129342294	https://nativapruebas.blob.core.windows.net/audios/registro-EPN-001.m4a
12:14:03	-0.209401338520325	-78.489545849289	https://nativapruebas.blob.core.windows.net/audios/registro-EPN-002.m4a

Figura 3.5 Verificación de inserción de registros en la base de datos parte 2.

En la Figura 3.6 se presentan todas las propiedades correspondientes a dos entidades de la tabla Comunidades, lugares utilizados para las pruebas realizadas en la EPN.

PartitionKey	RowKey	Comunidad	Pueblo	Region	Provincia	Latitud	Longitud
Registro-EPN-006	comunidad	Eléctrica	Kitukara	Sierra	Pichincha	-0.209425159672768	-78.4892116339602
Registro-EPN-007	comunidad	CEC	Kitukara	Sierra	Pichincha	-0,209335474414806	-78,4887542012268

Figura 3.6 Verificación de inserción de comunidades en la base de datos.

La Figura 3.7 permite comprobar la inserción de nuevas entidades en la tabla Hablantes, incluyendo las propiedades conforme a los requerimientos del sistema. Todas las entidades de las tablas NoSQL de Azure tienen la propiedad `Timestamp`, ésta permite llevar un registro de tiempo de la última modificación de la entidad del lado del servidor. Esta propiedad se agrega automáticamente en cada inserción y se autoincrementa con cada actualización.

PartitionKey	RowKey	Timestamp	Nombre	Edad	Genero
Registro-EPN-006	hablante	2018-10-25T17:21:18.907Z	Roberto Guerrero	24	Masculino
Registro-EPN-007	hablante	2018-10-25T17:21:25.206Z	Diego Alvarado	22	Masculino

Figura 3.7 Verificación inserción de hablantes en la base de datos.

Las inserciones de entidades dentro de la tabla Investigadores se realizaron con éxito, como se puede apreciar en la Figura 3.8 donde se encuentran las respectivas propiedades.

PartitionKey	RowKey	Timestamp	Nombre	Identificacion	Genero
Registro-006	investigador	2018-10-28T16:46:41.559Z	Diego Alvarado	Recolector	Masculino
Registro-007	investigador	2018-10-28T16:46:56.297Z	Diego Alvarado	Recolector	Masculino

Figura 3.8 Verificación de inserción de investigadores en la base de datos.

Los 42 archivos de audio correspondientes a las pruebas realizadas en la EPN se encuentran en el CD adjunto al presente proyecto de titulación, dentro de la carpeta ANEXOS VIII.

3.2 Puesta en marcha del proyecto Nativa

El domingo 28 de octubre de 2018, fue la puesta en marcha del proyecto, realizando pruebas de funcionamiento del prototipo multimedia que permite el registro de la lengua nativa quichua. Las pruebas fueron realizadas en la comunidad San Francisco de Chibuleo, perteneciente a la nacionalidad Kichwa del país Ecuador. San Francisco se ubica en la provincia de Tungurahua, región sierra.

En total se realizó el registro de 356 palabras de la lengua nativa, desde siete dispositivos Android, detalles de estos se indican en la Tabla 3.3. El listado de los estudiantes de la EPN que realizaron la adquisición de datos se detalla en la Tabla 3.2.

Tabla 3.3 Información de los dispositivos móviles utilizados en el proyecto

Nombre dispositivo	Modelo	Versión Android	Nivel API
Samsung Galaxy J1 ace	SM-J111M	Lollipop 5.1.1	22
Samsung Galaxy J7	SM-J700M	Marshmallow 6.0.1	23
Xiaomi	Redmi Note 4	Nougat 7.0	24
Huawei P10 lite	WAS-LX3	Nougat 7.0	24
Samsung Galaxy J5	SM-J510MN	Nougat 7.1.1	25
Samsung Galaxy S7 Edge	SM-G935FD	Oreo 8.0	26
Samsung Galaxy S8+	SM-G955FD	Oreo 8.0	26

La ubicación de la ciudad de Quito y de la comunidad San Francisco de Chibuleo, se muestran dentro del territorio continental de Ecuador en la Figura 3.9.

Figura 3.9 Territorio continental de la república del Ecuador

El camino por carretera para llegar a la comunidad se indica en la Figura 3.10, el traslado duró aproximadamente 3 horas en vehículo particular, partiendo desde Quito y recorriendo un aproximado de 171 km en el trayecto de ida.

Figura 3.10 Trayecto ciudad de Quito - Comunidad San Francisco de Chibuleo.

En la Figura 3.11 se observa la ubicación de la comunidad San Francisco con referencia a la ciudad de Ambato[19], la distancia por carretera entre estas localidades es de 15,9 km aproximadamente.

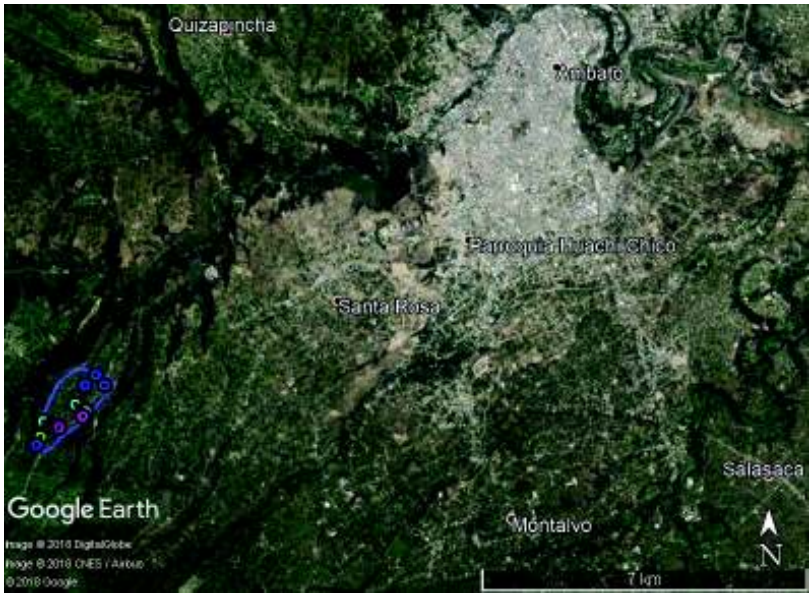


Figura 3.11 Ubicación de San Francisco de Chibuleo con relación a la ciudad de Ambato.

[19] Capital de la provincia de Tungurahua

3.2.1 Registro de palabras quichua

El día de puesta en marcha del proyecto, se arribó al centro de la comunidad indígena San Francisco de Chibuleo a las 10h00, se realizaron grabaciones de palabras quichuas desde los diferentes dispositivos móviles, desde las 10h40 hasta las 13h37.

En la Figura 3.12 se observa el instante en el que se realiza la toma de un registro oral de la palabra volcán, en quichua *urku*.

Figura 3.12 Registro de palabras quichua *urku*.

Se realizaron registros a varios hablantes dentro de la comunidad. En la Figura 3.13 se indica la grabación de la palabra quichua *ñan*, en español camino.

Figura 3.13 Registro de la palabra quichua *ñan*.

Por la extensión de la comunidad indígena, los estudiantes fueron trasladados a diferentes puntos dentro de la comunidad con la ayuda de dos automóviles particulares, y desde ahí se movilizaron a pie para realizar la toma de registros. En la Figura 3.14 se observa el traslado de los estudiantes a nuevos puntos para realizar nuevos registros.

Figura 3.14 Movilización de estudiantes para adquirir nuevos registros.

En algunos sectores de la comunidad se pudieron realizar la toma de varios registros a diferentes personas, en la Figura 3.15 se observa la adquisición de registros a cuatro hablantes del idioma quichua.

Figura 3.15 Toma de registros a varios quichua hablantes.

Por ser fin de semana, el día en que se ejecutó el proyecto muchos hablantes se encontraban realizando actividades relacionas al trabajo comunitario en el campo, en la Figura 3.16 se muestra la toma de registros bajo esas condiciones.

Figura 3.16 Registro de palabras del idioma quichua.

En total se realizó la grabación de 365 registros orales entre 30 miembros de la comunidad San Francisco de Chibuleo. En la Figura 3.17 y Figura 3.18 se observa la toma de registros de palabras quichuas a hablantes nativos dentro de la comunidad.

Figura 3.17 Registros individuales de palabras en quichua.

Figura 3.18 Registros individuales de palabras en quichua.

3.2.2 Verificación de funcionamiento del servicio GPS

Para comprobar el correcto funcionamiento del servicio GPS dentro de la aplicación móvil, se grafican las coordenadas de localización geográfica de cada registro en el programa Google Earth Pro.

En la Figura 3.19 se observa parte del área de la comunidad, la superficie cerrada no representa los límites de la comunidad San Francisco, sino la zona de realización del proyecto. Según datos proporcionados por Google Earth Pro, el área de la zona en la que se desarrolló el proyecto fue de 152 hectáreas.

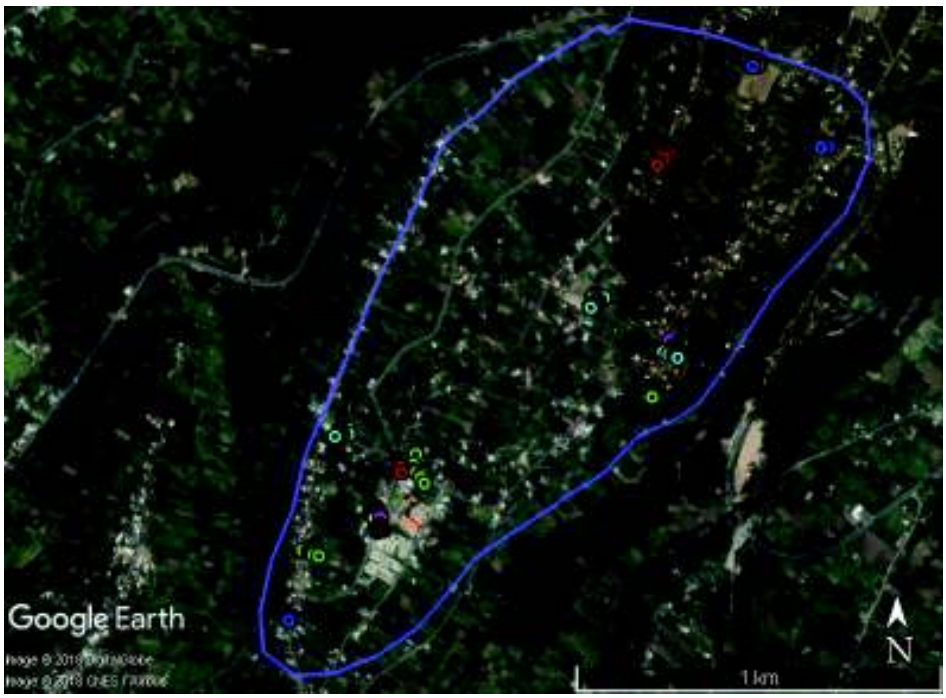


Figura 3.19 Comunidad San francisco de Chibuleo

Las ubicaciones geográficas de la toma de registros se muestran con diferente color, se utilizó el mismo código de la Tabla 3.2, identificando al estudiante que realizó la adquisición de grabaciones de audio.

Para mejor visualización se realizan tomas más cercanas de la comunidad, dividiéndola en tres sectores, estas divisiones tienen propósitos de visualización para el presente proyecto. En la Figura 3.20 se presenta la sección norte de la comunidad.

Figura 3.20 Comunidad San Francisco, sector de visualización norte.

En la Figura 3.21 se observa la sección centro de la comunidad San francisco.

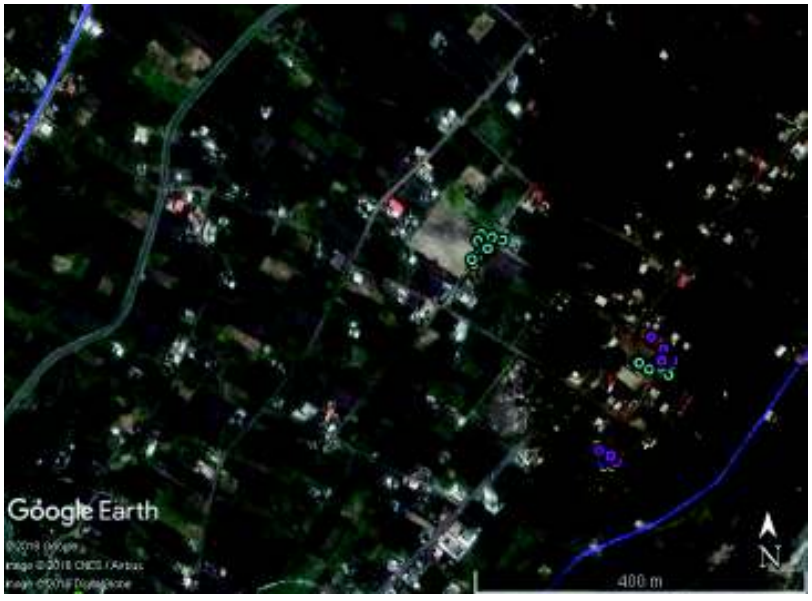


Figura 3.21 Comunidad San Francisco, sector de visualización centro.

Los metadatos de cada registro fueron importados al programa Google Earth Pro, a través de archivos de formato .csv[20], este tipo de documento se obtiene al exportar el archivo de visualización de datos tipo Excel. En la Figura 3.22 se ve un ejemplo de visualización de estos metadatos.

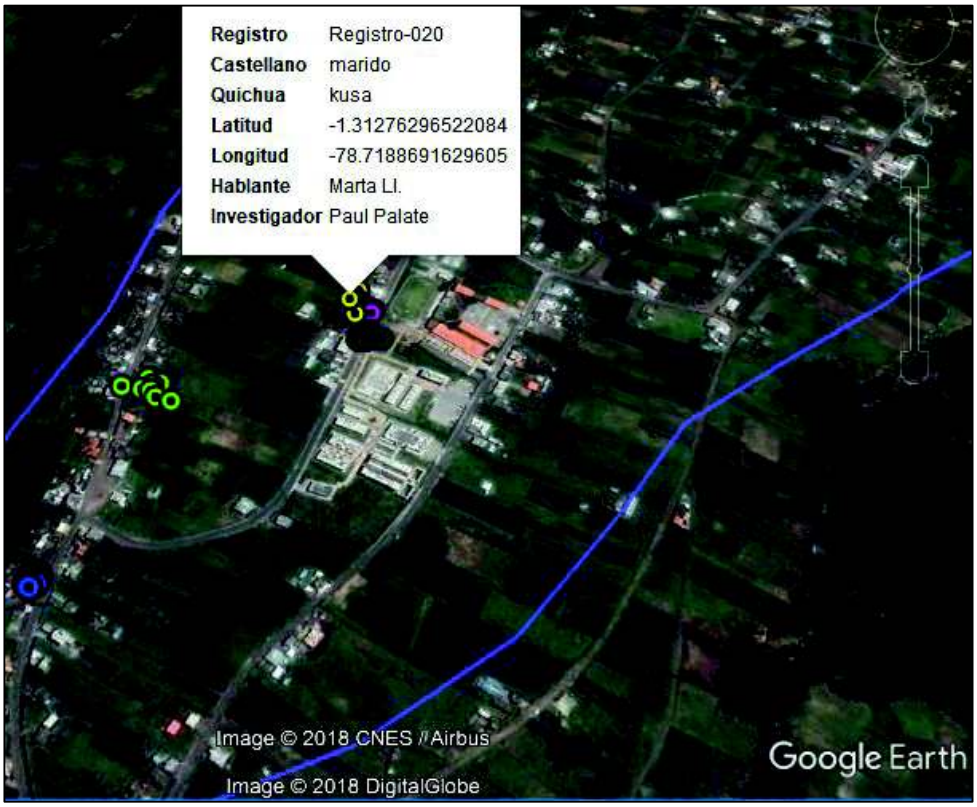


Figura 3.22 San Francisco, sector de visualización sur visto desde Google Earth Pro.

Los datos que se muestran son un extracto de los metadatos del registro, para una visualización de toda la información se recomienda usar: el archivo de visualización Excel, Microsoft Azure Storage Explorer o desde cualquier navegador web; ingresando al portal de Azure, previa autenticación.

3.2.3 Verificación de almacenamiento en la nube

En esta sección se realiza un análisis de métricas del almacenamiento en la nube, se verifica que todos los archivos de audio hayan sido guardados en la nube, también se comprueba que las entidades hayan sido insertadas correctamente en la base de datos de tipo no relacional.

[20].csv. - (*comma separated values*, valores separados por coma), formato de archivo abierto para representar datos en forma de tabla.

3.2.3.1 Subida de archivos y entidades a la cuenta de almacenamiento

El registro-001 fue insertado en la nube a las 11h35, mientras que el registro-356 a las 20:03. En la Figura 3.23 se detalla la cantidad de datos que fueron ingresados en este intervalo de tiempo a la cuenta de almacenamiento Azure, la cantidad de datos están medidos en KiB[21] y MiB[22].

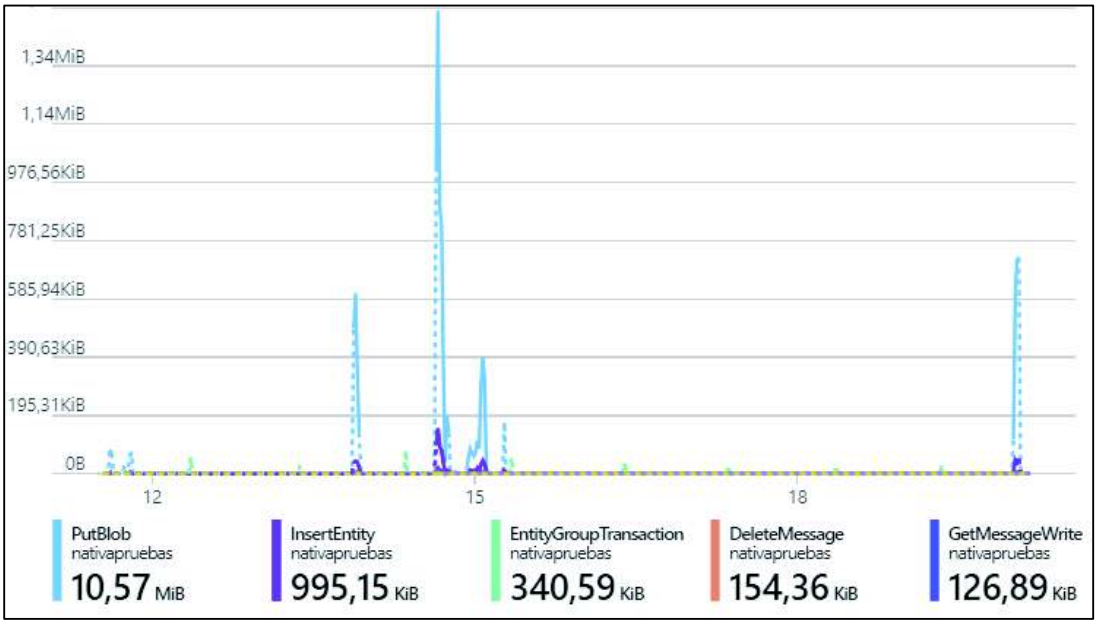


Figura 3.23 Cantidad de datos totales ingresados a la cuenta de almacenamiento

La mayoría de los registros fueron subidos a la nube desde la 01h50 hasta las 15h15, coincidiendo con el almuerzo que tuvieron los participantes del proyecto después de la visita a la comunidad San Francisco. Los datos ingresados en este intervalo se detallan en la Figura 3.24.

Los registros del dispositivo Samsung Galaxy J1 Ace fueron subidos a la nube en horas de la noche, cuando se regresó a Quito. El dispositivo mencionado no contaba con paquete de datos móviles, razón por la cual, fue necesario subir los registros mediante conexión wifi.

[21] Kibibyte (KiB) = 2^{10} bytes = 1024 bytes
[22] Mebibyte (MiB) = 2^{20} bytes = 1048576 bytes

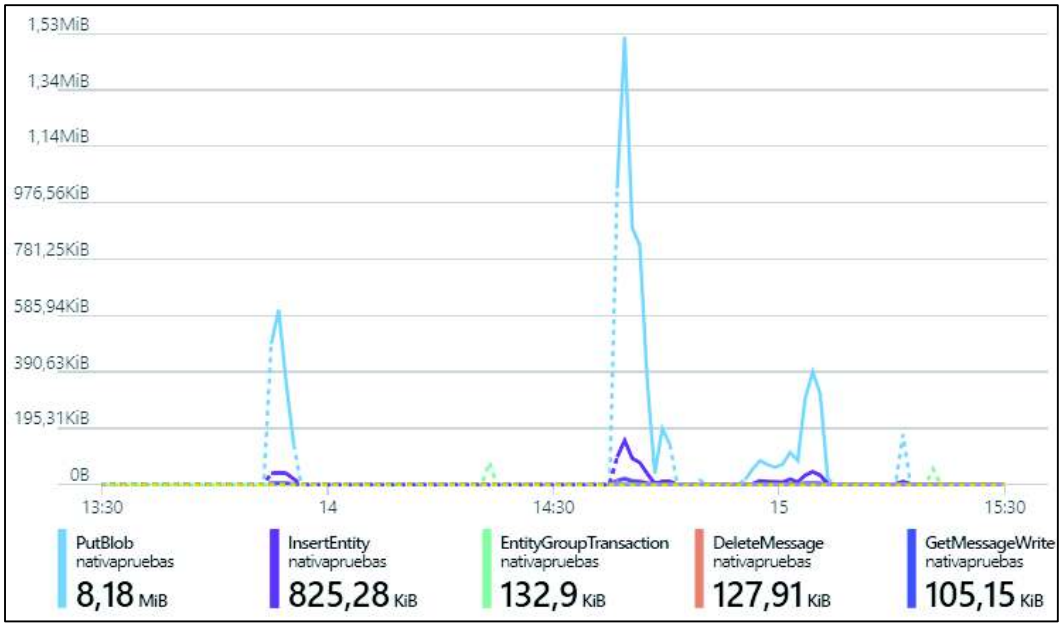


Figura 3.24 Datos ingresados a la nube en el intervalo del almuerzo.

El tiempo promedio que le tomó al servidor de la cuenta de almacenamiento Azure procesar todas las transacciones fue de 19,29 segundos, intervalo medido del lado del servidor. Por registro se tiene un tiempo promedio de latencia de 54,19 mS, que incluye el almacenamiento del archivo de audio y la inserción de los datos en las 4 tablas NoSQL. Los detalles de este tiempo de latencia a lo largo del intervalo en que se subieron todos los registros se ve en la Figura 3.25.

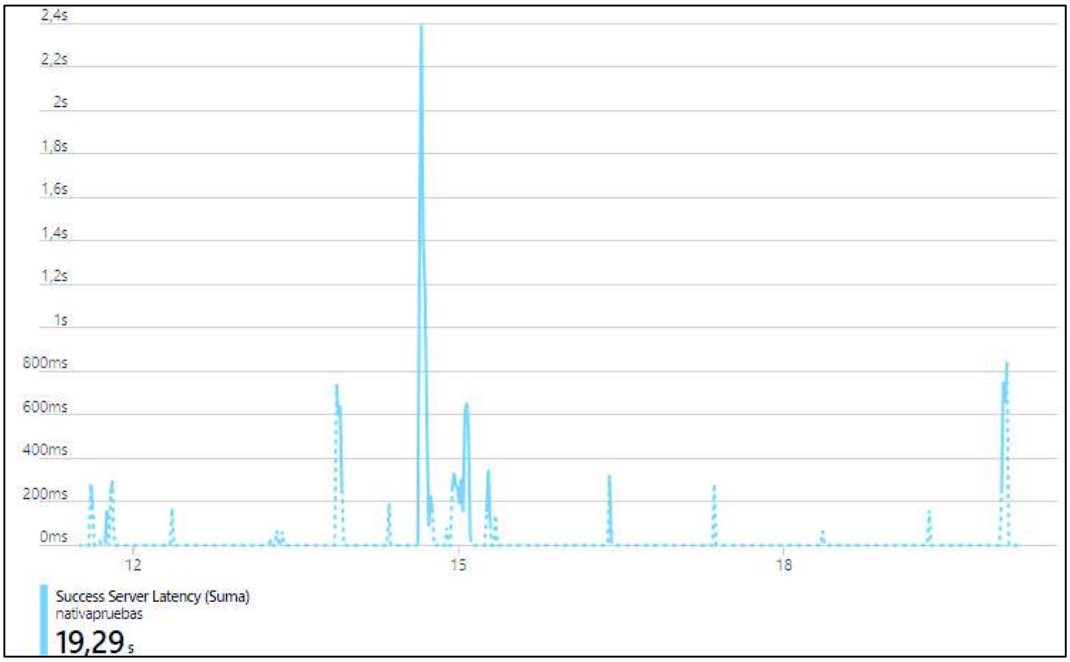


Figura 3.25 Latencia de todas las transacciones exitosas del lado del servidor.

La latencia promedio de extremo a extremo de todas las transacciones exitosas fue de 17,55 minutos, se mide desde que el cliente de la aplicación Android realiza la solicitud de envío de registro hasta recibir la confirmación que el proceso fue realizado con éxito. Cada registro tuvo una latencia promedio de extremo a extremo de 2,96 segundos. En la Figura 3.26 se detallan los valores de esta latencia a lo largo del tiempo.

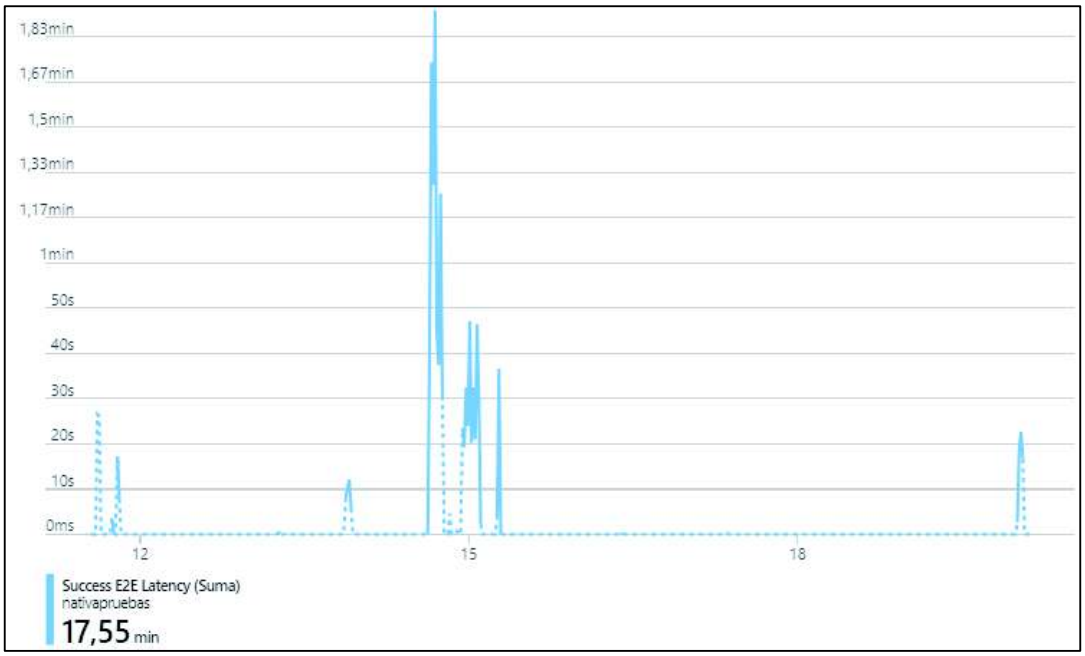


Figura 3.26 Latencia total de extremo a extremo de las transacciones exitosas.

3.2.3.2 Almacenamiento de archivos y entidades en la nube

Los archivos de audio fueron guardados en el contenedor tipo BLOB audios y se presentan en la Figura 3.27. En la propiedad última modificación (*Last Modified*), se guarda la fecha y hora en la que se subieron los archivos a la nube o la fecha de la última modificación del archivo, de ser el caso.

El programa de escritorio Microsoft Azure Storage Explorer permite el proceso de visualización y edición de todos los recursos de almacenamiento en la nube. En la parte izquierda de la Figura 3.27, se observa la estructura de una cuenta de almacenamiento Azure.

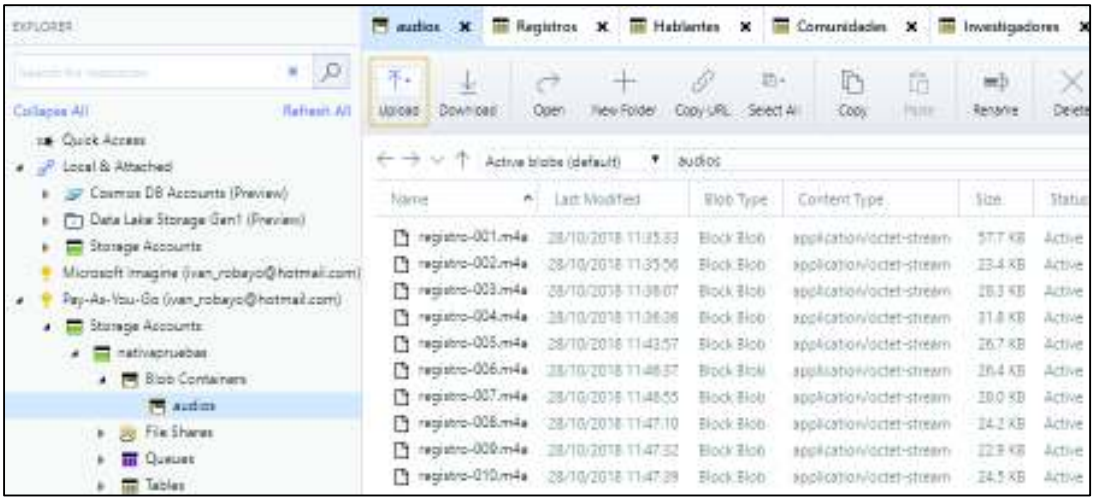


Figura 3.27 Archivos de audio almacenados en el contenedor audios.

En la Figura 3.28 y Figura 3.29 se muestran las propiedades de las primeras entidades en la tabla Registros, las propiedades fecha y hora hacen referencia al instante en que se realizó la grabación de audio en el dispositivo móvil.

PartitionKey	RowKey	Timestamp	TipoRegistro	Castellano	Nativo	Duracion	Fecha
Registro-001	registro	2018-10-28T16:35:35.174Z	Palabra	A gusto	Ninantak	00:04.64	2018-10-28
Registro-002	registro	2018-10-28T16:35:56.763Z	Palabra	Abrazar	Ukllana	00:01.71	2018-10-28
Registro-003	registro	2018-10-28T16:36:08.404Z	Palabra	Aceite	Wira	00:02.13	2018-10-28
Registro-004	registro	2018-10-28T16:36:37.796Z	Palabra	Aconsejar	Kunana	00:02.43	2018-10-28
Registro-005	registro	2018-10-28T16:44:00.446Z	Palabra	Acordarse	Yuyarina	00:01.99	2018-10-28
Registro-006	registro	2018-10-28T16:46:40.883Z	Palabra	Además	Ashtawan	00:01.97	2018-10-28
Registro-007	registro	2018-10-28T16:46:55.706Z	Palabra	Acostarse	Siririna	00:02.11	2018-10-28
Registro-008	registro	2018-10-28T16:47:12.641Z	Palabra	Actual	Kunan	00:01.78	2018-10-28
Registro-009	registro	2018-10-28T16:47:33.137Z	Palabra	Adivinar	Watuna	00:01.67	2018-10-28
Registro-010	registro	2018-10-28T16:47:40.107Z	Palabra	Abismo	Kaka	00:01.81	2018-10-28

Figura 3.28 Inserción de entidades en la tabla NoSQL Registros parte 1.

Hora	Latitud	Longitud	Link
11:05:27	-1,30889902253707	-78,7113997971457	https://nativapruebas.blob.core.windows.net/audios/registro-001.m4a
11:07:10	-1,3088287182971	-78,71175390743	https://nativapruebas.blob.core.windows.net/audios/registro-002.m4a
11:07:29	-1,30897590159132	-78,7114013727344	https://nativapruebas.blob.core.windows.net/audios/registro-003.m4a
11:07:50	-1,30894493880868	-78,7114138959751	https://nativapruebas.blob.core.windows.net/audios/registro-004.m4a
11:08:13	-1,30895453783068	-78,7113816606279	https://nativapruebas.blob.core.windows.net/audios/registro-005.m4a
11:08:39	-1,30886409450215	-78,7114940269025	https://nativapruebas.blob.core.windows.net/audios/registro-006.m4a
11:09:05	-1,30888713878774	-78,7116212951757	https://nativapruebas.blob.core.windows.net/audios/registro-007.m4a
11:09:23	-1,30881683567581	-78,7117310257898	https://nativapruebas.blob.core.windows.net/audios/registro-008.m4a
11:09:50	-1,30891432659919	-78,7114814243483	https://nativapruebas.blob.core.windows.net/audios/registro-009.m4a
11:10:11	-1,30891376234924	-78,711480286713	https://nativapruebas.blob.core.windows.net/audios/registro-010.m4a

Figura 3.29 Inserción de entidades en la tabla NoSQL Registros parte 2.

El contenido de las entidades dentro de las tablas NoSQL puede ser observado a través de Microsoft Azure Storage Explorer o el archivo de visualización tipo Excel, pero no desde el portal de Azure, allí se pueden observar las tablas, pero no su contenido.

En la Figura 3.30, se muestran las últimas entidades de la tabla Hablantes obtenidas desde el archivo ofimático tipo Excel.

PartitionKey	RowKey	Timestamp	Nombre	Edad	Genero
Registro-347	hablante	29/10/2018 1:02	Patricio U	33	Masculino
Registro-348	hablante	29/10/2018 1:02	Patricio U	33	Masculino
Registro-349	hablante	29/10/2018 1:02	Patricio U	33	Masculino
Registro-350	hablante	29/10/2018 1:02	Patricio U	33	Masculino
Registro-351	hablante	29/10/2018 1:02	Patricio U	33	Masculino
Registro-352	hablante	29/10/2018 1:02	Patricio U	33	Masculino
Registro-353	hablante	29/10/2018 1:02	Mariano S	43	Masculino
Registro-354	hablante	29/10/2018 1:02	Mariano S	43	Masculino
Registro-355	hablante	29/10/2018 1:02	Mariano S	43	Masculino
Registro-356	hablante	29/10/2018 1:02	Mariano S	43	Masculino

Figura 3.30 Entidades de la tabla NoSQL Hablantes.

En la Figura 3.31 y Figura 3.32 se observan entidades intermedias de las tablas Comunidades e Investigadores respectivamente, fueron conseguidas desde el archivo de visualización Excel.

PartitionKey	RowKey	Timestamp	Comunidad	Pueblo	Pais	Region	Provincia	Latitud	Longitud
Registro-272	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30919717	-78.71161262
Registro-273	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30896392	-78.71160131
Registro-274	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30896392	-78.71160131
Registro-275	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30896392	-78.71160131
Registro-276	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30919717	-78.71161262
Registro-277	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30896392	-78.71160131
Registro-278	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30919717	-78.71161262
Registro-279	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30896392	-78.71160131
Registro-280	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30896392	-78.71160131
Registro-281	comunidad	28/10/2018 20:03	San Francisco	Chibuleo	Ecuador	Sierra	Tungurahua	-1.30919717	-78.71161262

Figura 3.31 Entidades de la tabla NoSQL Comunidades.

PartitionKey	RowKey	Timestamp	Nombre	Identificacion	Genero
Registro-069	investigador	28/10/2018 19:37	Roberto Guerrero	Recolector	Masculino
Registro-070	investigador	28/10/2018 19:37	Diego Alvarado	Recolector	Masculino
Registro-071	investigador	28/10/2018 19:37	Kevin Cajas	Recolector	Masculino
Registro-072	investigador	28/10/2018 19:37	Diego Alvarado	Recolector	Masculino
Registro-073	investigador	28/10/2018 19:37	Roberto Guerrero	Recolector	Masculino
Registro-074	investigador	28/10/2018 19:37	Kevin Cajas	Recolector	Masculino
Registro-075	investigador	28/10/2018 19:37	Diego Alvarado	Recolector	Masculino
Registro-076	investigador	28/10/2018 19:37	Kevin Cajas	Recolector	Masculino
Registro-077	investigador	28/10/2018 19:37	Diego Alvarado	Recolector	Masculino
Registro-078	investigador	28/10/2018 19:37	Roberto Guerrero	Recolector	Masculino

Figura 3.32 Entidades de la tabla NoSQL Investigadores.

3.3 Encuesta de validación

Luego de finalizar el registro de palabras quichuas, se realizó una encuesta a los estudiantes que participaron en el proyecto, con el fin de medir la interacción que tuvieron con la aplicación móvil. La encuesta realizada tiene 13 preguntas divididas en tres secciones. El contenido de la encuesta se encuentra en el ANEXOS IX.

3.3.1 Con respecto al uso de la aplicación móvil

La primera pregunta tiene como objetivo calificar el aspecto visual de la aplicación móvil Android. Los resultados de la pregunta se detallan en la Figura 3.33, el 63% de los encuestados afirman que el aspecto visual de la aplicación es bueno.

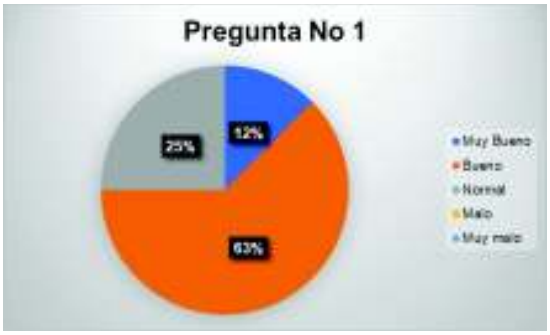


Figura 3.33 Resultados pregunta 1. ¿Cómo calificaría el aspecto visual de la aplicación?

La segunda pregunta determina la percepción que tiene el usuario sobre el tiempo de carga al iniciar la aplicación. El 62% de los usuarios considera que la aplicación tiene un inicio muy rápido. Los resultados se detallan en la Figura 3.34.

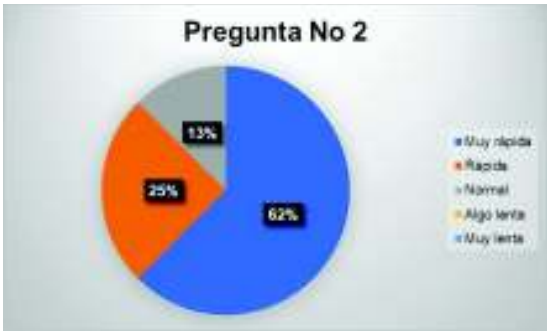


Figura 3.34 Resultados pregunta 2. ¿Cómo le pareció la velocidad de carga al iniciar la aplicación?

La tercera pregunta evalúa el nivel de dificultad al navegar por la aplicación. En la Figura 3.35 se observa que el 75% de los encuestados afirma que le resultó muy fácil navegar por la aplicación móvil.

Figura 3.35 Resultados pregunta 3. ¿Cómo le resultó navegar por la aplicación?

La cuarta pregunta pretende evaluar que tan difícil le fue al usuario el ingreso de metadatos de hablante, investigador y comunidad. Según la Figura 3.36 el 50% considera que el ingreso de matados fue muy fácil y el 37% relativamente fácil.

Figura 3.36 Resultados pregunta 4. ¿Cuál fue el nivel de dificultad para el ingreso de metadatos?

La quinta pregunta permite medir la dificultad presentada al momento de realizar la grabación de nuevos registros. El 38% de los usuarios afirma que el proceso fue relativamente fácil, mientras que el 37% considera que fue muy fácil. Los detalles de los resultados se encuentran en la Figura 3.37.

Figura 3.37 Resultados pregunta 5. ¿Cuál fue el nivel de dificultad para realizar nuevos registros de audio?

La sexta pregunta permite verificar la percepción que tuvo el usuario con la velocidad de subida de registros a la nube. Según la Figura 3.38 el 38% de encuestados considera que la subida de archivos fue rápida y el 37% afirma que la velocidad de carga de archivos fue muy rápida.

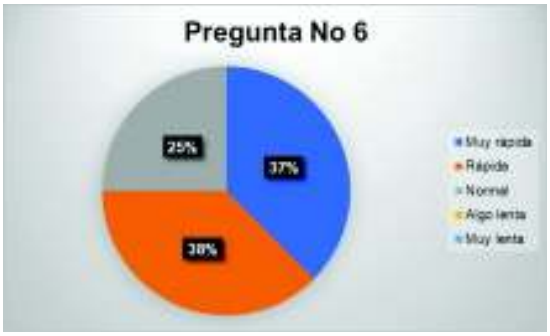


Figura 3.38 Resultados pregunta 6. ¿Cuál fue la velocidad de subida de registros a la nube?

3.3.2 Con respecto a la capacitación previa al proyecto

La pregunta 7 pretende evaluar la estimación del usuario acerca de la inducción recibida, el 50% de participantes considera al nivel de capacitación como muy bueno, mientras que el 37% lo determina como bueno.

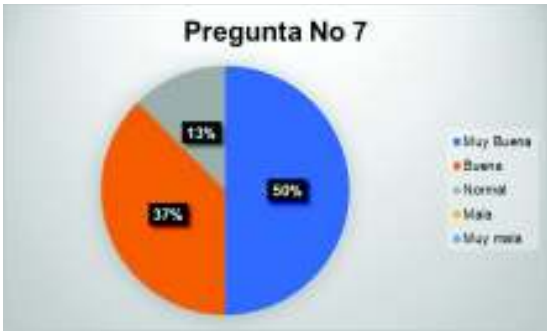


Figura 3.39 Resultados pregunta 7. ¿Cómo calificaría la inducción previa a la realización del proyecto?

La octava pregunta pretende evaluar la apreciación del tiempo que duró la capacitación. Según la Figura 3.40 el 87% de usuarios considera que el tiempo de capacitación fue normal y un 13% afirma que el tiempo fue corto.

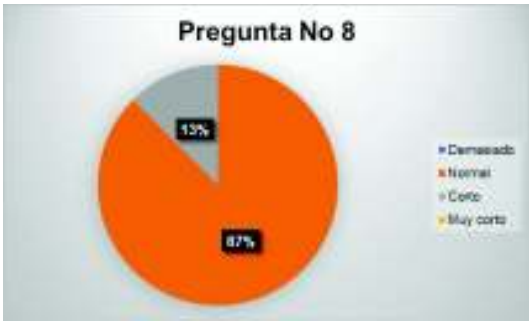


Figura 3.40 Resultados pregunta 8. ¿Cuál fue su apreciación del tiempo que duró la capacitación?

La novena pregunta permite cualificar el nivel de claridad de las instrucciones proporcionadas por el capacitador. El 75% de encuestados considera que las instrucciones fueron muy claras, mientras que 12% afirma que las instrucciones fueron relativamente claras. Los detalles de los resultados se encuentran en la Figura 3.41.

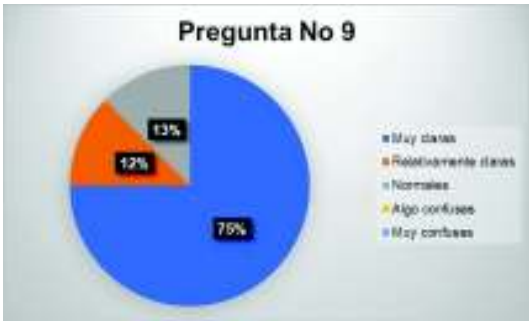


Figura 3.41 Resultados pregunta 9. ¿Cómo fueron las indicaciones del instructor?

3.3.3 Con respecto a la realización del proyecto

La décima pregunta permite validar la percepción que tuvo el usuario sobre la participación de los miembros de la comunidad indígena. Según la Figura 3.42 el 38% de los encuestados considera que los miembros de la comunidad tuvieron una participación relativamente activa, mientras que el 50% estima que la participación fue normal.

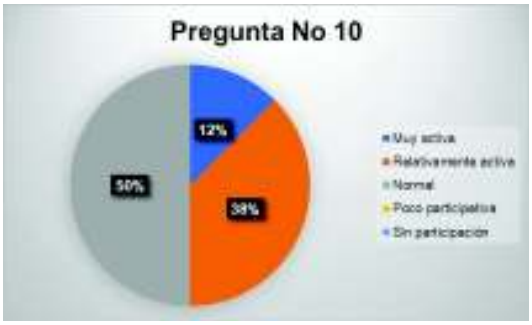


Figura 3.42 Resultados pregunta 10. ¿Cómo fue la participación de los miembros de la comunidad?

La pregunta once permite evaluar la estimación sobre el interés de la comunidad en la participación del proyecto. El 37% considera que hubo un interés de participación muy bueno, mientras que el 38% considera un nivel de interés bueno, como se ve en la Figura 3.43.

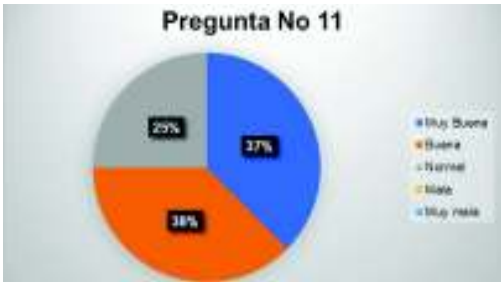


Figura 3.43 Resultados pregunta 11. ¿Cómo fue el interés de la comunidad en la participación del proyecto?

La pregunta doce permite medir la apreciación del usuario, sobre si la realización de proyectos como este permite difundir la riqueza lingüística del quichua. El 50% de los encuestados está totalmente de acuerdo con esta afirmación y el 37% está de acuerdo.

Figura 3.44 Resultados pregunta 12. ¿Podrían este tipo de proyectos, ayudar a difundir la riqueza lingüística y cultural de las comunidades indígenas?

La pregunta trece permite medir la disponibilidad de los estudiantes para una futura realización de proyectos similares. En la Figura 3.45 se observa que el 87% de los participantes estarían dispuestos a participar en nuevos proyectos de las mismas características.

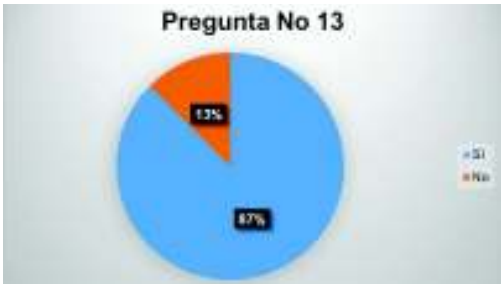


Figura 3.45 Resultados pregunta 13. ¿Participaría nuevamente en proyectos similares?

4. CONCLUSIONES Y RECOMENDACIONES

Al finalizar el presente Proyecto Técnico se obtuvieron las siguientes conclusiones y recomendaciones:

4.1 Conclusiones

- Se logró implementar por completo un prototipo multimedia en base a una aplicación móvil Android, la cual permite el registro y almacenamiento de lenguas en peligro de extinción. Todos los datos y metadatos recolectados fueron almacenados exitosamente en la nube, se preservarán a lo largo del tiempo para su posterior estudio, análisis o utilización.

- La aplicación móvil permitió recolectar la pronunciación, escritura y traducción de 356 palabras pertenecientes al idioma quichua. También se logró guardar metadatos del hablante nativo, comunidad a la que pertenece y de la persona que realizó la adquisición de datos. Toda esta información es pertinente al proceso de registro de lenguas nativas en peligro de extinción.

- La base de datos de tipo no relacional, alojada en la nube, permitió recibir y guardar todos los metadatos de forma ágil y segura; con una tasa de éxito del 100%, almacenando un total de 1424 entidades, distribuidas en las cuatro tablas NoSQL de la cuenta de almacenamiento perteneciente al presenta proyecto.

- El servicio de mensajes de colas gestionó todas las peticiones de los clientes Android, otorgando identificadores únicos a cada registro. Gran parte de los registros fueron subidos a la nube al mismo tiempo. A pesar de eso, los diferentes dispositivos móviles que participaron en el proyecto pudieron subir toda la información sin ningún problema.

- La aplicación móvil tuvo un alto desempeño, permitió la captura de nuevos registros, reproducción de grabaciones previas, acceder a datos proporcionados por el GPS, enviar información a la nube e introducir nuevos metadatos. Todas estas acciones realizadas de manera simultánea. Sin disminuir la fluidez en la experiencia de usuario; debido a la programación de métodos asincrónicos, uso de varios hilos de programación y la implementación de servicios en segundo plano.

- La utilización de tecnologías de la información para la conservación de lenguas nativas como el quichua en la nube, permite divulgar los conocimientos ancestrales propios de este tipo de lenguajes y apoyar a procesos tanto de enseñanza como re-enseñanza del idioma.

- Existe una amplia gama de servicios en la nube, para cada necesidad se puede encontrar una solución en cualquiera de los principales proveedores de nube. No existe una mejor o peor selección del tipo de servicio a utilizar, estos dependen de los requerimientos del sistema y de la arquitectura de desarrollo a implementarse.
- Con la existencia de una base de datos que contenga el idioma quichua se puede empezar a desarrollar múltiples herramientas tecnológicas. A partir del almacenamiento de la lengua en la nube, ya que facilita interoperabilidad con diferentes tipos de tecnologías de desarrollo.
- Utilizar C# como lenguaje de programación permitirá una futura extensión de la aplicación móvil, enfocándola al desarrollo multiplataforma. Permitiendo trabajar con dispositivos móviles iOS; realizando los respectivos ajustes al sistema y reutilizando gran parte del código usado en este proyecto.
- En la puesta en marcha del proyecto, se evidenció que dentro de las comunidades indígenas todavía no se habla un quichua unificado en su totalidad, existen ligeras variaciones en la traducción de ciertas palabras quichuas en comparación a las encontradas en la referencia bibliográfica [5].
- El desarrollo de proyectos tecnológicos de este tipo permite gestionar la creación de futuros proyectos que estén encaminados a satisfacer necesidades específicas de las comunidades indígenas, creando canales de comunicación directos entre representantes de la academia y miembros de la comunidad.
- El proyecto desarrollado permitió a los estudiantes de la EPN conocer una parte de la riqueza cultural de las comunidades indígenas, involucrarse en la realidad que viven las poblaciones indígenas del Ecuador y fortalecer uno de los pilares de la educación superior, servir al bienestar de la sociedad.

4.2 Recomendaciones

- Es indispensable que la batería de los dispositivos móviles se encuentre en buen estado, la aplicación en sí tiene un consumo de energía normal, se realiza esta recomendación porque el proceso de registro de lenguas nativas puede llegar a extenderse.
- Se recomienda mejorar la experiencia de usuario, proporcionando una mejora en el diseño de las pantallas de la aplicación móvil, para estos fines sería ideal incluir un diseñador gráfico en el equipo de trabajo.

- En futuras extensiones del presente proyecto se sugiere agregar un editor básico de audio, dentro de la misma aplicación. Así, el recolector de datos podrá cortar partes de la grabación que considere impertinentes para los fines de la investigación.
- Dar continuidad a este proyecto, agregando nuevas funcionalidades como: adquisición de video, crear un web API que permita gestionar la creación, administración y autenticación de nuevos usuarios.
- Se recomienda utilizar la palabra “quichua” para referirse al idioma que habla la nacionalidad Kichwa, siempre y cuando el texto del documento esté escrito en castellano, ya que ésta consta en el diccionario de la RAE. Si el idioma de escritura fuera el quichua, lo correcto sería utilizar la palabra en nativo *kichwa* para hacer referencia al idioma.
- Para fines de análisis migratorio se recomienda añadir un campo en los metadatos del hablante que permita identificar si pertenece a la comunidad en la que se está realizando la adquisición de datos. Dando la posibilidad de añadir la comunidad de origen del hablante nativo.
- Para mejorar la calidad de las grabaciones, se recomienda utilizar un micrófono externo con mayores prestaciones técnicas al hardware del dispositivo móvil. También se puede adecuar una estación fija con características técnicas que permitan aislar el ruido ambiental. Esta recomendación es aplicable cuando el asentamiento geográfico de las comunidades es poco disperso.
- El prototipo desarrollado cumple y sobrepasa todas las expectativas técnicas, sin embargo, surge la necesidad de vincular al proyecto a profesionales de otras áreas inherentes a la investigación lingüística, de ser el caso y si existe el interés por parte del lector de participar en futuros registros de lenguas nativas, se sugiere usar los datos de contacto proporcionados al inicio del presente documento.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] C. Moseley, Atlas de las Lenguas del Mundo en Peligro, Segunda ed., Paris: Ediciones UNESCO, 2010.

[2] J. Good, J. Hirschberg y O. Rambow, «Workshop on the Use of Computational Methods in the Study of Endangered Languages,» 2014. [En línea]. Available: https://goo.gl/k41rPr. [Último acceso: 23 julio 2018].

[3] J. Gómez, «Patrimonio lingüístico, revitalización y documentación de lenguas amenazadas,» *Revista Nacional de Cultura del Ecuador,* vol. XIII, pp. 10-21, 2008.

[4] K. Klessa, «Language Documentation,» de *Book of Knowledge of Languages in Danger*, M. Hornsby, Ed., Poznań, Adam Mickiewicz University, 2014, pp. 106-107.

[5] Kichwa Yachakukkunapa Shimiyuk Kamu, Quito: Ministerio de Educación, 2009.

[6] M. Haboud, «Quichua y castellano en los Andes ecuatorianos : los efectos de un contacto prolongado,» Abya-Yala, Quito, 1998.

[7] G. López, «http://www.ecuadorencifras.gob.ec,» [En línea]. Available: https://goo.gl/9mQGKC. [Último acceso: 11 noviembre 2018].

[8] CONAIE, «Nacionalidades - Chibuleo,» 19 julio 2014. [En línea]. Available: https://goo.gl/KLoD7w. [Último acceso: 13 septiembre 2018].

[9] C. Olmedo, Interviewee, *Datos de la comunidad San Francisco.* [Entrevista]. 13 noviembre 2018.

[10] J. J. Espín, Interviewee, *Punto de vista a cerca de la lengua nativa quichua.* [Entrevista]. 28 octubre 2018.

[11] A. Silberschatz, H. Korth y S. Sudarshan, Fundamentos de bases de datos, Quinta ed., Madrid: McGraw-Hill, 2006, pp. 1-10.

[12] «Gestión de Bases de Datos,» IES Luis Vélez de Guevara, 3 julio 2018. [En línea]. Available: https://goo.gl/6bfRNw. [Último acceso: 26 julio 2018].

[13] Whitepaper Bases de datos NoSQL, Acens Technologies SL., Madrid, 2014.

[14] Microsoft, «¿Qué es la informática en la nube?,» Microsoft Azure, [En línea]. Available: https://goo.gl/zNBc61. [Último acceso: 11 agosto 2018].

[15] M. Moreno, Computación en la Nube, Buenos Aires: UCEMA, 2015.

[16] L. J. Aguilar, «Cloud Computing,» *Journal of the Higher School of National Defense Studies,* vol. 0, pp. 83-103, 2012.

[17] Amazon Web Services, «Acerca de AWS,» [En línea]. Available: https://goo.gl/A2LWpa. [Último acceso: 11 agosto 2018].

[18] E. Morse, « Google Cloud Official Blog,» julio 2012. [En línea]. Available: https://goo.gl/rnZZAL. [Último acceso: 11 agosto 2018].

[19] Microsoft Azure, «Infraestructura global de Azure,» [En línea]. Available: https://goo.gl/ikWicP. [Último acceso: 11 agosto 2018].

[20] Android Open Source Project, «About the platform,» 25 abril 2018. [En línea]. Available: https://goo.gl/jTb6b3. [Último acceso: 13 agosto 2018].

[21] Android Open Source Project, «Java Native Interface Tips,» 15 junio 2018. [En línea]. Available: https://goo.gl/5GgeC9. [Último acceso: 14 agosto 2018].

[22] Android Open Source Project, «Buttons,» 23 abril 2018. [En línea]. Available: https://goo.gl/NvBTsm. [Último acceso: 29 agosto 2018].

[23] M. McLemore, C. Dunn y B. Umbaugh, «Microsoft Docs,» 06 junio 2018. [En línea]. Available: https://goo.gl/RwyhTc. [Último acceso: 30 agosto 2018].

[24] Android Open Source Project, «Listview,» 06 junio 2018. [En línea]. Available: https://goo.gl/nrsge6. [Último acceso: 30 agosto 2018].

[25] M. McLemore y O. , «Microsoft Docs,» 24 abril 2018. [En línea]. Available: https://goo.gl/bKEqcA. [Último acceso: 30 agosto 2018].

[26] Android Open Source Project, «Dialogs,» 25 abril 2018. [En línea]. Available: https://goo.gl/iKWj4Z. [Último acceso: 30 agosto 2018].

[27] Android Open Source Project, «Spinners,» 25 abril 2018. [En línea]. Available: https://goo.gl/WLshnQ. [Último acceso: 31 agosto 2018].

[28] M. McLemore y O. , «Microsoft Docs - Spinner,» 05 febrero 2018. [En línea]. Available: https://goo.gl/Ms64kD. [Último acceso: 31 agosto 2018].

[29] Android Open Source Project, «Documentation Guides - Toasts overview,» 23 mayo 2018. [En línea]. Available: https://goo.gl/oxKNKj. [Último acceso: 31 agosto 2018].

[30] Android Open Source Project, «Documentation Guides - ProgressBar,» 06 junio 2018. [En línea]. Available: https://goo.gl/88HFL7. [Último acceso: 31 agosto 2018].

[31] Microsoft, «Microsoft Docs - ProgressBar Class,» [En línea]. Available: https://goo.gl/ouhDdo. [Último acceso: 31 agosto 2018].

[32] Android Open Source Project, «Documentation Reference - SeekBar,» 06 junio 2018. [En línea]. Available: https://goo.gl/PgUpKz. [Último acceso: 31 agosto 2018].

[33] M. McLemore, T. Opgenorth, B. Umbaught y C. Dunn, «Microsoft Docs - Local notifications,» 15 agosto 2018. [En línea]. Available: https://goo.gl/e5JbDt. [Último acceso: 31 agosto 2018].

[34] Android Open Source Project, «Documentation Guides - Layouts,» 25 abril 2018. [En línea]. Available: https://goo.gl/f1tP3B. [Último acceso: 31 agosto 2018].

[35] Android Open Source Project, «Documentation Guides - Linear layout,» 25 abril 2018. [En línea]. Available: https://goo.gl/4PX2mZ. [Último acceso: 01 septiembre 2018].

[36] M. McLemore, C. Dunn y B. Umbaugh, «Microsoft Docs - Activity Lifecycle,» 27 febrero 2018. [En línea]. Available: https://goo.gl/VK9qpc. [Último acceso: 03 septiembre 2018].

[37] Android Open Source Project, «Documentation Guides - Introduction to Activities,» 17 abril 2018. [En línea]. Available: https://goo.gl/ovUkwK. [Último acceso: 03 septiembre 2018].

[38] Android Open Source Project, «Documentation Guides - Servicios,» 25 abril 2018. [En línea]. Available: https://goo.gl/7Qtw3E. [Último acceso: 07 septiembre 2018].

[39] Android Open Source Project, «Documentation Guides - Manifiesto de la app,» 25 abril 2018. [En línea]. Available: https://goo.gl/iJQ33P. [Último acceso: 07 septiembre 2018].

[40] O. y T. Opgenorth, «Microsoft Docs - Permissions In Xamarin.Android,» 01 julio 2018. [En línea]. Available: https://goo.gl/vKxFnw. [Último acceso: 07 septiembre 2018].

[41] Android Open Source Project, «Documentation Guides - Uses SDK element,» 25 abril 2018. [En línea]. Available: https://goo.gl/7NYJBq. [Último acceso: 07 septiembre 2018].

[42] A. Burns, B. Umbaugh y C. Dunn, «Microsoft Docs - Introduction to Mobile Development,» 27 marzo 2017. [En línea]. Available: https://goo.gl/VkCRMU. [Último acceso: 12 septiembre 2018].

[43] D. Britch, C. Dunn, C. Petzold, M. Cooper y B. Umbaugh, «Microsoft Docs - An Introduction to Xamarin.Forms,» 12 enero 2016. [En línea]. Available: https://goo.gl/xDTC9b. [Último acceso: 13 septiembre 2018].

[44] L. Carvajal, Metodología de la Investgación Científica. Curso general y aplicado, 28 ed., Santiago de Cali: U.S.C., 2006, p. 139.

[45] J. Thurman, «Can Dying Languages Be Saved?,» *The New Yorker,* vol. XCI, nº 6, pp. 32-39, 2015.

[46] P. Helguera, «La Escuela Panamericana del Desasosiego,» 2009. [En línea]. Available: https://goo.gl/trjYLw. [Último acceso: 15 noviembre 2017].

[47] C. Petzold, «Creating Mobile Apps with Xamarin.Forms,» Microsoft Press, 2016. [En línea]. Available: https://goo.gl/mE2hNr. [Último acceso: 5 noviembre 2017].

[48] M. Collier y R. Shahan, «Fundamentals of Azure,» Microsoft Press, 2015. [En línea]. Available: https://goo.gl/jYrbT6. [Último acceso: 2017 noviembre 14].

[49] M. Vilchis, «La web 2.0 y la nube,» Universidad Autónoma del Estado de Hidalgo, [En línea]. Available: https://goo.gl/DHFpR3. [Último acceso: 6 agosto 2018].

6. ANEXOS

ANEXO I. Código fuente (Anexo digital)

ANEXO II. Instalación Xamarin

ANEXO III. Creación cuenta de almacenamiento Azure

ANEXO IV. Instalación librerías adicionales

ANEXO V. Importación metadatos Google Earth

ANEXO VI. Instalador de la aplicación (Anexo digital)

ANEXO VII. Archivo de visualización Excel (Anexo digital)

ANEXO VIII. Registros de audio (Anexo digital)

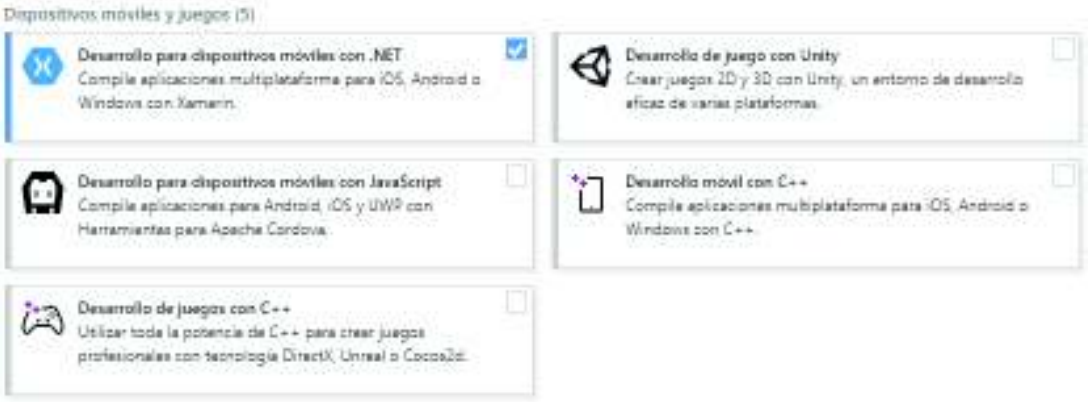
ANEXO IX. Encuesta de validación (Anexo digital)

ANEXO II Instalación Xamarin

1) Abrir dentro de Visual Studio Herramientas/ Obtener herramientas y características.

2) Seleccionar Desarrollo para dispositivos móviles con .NET.

3) Instalar

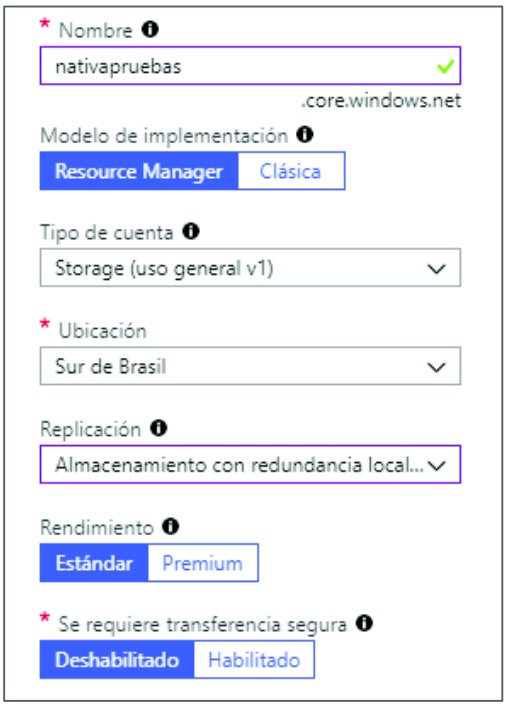


ANEXO III Creación cuenta de almacenamiento Azure

1) Crear una cuenta en el portal de Azure en https://portal.azure.com/

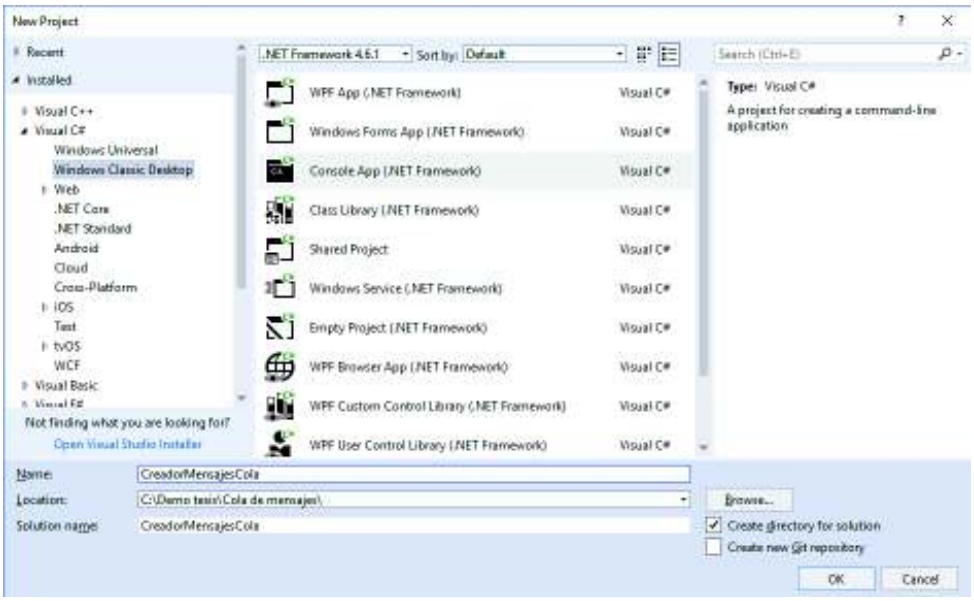
2) Crear una cuenta de almacenamiento, el nombre de esta cuenta solo puede contener letras minúsculas y números. Debe tener entre 3 y 24 caracteres.

3) Crear un grupo de recursos, el nombre del grupo puede estar conformado por caracteres alfanuméricos, puntos, guiones bajos, guiones y paréntesis, pero no pueden terminar con un punto.

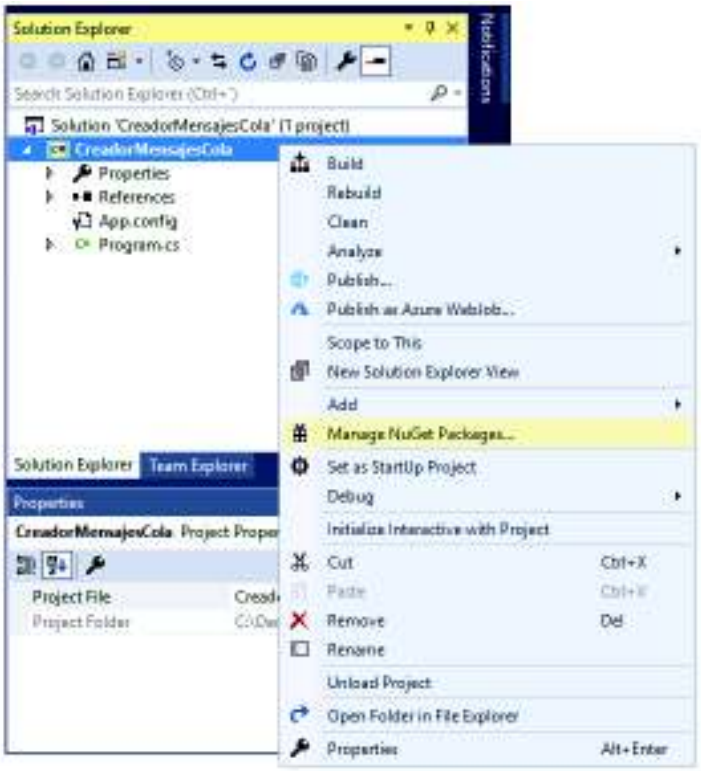


ANEXO IV Creación programa creador mensaje de colas

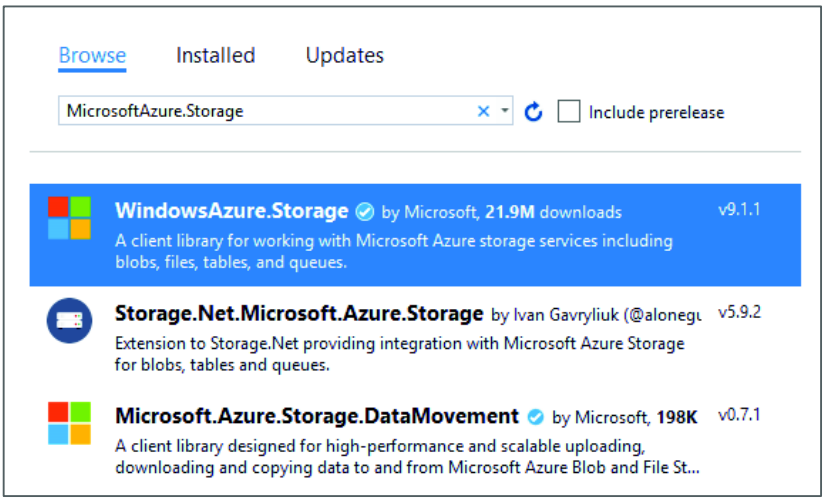
Para crear los mensajes de cola, creamos un nuevo proyecto en Visual Studio, crear una aplicación de consola bajo el contexto de escritorio clásico de Windows

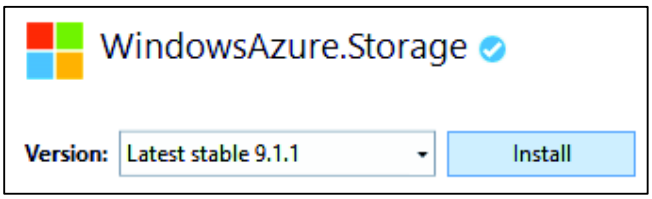


Luego de crear el proyecto, dar clic derecho sobre la aplicación y seleccionar el administrador de paquetes NuGet.



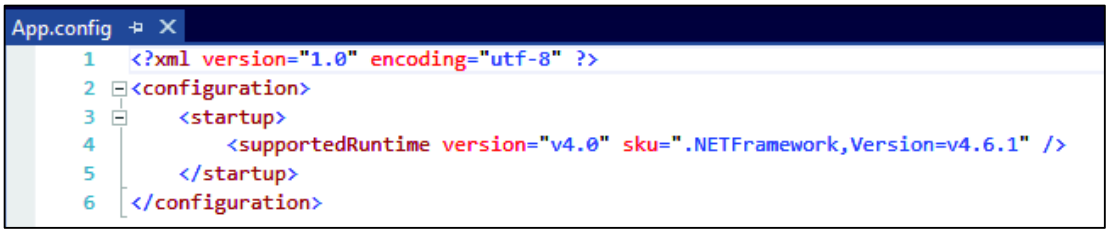
En el buscador del administrador de paquetes NuGet escribir Microsoft Azure Storage, seleccionar el paquete y dar clic en instalar, aceptar la licencia.





Instalar el paquete Microsoft.WindowsAzure.ConfigurationManager siguiendo los pasos para instalar el paquete Windows.Azure.Storage

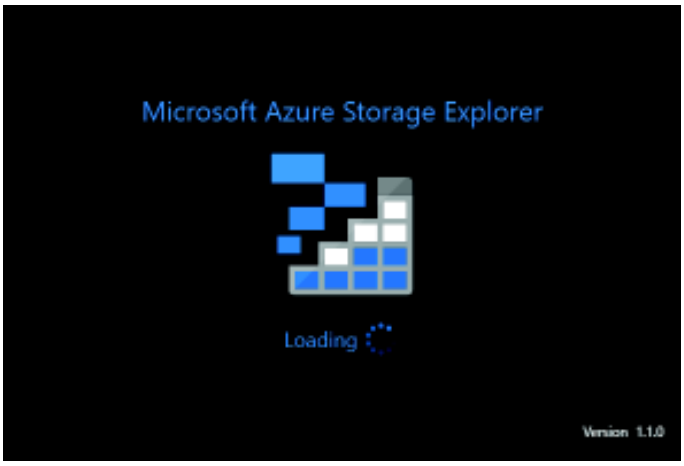
Agregar al archivo App.config de la aplicación el valor de la cadena de conexión, este valor se lo obtiene del portal de Azure, en la pestaña claves de acceso que se encuentra en las configuraciones de la cuenta de almacenamiento “nativapruebas”.



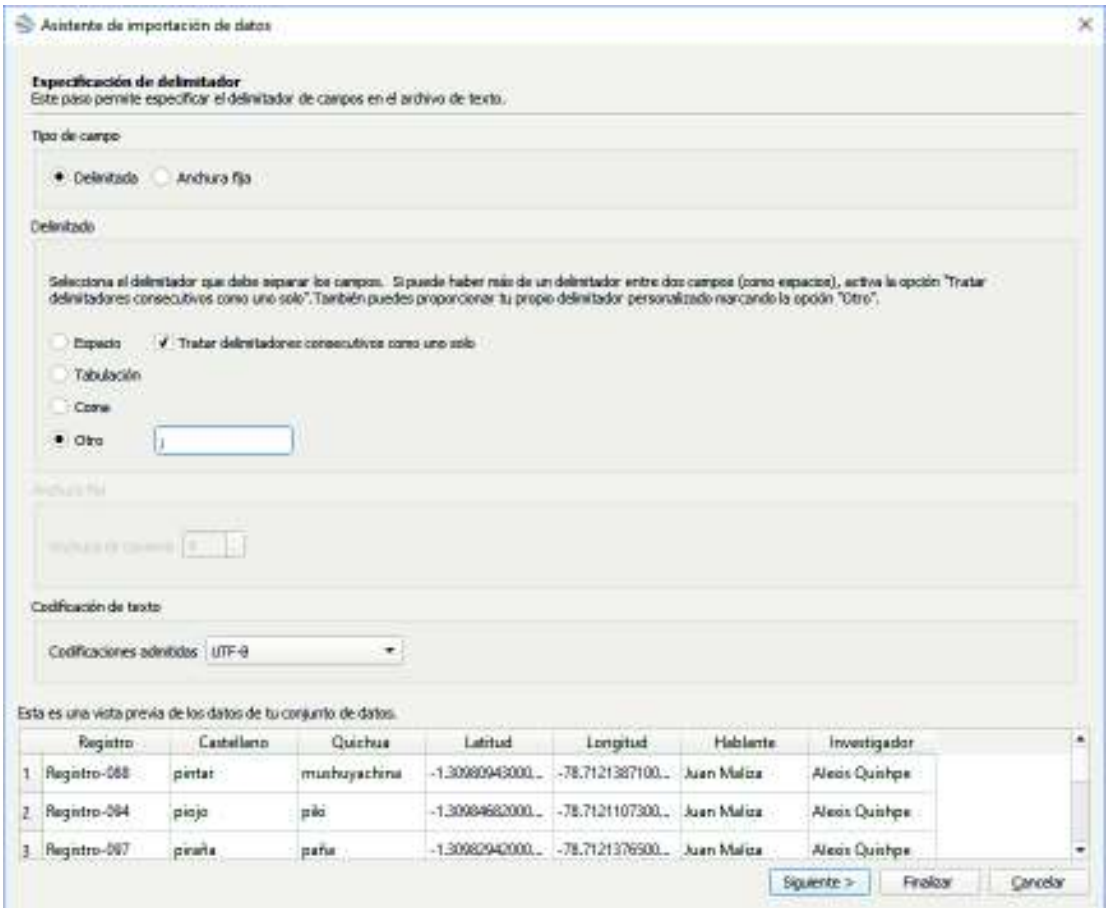
ANEXO V Instalación Azure Storage Explorer

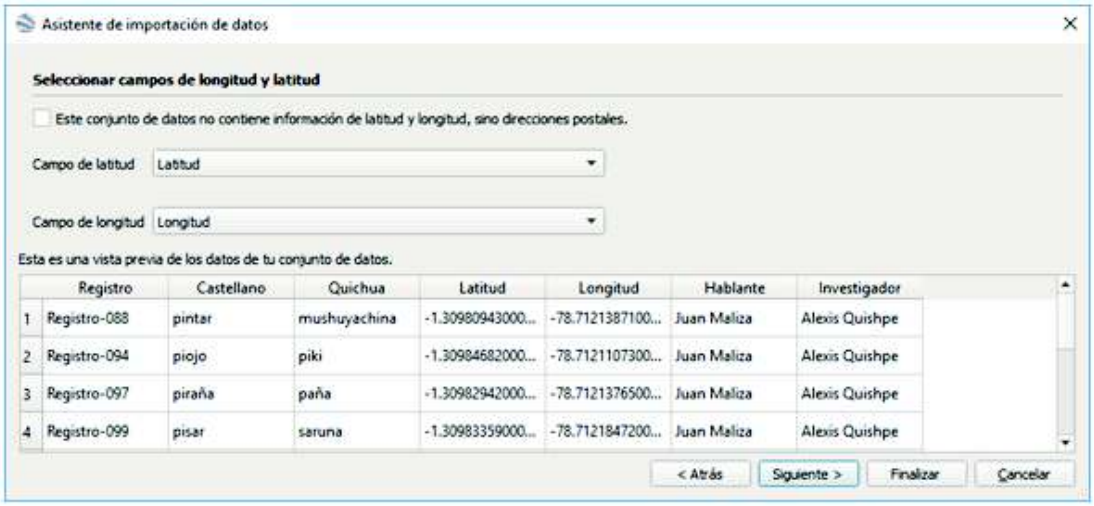
1) Descargar el instalador de Azure Storage Explorer desde el siguiente enlace https://azure.microsoft.com/en-us/features/storage-explorer/

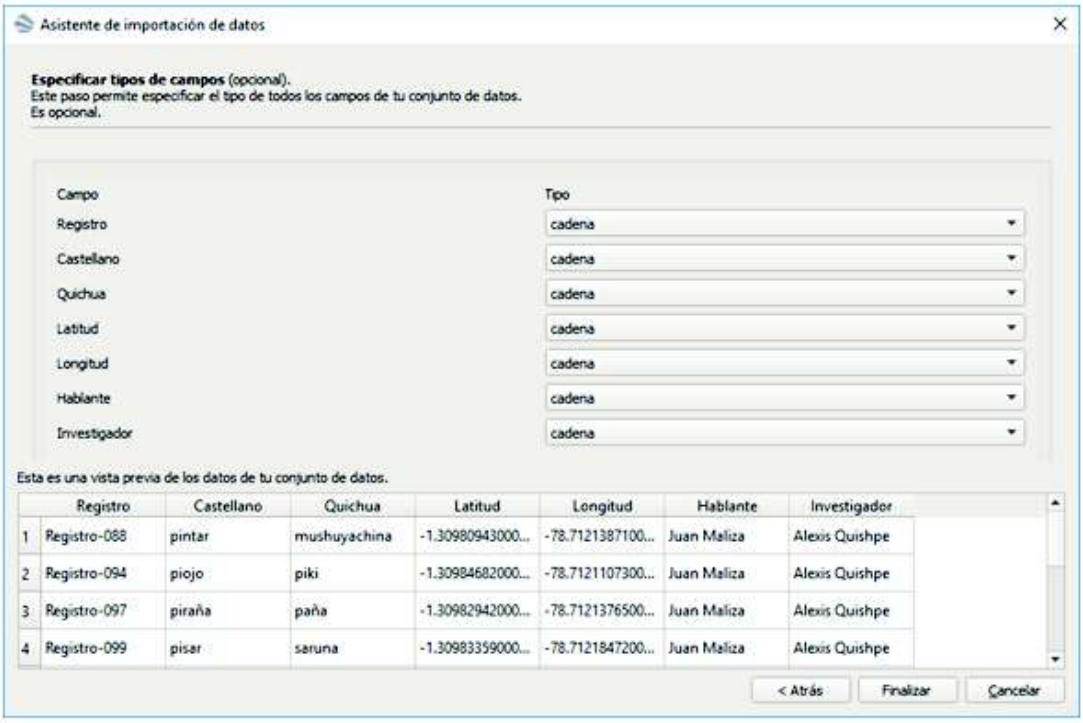
2) Instalar el archivo StorageExplorer.exe

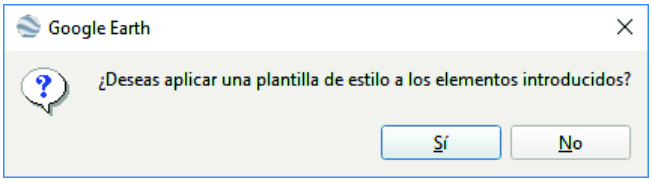


ANEXO VI Importación metadatos Google Earth

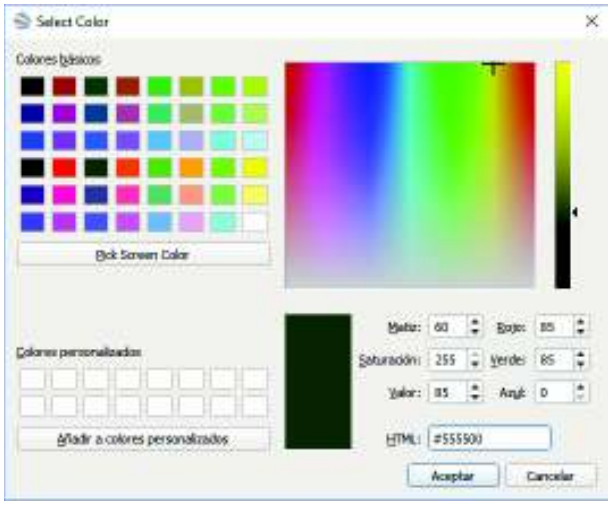





Asistente de importación de datos
Especificar tipos de campos (opcional).
Este paso permite especificar el tipo de todos los campos de tu conjunto de datos.
Es opcional.
Campo
Tipo
Registro
Castellano
Quichua
Latitud
Longitud
Hablante
Investigador
cadena
Esta es una vista previa de los datos de tu conjunto de datos.
Registro
Castellano
Quichua
Latitud
Longitud
Hablante
Investigador
1 Registro-088 pintar mushuyachina -1.30980943000... -78.7121387100... Juan Maliza Alexis Quishpe
2 Registro-094 piojo piki -1.30984682000... -78.7121107300... Juan Maliza Alexis Quishpe
3 Registro-097 piraña paña -1.30982942000... -78.7121376500... Juan Maliza Alexis Quishpe
4 Registro-099 pisar saruna -1.30983359000... -78.7121847200... Juan Maliza Alexis Quishpe
< Atrás
Finalizar
Cancelar


Google Earth
¿Deseas aplicar una plantilla de estilo a los elementos introducidos?
Sí
No


Configuración de plantilla de estilo
Nombre
Color
Icono
Altura
Establecer color desde campo
Utilizar un único color
Utilizar colores aleatorios
Esta tabla de vista preliminar contiene los diez primeros elementos del conjunto de datos.
Registro
Castellano
Quichua
Latitud
Longitud
Hablante
1 Registro-088 pintar mushuyachina -1.30980943000... -78.7121387100... Juan Maliza
2 Registro-094 piojo piki -1.30984682000... -78.7121107300... Juan Maliza
3 Registro-097 piraña paña -1.30982942000... -78.7121376500... Juan Maliza
4 Registro-099 pisar saruna -1.30983359000... -78.7121847200... Juan Maliza
5 Registro-102 piso pamba -1.30985457000... -78.7121361600... Juan Maliza
Aceptar
Cancelar


Select Color
Colores básicos
Pick Screen Color
Colores personalizados
Añadir a colores personalizados
Matiz: 60
Saturación: 255
Valor: 85
Rojo: 85
Verde: 85
Azul: 0
HTML: #555500
Aceptar
Cancelar

ORDEN DE EMPASTADO