ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“Desarrollo de una app y sistema web para el levantamiento de información de especies vegetales nativas”

INFORME DE MATERIA INTEGRADORA

Previo a la obtención del Título de:

INGENIERO EN CIENCIAS COMPUTACIONALES

Roberto Steven Bonilla Acosta

Jorge Mauricio Guzmán Contreras

GUAYAQUIL – ECUADOR

AÑO: 2017

AGRADECIMIENTOS

Mis más sinceros agradecimientos a mis profesores Boris Vintimila, PhD. y Denis Romero, PhD. por todo el apoyo brindado para que este proyecto tenga un buen término.

A mis padres por estar siempre apoyándome en todos los pasos que doy.

Roberto Bonilla Acosta

Mis agradecimientos los divido en dos partes: Los docentes de la ESPOL, en especial a mis tutores de este proyecto Boris Vintimilla, Ph. D. y Dennis Romero, Ph. D. por todo su apoyo pero también a todos aquellos que no alcanzo a nombrar pero de quienes aprendí no sólo conocimientos, sino valores como el esfuerzo y la excelencia. La segunda parte del agradecimiento es a los tres pilares que han hecho posible este logro: Dios, porque después de cometer ciertos errores me di cuenta que transitar por el camino correcto y respetar sus reglas tiene sus frutos; mis padres, por todo el apoyo incondicional que recibí de su parte; y finalmente mi perseverancia, que me ha ayudado a obtener varios logros, porque al revisar la historia de los mayores personajes en el mundo podemos observar que genios fracasados hay muchos, pero los que han llegado al éxito son aquellos que a pesar de los obstáculos que han tenido, se han mantenido luchando hasta alcanzar su meta.

En resumen, mis sinceros agradecimientos a quienes de una u otra forma han aportado en mi formación como profesional.

Jorge Mauricio Guzmán Contreras

DEDICATORIA

El presente proyecto lo dedico a mi esposa por ser la fuerza que me impulsa para seguir buscando nuevas metas.

Roberto Bonilla Acosta

El fruto de este esfuerzo lo dedico a mis padres por todo su apoyo brindado. Pero también para los jóvenes que se inician en la universidad, a quienes además les invito a soñar en grande y a saber que con esfuerzo y dedicación, todo es posible.

Jorge Mauricio Guzmán Contreras

TRIBUNAL DE EVALUACIÓN

Boris Vintimilla, Ph. D.	**Dennis Romero, Ph. D.**
PROFESOR EVALUADOR	PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, (nos) corresponde exclusivamente; y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Roberto Steven Bonilla Acosta

C. I. # 092601638 - 7

Jorge Mauricio Guzmán Contreras

C. I. # 092338828 - 4

RESUMEN

El problema que fue objeto de estudio para este proyecto es la falta de información suficiente de las especies vegetales de nuestro país, su distribución en nuestro territorio, las variedades, su estado de conservación, o la falta de entidades que provean suficiente información entorno a este tema.

La solución propuesta se basa en dar los primeros pasos en la creación de un sistema con información suficiente de nuestras especies vegetales nativas, que sea centralizada y al alcance de la comunidad. Con este objeto se ideó crear un repositorio central de datos de estas especies, que sea administrado por una página web por usuarios expertos, y que sea alimentada de información de las especies en campo por usuarios comunes a través de una aplicación móvil desarrollada para este fin. A nuestro proyecto lo llamamos LESVENA (Levantamiento de Información de Especies Vegetales Nativas).

Finalmente, se logró crear la aplicación móvil, el repositorio central con datos iniciales de prueba, el sistema web administrador de información, y la intercomunicación entre todas estas plataformas. De forma general de provee al usuario de 2 principales funcionalidades (descargando la app en su celular): Recolección de especies, que consiste en capturar fotos de especies con la cámara del dispositivo móvil, guardar esta foto y registrar los datos de la especie fotografiada, todos estos datos son guardados en memoria interna del dispositivo. La segunda gran funcionalidad en la aplicación es permitir el reconocimiento de especie en campo basados en la fotografía capturada por el usuario, el usuario toma la foto y obtiene los datos de la especie fotografiada. Todos estos datos se sincronizan en cierto momento con el repositorio central de datos alojado en nuestro servidor.

Tanto la metodología de desarrollo como los detalles técnicos aplicados para la implementación de nuestra solución son descritos a lo largo de los 4 capítulos que contiene el presente documento.

ÍNDICE GENERAL

CAPÍTULO 1

1. PROBLEMA A RESOLVER.

1.1 Antecedentes

El Ecuador es uno de los 17 países mega diversos del planeta [1], estos 17 países albergan más del 70% de la biodiversidad del planeta.

Gracias a esta biodiversidad y al clima de nuestro país podemos encontrar distintos biomas que de acuerdo al *cuarto informe nacional para el convenio sobre la diversidad biológica* "en su territorio continental, Ecuador posee siete biomas, los cuales son: bosques húmedos tropicales, bosques secos tropicales, sabanas, matorrales xerofíticos, bosques montanos, páramos y manglares. De manera más específica, el Ecuador continental tiene 25 zonas de vida, 18 formaciones geobotánicas y Harling propuso para Ecuador 16 tipos de vegetación." [1]

Taxón	*Especies conocidas*	*Especies amenazadas (%)*
Plantas vasculares	17058	1716 (10%)
Mamíferos	382	42 (11%)
Aves	1655	71 (4%)
Anfibios	464	171 (37%)
Reptiles	404	11 (3%)
Peces	1539	18 (1%)

Tabla 1: Número de especies conocidas y amenazadas en el Ecuador [1]

La Tabla 1 nos ayuda a comprender la gran variedad de especies que hay en el país y nos permite ratificar la importancia de tener un seguimiento de la salud de estos biomas.

Para poder hacer un seguimiento de estos biomas es necesario realizar levantamientos de información periódicos para analizar el crecimiento de las especies, determinar si hay especies invasoras, si los individuos de una población están presentando problemas de salud y muchas más implicaciones.

1.2 Situación actual

En la actualidad se dispone de pocas herramientas que permitan al usuario un reconocimiento inmediato de una especie vegetal nativa. De esto, pocos proyectos, al estar desarrollados en otra región, cuentan con información suficiente de datos que permita realizar actividades de investigación en estos campos como, por ejemplo, reconocimiento de especies vegetales nativas.

La creación y actualización de data sets se la realiza con métodos tradicionales sin que involucren mayor tecnología siendo el GPS el que ha realizado una mayor penetración en este tema.

1.3 Objetivos

1.3.1 Objetivo general

Crear una aplicación móvil y web para levantar información de las especies vegetales nativas.

1.3.2 Objetivos específicos

Crear una aplicación móvil que permita tomar fotos, guardar la georreferencia de donde se la tomó y enviarla a almacenar a un servidor de base de datos.

Devolver información sobre la especie a la que se fotografió, por medio del consumo de un sistema de reconocimiento de imágenes que debe ser entrenado.

Crear una aplicación web que permite informar y distribuir los data sets generados con la aplicación móvil.

1.4 Alcance

La aplicación móvil permite fotografiar especies vegetales nativas para su posterior reconocimiento por medio de un sistema de reconocimiento de

imágenes, estas mismas imágenes serán enviadas desde el móvil que toma la foto al servidor donde serán almacenadas en un sistema de base de datos y junto a la imagen se almacena la información de la georreferencia donde se tomó la foto, información que siempre debe de estar asociada a la imagen y para lograrlo la aplicación no funciona si el GPS del móvil está apagado.

Se creará un data set inicial con 2 a 3 tipos de flores que se encuentren en la ESPOL.

1.5 Trabajos similares

1.5.1 *Celebrate Urban Birds* (Celebra las Aves Urbanas)

Celebra las Aves Urbanas es un proyecto desarrollado en el 2007, en el Laboratorio de Ornitología de la Universidad de Cornell (*The Cornell Lab of Ornithology*).

Este trabajo cita entre sus objetivos el levantamiento de información sobre aves en áreas verdes urbanas, para lo cual fijan ciertos parámetros que el observador debe cumplir.

Este proyecto solo es una aplicación web que permite subir la información recopilada durante la observación y proporciona información, imágenes y sonidos, que nos ayuda a categorizar las aves observadas. [2]

1.5.2 PlantNet

Esta es una aplicación para la recopilación, anotación y recuperación de imágenes para ayudar en la identificación de plantas.

Fue desarrollada por un consorcio formado por científicos de CIRAD, INRA, INRIA, IRD, y la red Tela Botánica en virtud de un proyecto financiado por la Fundación Agrópolis.

Esta aplicación tiene un sistema de reconocimiento de imágenes que entrega información a los usuarios sobre la foto captada. Presenta un modo de contribución para la continua alimentación del sistema de reconocimiento de imágenes. [3]

1.5.3 Virginia Tech Tree Identification

Esta es una aplicación para la identificación de 969 plantas leñosas ubicadas en América del Norte.

Los usuarios podrán obtener información de las especies presentes en el sector que se encuentra activando el GPS o ingresando la ubicación, dirección o código postal del sitio de interés.

Para acotar la información devuelta se puede contestar preguntas relacionadas a la planta interesada o ingresar la especie en particular que se desea encontrar. [4]

1.5.4 Arbolapp

Esta es una aplicación para la identificación de plantas ubicadas en la península Ibérica y las islas Baleares.

La aplicación tiene una búsqueda guiada por medio de preguntas relacionadas a la planta interesada, forma de las hojas, distribución de las mismas, forma, color y tamaño de las flores entre otros. [5]

1.6 Solución propuesta

1.6.1 Solución esquemática

La idea escogida como solución del problema planteado en este proyecto consta de 3 módulos: Módulo WEB, Módulo APP, Módulo DATASET (contiene el *data set* para el almacenamiento de la información recabada).

El *data set* será alimentado por los otros dos módulos elaborados, pero además este estará recibiendo retroalimentación y siendo consumido por un servidor externo ya desarrollado y facilitado por el CIDIS. El módulo APP consta de una solución móvil para teléfonos inteligentes con cámara y Geo Localizador, el mismo que estará diseñado y planeado como herramienta que permita tomar imágenes, determinar la ubicación geográfica (en datos de latitud y longitud) de la imagen, especificar ciertas características y enviar el dato hacia el *data set*, en nuestro servidor.

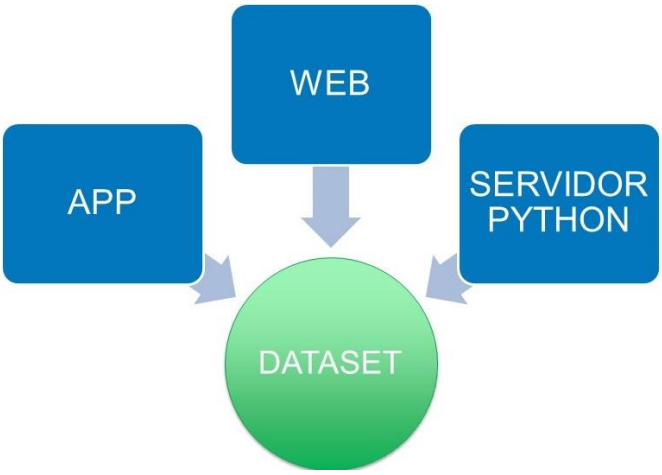


Figura 1.1: Esquema alimentación del *data set*

Se representa los módulos como está dividido todo nuestro sistema. El cuadrado APP representa el módulo APP, lo propio el cuadrado WEB representa el módulo WEB. El cuadrado SERVIDOR PYTHON representa un módulo externo del cual nuestro sistema se va a servir. Finalmente el circulo representa el ambiente servidor donde está alojado la base de datos del *data set*.

El módulo WEB consta de un sistema en ambiente web completo y nos permitirá proporcionar dos soluciones: administrar el contenido de la base de datos de la *data set* y, presentar mediante un Host público, los datos del *data set*, para que estos puedan ser fácilmente localizados y consultados por la comunidad científica y público en general. El módulo WEB es un sistema web de pequeño alcance pero totalmente funcional y orientado a transacciones y estará formado por dos partes: estructura de *front-end* y, estructura de *back -end* con conexión a base de datos.

1.6.2 Requerimientos para el diseño del dispositivo

En base a la idea escogida para nuestra resolución del problema planteado hemos determinado las siguientes herramientas técnicas:

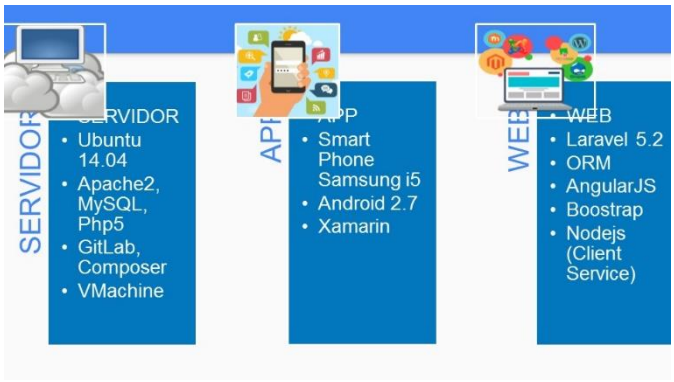


Figura 1.2: Características técnicas de cada módulo

Se presenta una figura representativa y las principales características técnicas de cada uno de los tres módulos de que está conformado nuestro sistema: DATASET (incluye la *data set*), APP Y WEB.

Para servidor web se usará sistema operativo Ubuntu 14.04, con ambiente LAMP, es decir servidor de aplicaciones web *Apache* 2, MySQL como motor de bases de datos, y PHP 5 como lenguaje de programación. Además, se requiere de software Git y página GitLab para gestionar el versionamiento del proyecto. Todo este ambiente se implementará en una máquina virtual con capacidades iniciales de 50 GB de almacenamiento, 6GB de memoria, sobre plataforma Windows 10 X64 y computadora base de procesador Core i5.

Para la aplicación móvil se usará como base de desarrollo las siguientes herramientas: Teléfono ´Smart Phone´ Samsung i5, sistema operativo *Android*. Además, para el desarrollo del código del proyecto usaremos la herramienta *Xamarin Visual Studio*, con C# como lenguaje de programación.

1.6.3 Estructura del proyecto

En base a la idea escogida para nuestra resolución del problema planteado hemos determinado la arquitectura de software establecida en la Figura 1.3. Además, ya hemos asignado un nombre a nuestro proyecto: LESVENA (Levantamiento de información de Especies Vegetales Nativas).

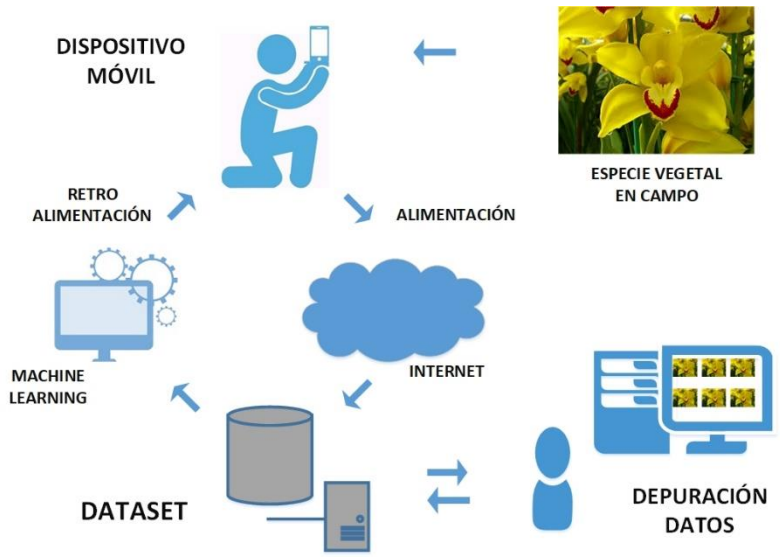


Figura 1.3: Arquitectura general del sistema

Los grupos de figuras agrupadas son representaciones de las arquitecturas que son usadas en el respectivo módulo. Es así que a la figura de nube y de motor de base de datos representa el servidor que contiene al Data Set, la pantalla de laptop con divisiones representa una página web y la imagen del celular representa la aplicación móvil.

CAPÍTULO 2

2. METODOLOGÍA TECNOLÓGICA.

En este capítulo se describe la metodología que se usa para definir la solución para el proyecto LESVENA. La metodología general del proyecto es una división en 3 principales módulos de acuerdo a las características tecnológicas de sus partes: APP, WEB, DATASET, pero luego en cada módulo se han usado metodologías adaptadas a sus características tecnológicas, tal es el caso de los módulos APP y WEB donde se han usado la metodología OOHDM que se basa en un conjunto de diagramas y etapas para visualizar las disposiciones de pantallas, funcionalidades e interacciones con los demás sub sistemas, mientras que en el módulo DATASET se ha usado una metodología de división en sub módulos según sus tareas. Tanto la planificación como la implementación de estos módulos se han realizado de forma simultánea puesto que todos los módulos funcionan de forma independiente pero correlacionada entre sí. En cuanto a las características del proyecto, se planifica LESVENA con las capacidades tecnológicas necesarias para el levantamiento de información de especies vegetales, haciendo uso de una aplicación móvil con tecnología *Android* en el módulo APP, una página web basada en tecnología PHP para la depuración de la base de datos central en el módulo WEB, y un servidor web con sistema operativo *Linux* como repositorio central de la información levantada más un API REST en lenguaje PHP para la conexión con la máquina de reconocimiento de imágenes, todo esto en el módulo DATASET.

En la siguiente sección se explicará con detalle en qué consiste cada módulo y las metodologías usadas para el desarrollo de cada uno de estos.

2.1 Metodología para el desarrollo del proyecto

En la metodología de desarrollo del proyecto se debe especificar las técnicas, procesos y tecnologías con las que se guía el equipo de desarrollo para la implementación del mismo. Es así que se ha planificado el desarrollo de este

proyecto dividiéndolo en módulos separados de acuerdo a sus componentes tecnológicos.

Los 3 principales componentes tecnológicos de este proyecto son: una aplicación móvil cuyo desarrollo es cubierto en un módulo al que llamamos APP, un sistema web en línea y disponible en una dirección IP pública cuyo desarrollo es cubierto en un módulo al que llamamos WEB, y el ambiente físico – digital del servidor en *Linux* cuyo desarrollo es cubierto en un módulo al que llamamos DATASET.

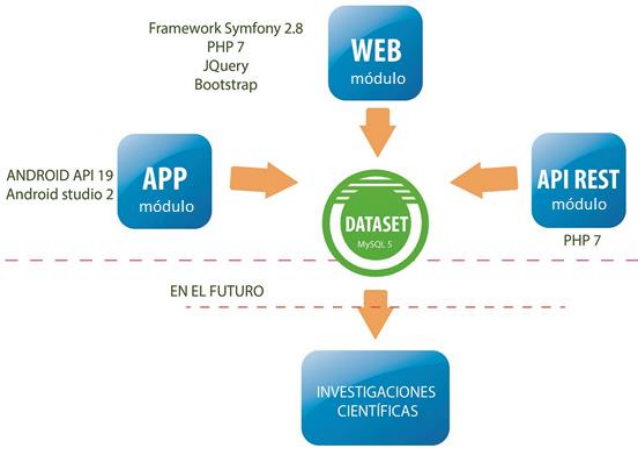


Figura 2.1 Metodología de desarrollo del proyecto

La Figura 2.1 presenta la forma en que está dividido el proyecto de acuerdo a sus módulos donde se incluye las principales características tecnológicas correspondientes a cada uno. El círculo verde representa el módulo DATASET, que contiene la base de datos central en la cual se almacena las fotos y datos recolectados de las especies vegetales. Cada uno de los rectángulos de color azul representa los módulos que interactúan con el módulo DATASET.

Luego, tenemos que el módulo APP representa la aplicación móvil desarrollada sobre la API 19 de Android a través del entorno de desarrollo *Android Studio* 3.0 y cuya función es alimentar al *data set* con los datos y fotografías recolectadas; el módulo WEB representa al sistema web desarrollado sobre PHP 7 con framework Symfony 2.8 del lado servidor mientras que *Bootstrap* del lado cliente y cuya función es administrar los datos registrados en la base de datos central

para hacer depuración continua de los datos registrados; el sub módulo API REST desarrollado sobre tecnología PHP 7 cuya función es establecer un conjunto de comandos para la gestión de la comunicación entre el dispositivo móvil y el sistema incrustado (tecnología *Python*) para reconocimiento de imágenes, esto para permitir el funcionamiento de la opción Reconocimiento de la aplicación móvil. Finalmente el rectángulo inferior representa uno de los posibles usos que se dará a la información recopilada y constantemente depurada de la *data set*: las investigaciones científicas. A continuación se explicará en detalle en las secciones siguientes la metodología, procedimiento, herramientas y arquitectura que se definen para la solución de cada uno de los módulos.

2.2 Módulo APP

En esta sección describiremos la metodología, el procedimiento, las herramientas y la arquitectura que han sido establecidas para el desarrollo de nuestra aplicación móvil para dispositivos con sistema operativo *Android* cuyas principales funcionalidades son: tomar fotografías y enviarlas a través de la internet hacia un servidor con dominio público, determinar la ubicación geográfica del usuario, recopilar un conjunto de datos en un formulario y enviar esta información hacia el servidor con dominio público, gestionar el almacenamiento interno de datos estructurados con una base de datos SQLite, gestionar envió – recepción de datos por HTTP para realizar sincronizaciones periódicas entre la base de datos interna de la aplicación y la base de datos central ubicada en el servidor con dominio público, y finalmente presentar pantallas de usuario final con diseños actuales, amigables y que permite interacciones prácticas con usuario final.

Para definir la solución de este módulo hemos usado la Metodología de Diseño Hipermedia Orientada a Objetos (OOHDM), la cual es una metodología propicia para el desarrollo de sistemas que tienen interacción con usuario final debido a que está basada en etapas de diseño visuales del proyecto permitiendo de esta forma la interacción entre el equipo de desarrollo y el cliente. La idea detrás de esta metodología es primero involucrar al cliente con el producto que ha solicitado

y la segunda intención es que el equipo de desarrollo obtenga retroalimentación de parte del cliente para que los cambios o nuevos rumbos en cuanto al proyecto puedan hacerse durante el tiempo de desarrollo del proyecto más no cuando esté terminado. Esta metodología es idónea especialmente en desarrollo de proyectos a gran escala.

Se debe indicar que las secciones 2.2.2 Procedimiento, 2.2.3 Herramientas y 2.2.4 Arquitectura, aunque son parte de la sección 2.2.1 de Metodología de Desarrollo OOHDM, han sido consideradas fuera de esta debido a que son partes esenciales que completan la definición de la solución para el mismo.

2.2.1 Metodología de desarrollo OOHDM

La cantidad de aplicaciones de software hipermedia crece de forma acelerada hoy en día, además, se ha estado volviendo cada vez más complejas debido a su alcance y las tecnologías de vanguardia. Es por esto que los equipos de desarrollo de software en la actualidad se han visto necesitados de aplicar metodologías de desarrollo de software que sean sistemáticas, puesto que se necesita que cada fase del proyecto pueda ser estudiada, obtener retroalimentación de parte del usuario final y evolucionar en caso de ser necesario, también aprovechar estas fases de construcción para aplicar reusabilidad utilizando fases de un proyecto en otro.

La Metodología de Desarrollo Hipermedia Orientado a Objetos (OOHDM por sus siglas en inglés) cubre los requerimientos de un proceso de desarrollo sistemático puesto que es basado en otra metodología sistemática como HDM (*Hypermedia Develop Method*) pero con el valor agregado de habérsela orientado a objetos para contemplar abstracciones visuales - modelos del proyecto y la independencia de los elementos en sistemas hipermedia. OOHDM está compuesto por 5 etapas en total aunque la primera muchas veces no es tomada en consideración por ser fase de levantamiento de información. Sus 4 etapas siguientes y cuales son aplicadas a nuestro proyecto son: diseño conceptual, diseño de flujo de navegación, diseño de interfaces abstractas e implementación.

En la etapa de diseño conceptual se busca obtener una correcta abstracción semántica del dominio del proyecto valiéndose de diagramas de modelado estructurales comúnmente usados en orientación a objetos. Las metodologías tradicionales de ingeniería de software no han sabido realizar buenas tareas de abstracción que faciliten la tarea de especificar aplicaciones hipermedia, lo cual se procura corregir con esta fase. En la etapa de diseño de flujo de navegación se plantea un contexto de flujo de navegación, el cual es un espacio de estudio de un escenario específico de ejecución del proyecto y está formado por nodos, enlaces, clases de contextos, y otros contextos de navegación, cuya principal tarea es organizar el espacio de flujo de navegación en conjuntos convenientes que puedan ser recorridos en un orden particular y que deberían ser definidos como caminos que le permitan al usuario conocer bien donde puede ir y como llegar al lugar deseado. En la etapa de diseño de interfaces abstractas se plasman las decisiones de diseño visual para el proyecto teniendo en consideración que deben ser tan independientes que los diseños debidos a la estructura visual para el flujo de navegación pueden ser desacoplados de aquellos debidos a la estructura visual para el modelo del dominio, pero trabajar de manera sinérgica al mismo tiempo. Finalmente en la etapa de implementación se debe especificar de manera detallada las decisiones en cuánto a aspectos técnicos en general como son: las tecnologías que serán usadas en cada fase del proyecto, las herramientas tecnológicas para la construcción del proyecto, las arquitecturas internas e incluso los procedimientos que se deberán seguir durante su desarrollo. [6]

2.2.1.1 Diseño conceptual

En esta fase se deben crear esquemas conceptuales que representen a los objetos del dominio, sus relaciones y subsistemas.

Básicamente se representa un diseño conceptual de un proyecto a partir del modelado de datos semánticos estructurales, idéntico al modelado estructurado de datos que se realiza en UML. En

OOHDM se construye este modelado estructurado con elementos de clases, relaciones y subsistemas predefinidos y que son similares a UML excepto tal vez por los atributos de las clases que pueden ser de múltiples tipos para representar perspectivas diferentes de los mismos objetos del mundo real. Usualmente son usados recursos agregados a este modelado a medida que el sistema aumenta su complejidad. [6]

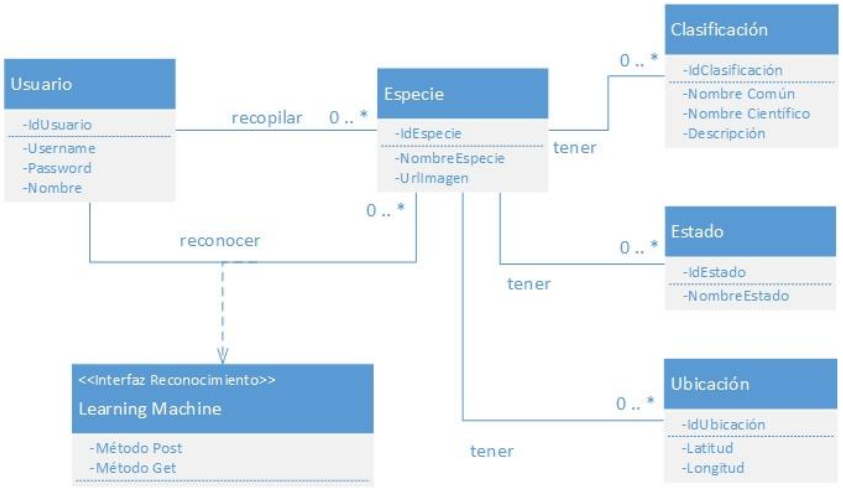


Figura 2.2: Diagrama de clases módulo APP

La Figura 2.2 muestra un modelado conceptual en UML para el módulo APP. Las relaciones recopilar y reconocer, dadas ambas entre las entidades Usuario y Especie, indican las actividades básicas a desarrollarse en la aplicación móvil. La aplicación permitirá al usuario registrar datos de especies en la recopilación, mientras que permitirá al usuario hacer reconocimiento de especies con la ayuda de *Machine Learning* incrustada en el proyecto. Las demás entidades son propiedades extendidas de la entidad Especie.

2.2.1.2 Diseño de flujo de navegación

En OOHDM la navegación es considerada un proceso crítico en el diseño de aplicaciones. Un modelo de flujo de navegación debe presentar una o varias vistas del modelo conceptual, debe ser una instancia del modelo conceptual en tiempo de ejecución. Además se puede contemplar modelos diferentes de acuerdo al usuario cual sea el caso de estudio.

Un diseño de flujo de navegación puede ser expresado en dos tipos de esquemas: el esquema de clases para flujo de navegación y el esquema de contexto para el flujo de navegación. Para el modelamiento de nuestro proyecto nos hemos servido del esquema de contexto para el flujo de navegación. Para el esquema de clases, OOHDM provee un conjunto de tipos predefinidos de clases para flujo de navegación: nodos, enlaces y estructuras de acceso. En este contexto el concepto de nodos y enlaces es el mismo de aquellos elementos que pertenecen a las metodologías tradicionales para modelamiento de aplicaciones hipermedia. [6]

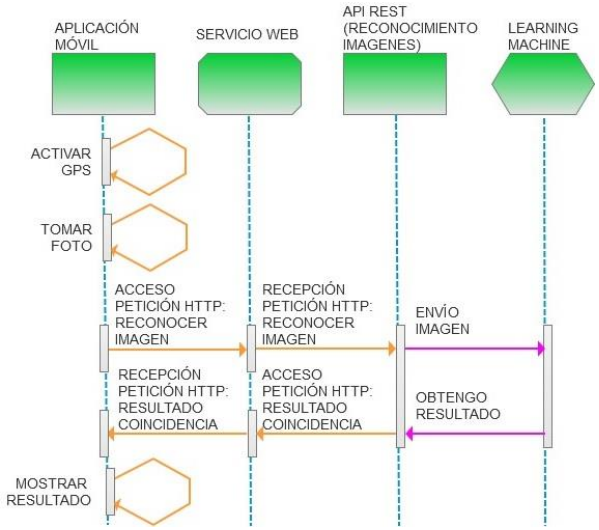


Figura 2.3: Flujo navegación en APP: funcionalidad recopilar

La Figura 2.3 muestra el flujo de actividades en la aplicación móvil para la funcionalidad recopilar. Las figuras en color verde representan los entornos que participan. En el entorno APLICACIÓN MÓVIL las tres primeras actividades se representan con flujos hacia sí mismo debido a que son actividades que se realizan dentro de la aplicación, como: activación del GPS, captura de fotos, o registro de datos. Mientras que, la última actividad representa la sincronización de la base interna de la APP con la base en el servidor principal a través de un servicio web.

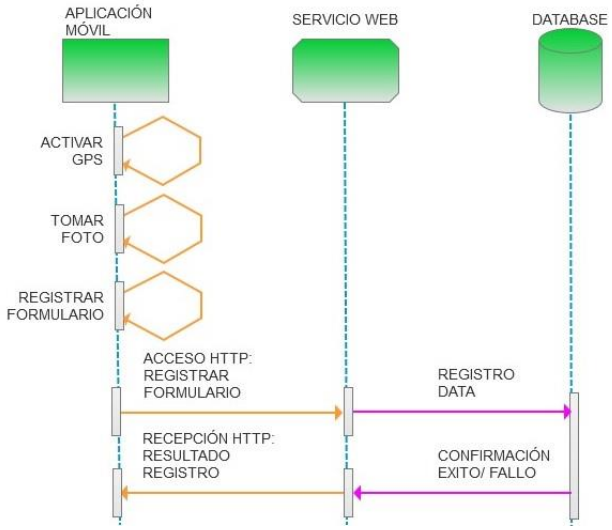


Figura 2.4: Flujo navegación en APP: funcionalidad reconocer

La Figura 2.4 muestra el flujo de actividades en la aplicación móvil para la funcionalidad reconocer. Las figuras en color verde representan los entornos que participan. En el entorno APLICACIÓN MÓVIL las dos primeras actividades se representan con flujos hacia sí mismo debido a que son actividades que se realizan dentro de la aplicación, como: activación del GPS o captura de fotos. Mientras que, la penúltima actividad representa la actividad de envío – recepción de datos

de especie en APP hacia la *Machine Learning* a través de las interfaces de comunicación: SERVICIO WEB y API REST. Finalmente, la última actividad es mostrar resultado de reconocimiento dentro de la propia aplicación móvil.

2.2.1.3 Diseño de interfaces abstractas

En esta fase deben ser definidas las estructuras visuales del proyecto: la forma en la cual aparecen los objetos del flujo de navegación, como los objetos de estas estructuras visuales activarán la navegación y a su vez como estos objetos activarán el resto de la funcionalidad de la aplicación.

Si la fase de flujo de navegación y esta fase de diseño de interfaces abstractas están correctamente elaboradas para el proyecto y por ende existe una clara separación entre estas, entonces será posible construir diferentes interfaces visuales para un mismo modelo de flujo de navegación, lo cual permitirá que para el proyecto sea transparentes las tecnologías de interfaz visual que sean implementadas. En esta fase generalmente son usados elementos para maquetado visual tales como: *mock ups, wareframes*, y prototipos en general. [6]

- Inicio aplicación

Figura 2.5: Prototipo pantallas APP: pantalla Inicio

En la Figura 2.5 se presenta el prototipo de pantalla (a) Inicio la cual es la pantalla de presentación de la aplicación. Esta pantalla tiene dos botones: CONSULTAR y RECOPILAR, acompañados del logo del proyecto. Al dar clic en botón CONSULTAR la aplicación muestra el conjunto de pantallas y funcionalidades para el modo de consulta. Mientras que, al dar clic en botón RECOPILAR se muestra el conjunto de pantallas y funcionalidades para el modo de recopilación de datos.

- Aplicación modo consulta

Figura 2.6: Prototipo pantallas APP: pantallas (a) Captura y (b) Reconoce

En la Figura 2.6 se presentan dos prototipos de pantalla. En pantalla (a) Captura se presenta la visualización de cámara del dispositivo móvil. En pantalla (b) Reconoce se muestra la foto que capturó el usuario y un botón. Al dar clic en botón RECONOCER se enviará la foto al servidor de reconocimiento de imágenes.

Figura 2.7: Prototipo pantallas APP: pantallas (a) Coincidencia y (b) Característica

En la Figura 2.7 se presentan dos prototipos de pantalla. En pantalla (a) Coincidencia se muestra una pequeña lista de resultados que es recibida desde el reconocedor de imágenes, con las 4 o 5 especies con mayor probabilidad de coincidencia. En pantalla (b) Característica se muestra un resumen de las características de la especie que más coincide.

- Aplicación modo recopilación

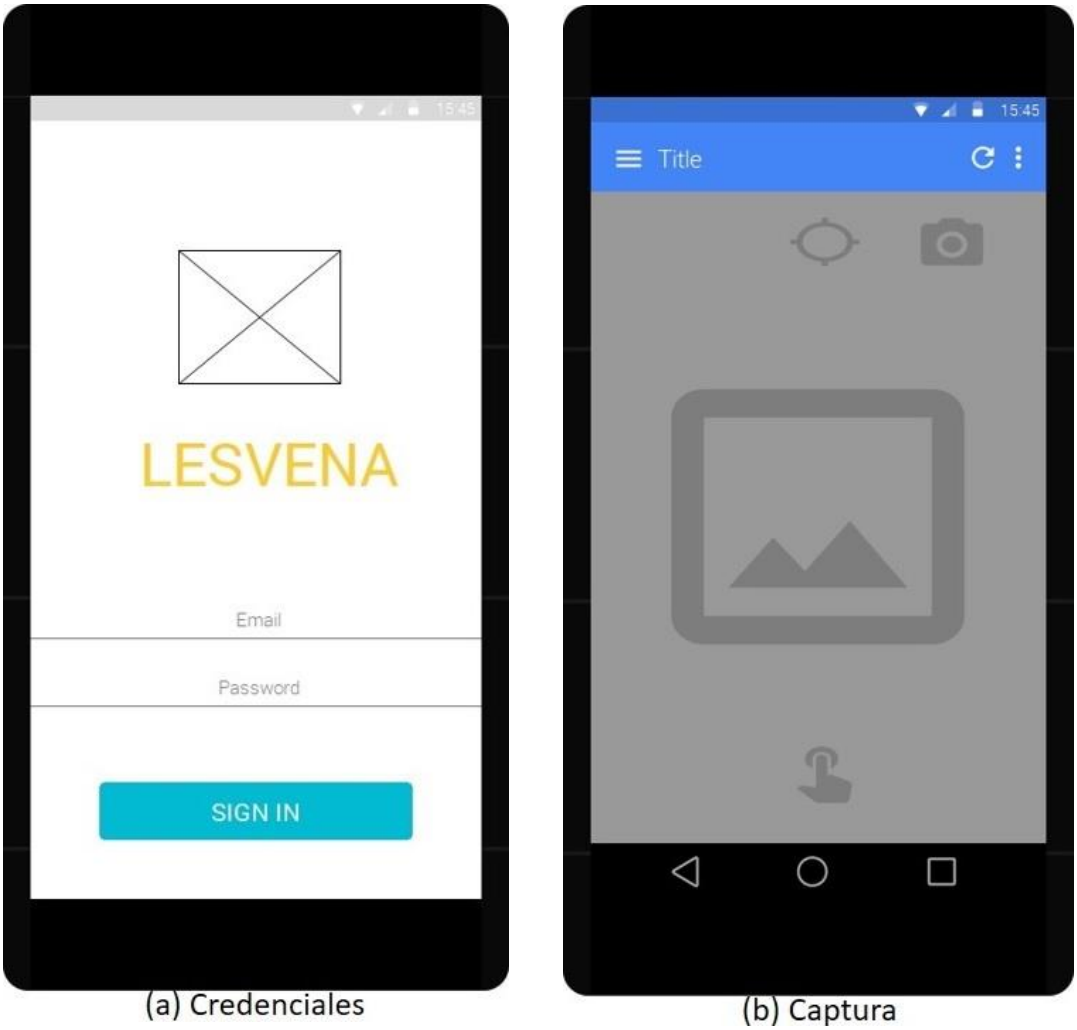


Figura 2.8: Prototipo pantallas APP: pantallas (a) Credenciales y (b) Captura

En la Figura 2.8 se presentan dos prototipos de pantalla. En pantalla (a) Credenciales se presenta una página de inicio para autenticación de usuarios. Al dar clic en botón AUTENTICAR se muestra el conjunto de pantallas y funcionalidades para modo recopilación de datos. En pantalla (b) Captura se presenta la visualización de cámara del dispositivo móvil.

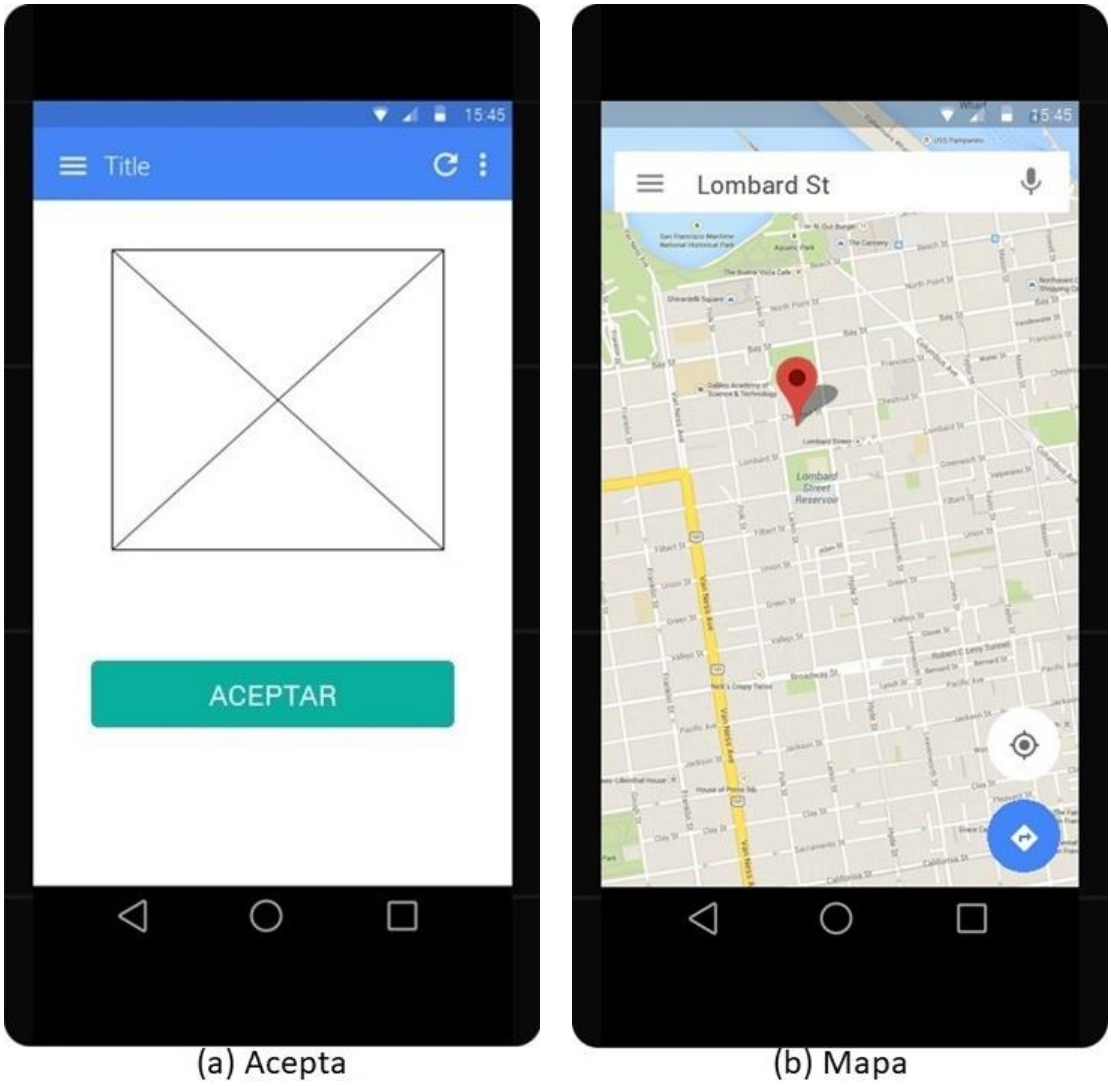


Figura 2.9: Prototipo pantallas APP: pantallas (a) Acepta y (b) Mapa

En la Figura 2.9 se presentan dos prototipos de pantalla. En pantalla (a) Acepta se muestra la foto que capturó el usuario y un botón. Al dar clic en botón ACEPTAR se enviará la foto al servidor y su link será grabado en SQLIte. En pantalla (b) Mapa se presenta una visualización de la aplicación Google Maps en la cual aparece un icono de ubicación para la especie capturada.

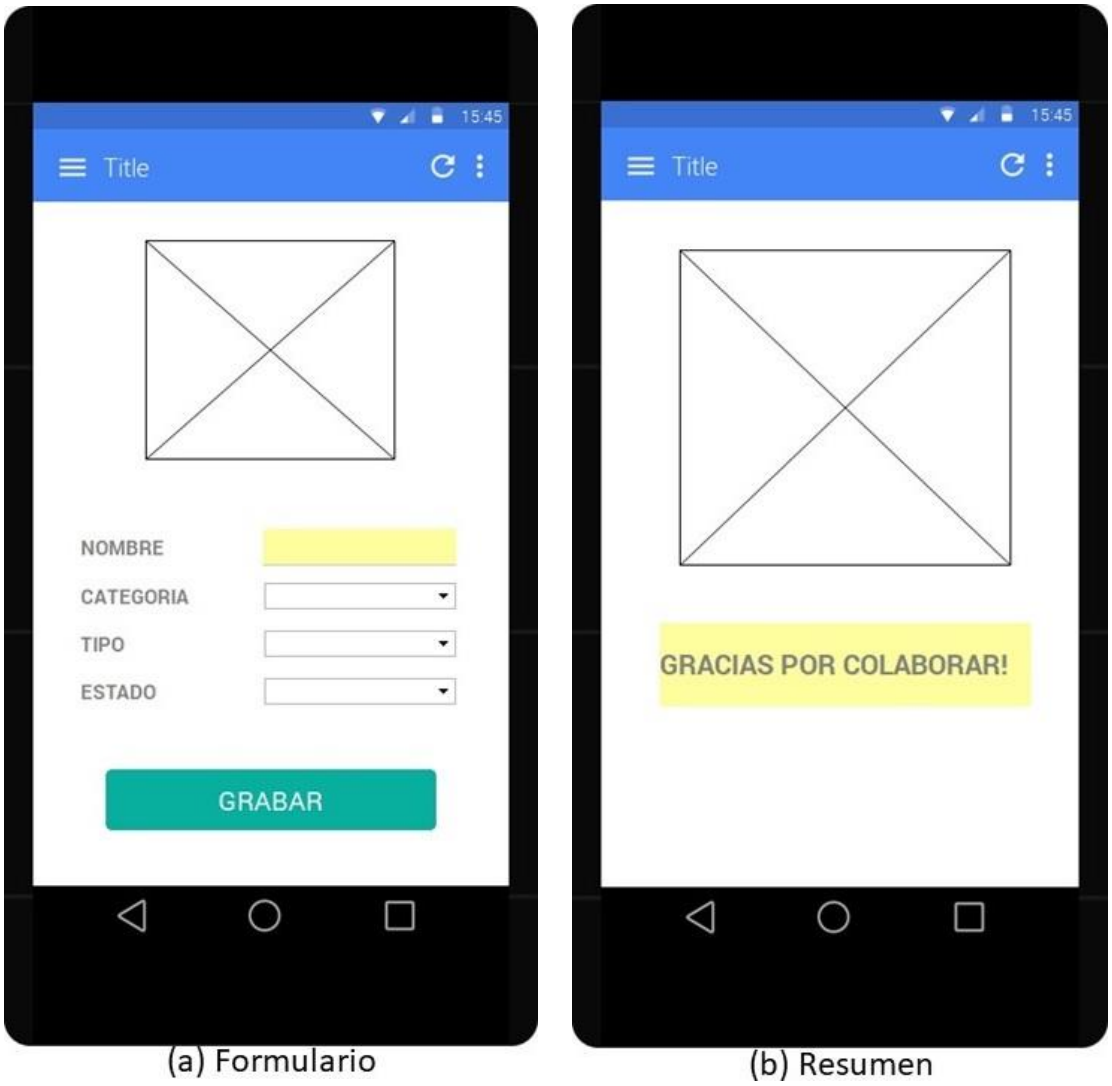


Figura 2.10: Prototipo pantallas APP: pantallas (a) Formulario y (b) Resumen

En la Figura 2.10 se presentan dos prototipos de pantalla. En pantalla (a) Formulario se maquetan tres secciones: La foto que fue capturada por el usuario, cajas de selección múltiple para CATEGORIA, ESPECIE y ESTADO con datos cargados desde de la base, y finalmente el botón GRABAR. En pantalla (b) Resumen se presenta la fotografía capturada y justo abajo un breve mensaje de agradecimiento.

2.2.1.4 Implementación

En esta fase el diseñador debe especificar los elementos y procedimientos para lograr implementar el diseño que ha plasmado en las fases anteriores. Se debe especificar un entorno particular en el cual va a correr la aplicación y en base a este entorno se deberán además especificar con detalle: las herramientas tecnologías para la implementación del proyecto, la arquitectura interna que tendrá, y el procedimiento que se debe seguir durante el desarrollo del proyecto hasta su puesta en marcha, los mismos que son descritos en las siguientes secciones. [6]

2.2.2 Procedimiento

Inicialmente escogimos las tecnologías de *Xamarin* para el desarrollo de nuestra aplicación móvil debido a su característica de compilación a los tres más importantes sistemas operativos del mercado en la actualidad: *Android*, iOS y *Windows Phone*.

Luego tuvimos que investigar las características, disponibilidad en el mercado y requisitos tanto para la adquisición de las herramientas como para aprender su programación debido a que ninguno de los integrantes del grupo tenía conocimiento de programación para teléfonos móviles.

Las revisiones bibliográficas realizadas produjo como resultado la siguiente desventaja con *Xamarin* en contrapeso de su ventaja de ser multiplataforma: la curva Aprendizaje vs Tiempo tiene bastante amplitud. El tiempo del que disponemos para desarrollar este proyecto es corto entonces esta razón nos llevó a inclinarnos por buscar herramientas más sencillas pero con menor tiempo de demanda en su aprendizaje y desarrollo.

Una alternativa que rápidamente encontramos fue la tecnología *Android* de *Google* que actualmente está dominando el mercado de las tecnologías móviles.
Encontramos tres principales ventajas para el desarrollo de aplicaciones nativas *Android*: Aprender a desarrollar aplicaciones móviles sencillas demanda de un relativo corto tiempo; existe una vasta documentación disponible en internet; su lenguaje para programación es Java, que es un lenguaje de programación muy conocido tanto por nosotros como en general en el mercado a la actualidad. Basados en las 3 ventajas mencionadas escogimos esta tecnología para el desarrollo de nuestra aplicación móvil.
Una vez que escogimos la tecnología en la cual desarrollar nuestra aplicación móvil, lo siguiente fue escoger las herramientas para el desarrollo de nuestra aplicación.

Inicialmente intentamos desarrollar con el IDE Eclipse y empezamos a conseguir las herramientas como Java SDK, SDK para *Android*, AVD (Android Virtual Device) y *Targets* para versiones del sistema operativo, pero encontramos inconvenientes al intentar integrar y hacer trabajar de manera sinérgica a todos.

Pudimos probar entonces el IDE *Android Studio* promocionado por el propio *Google* y encontramos que este IDE ya trae integradas las piezas como los SDK, distintas *targets* para sistemas operativos *Android*, emuladores de dispositivos móviles, etc. Para el desarrollo de la aplicación no fue necesario de un dispositivo móvil físico, ni siquiera para las compilaciones debido a que *Android* permite descargar distintas versiones de emuladores.

Android Studio cuenta además con un administrador de descargas para las dependencias para el SDK de *Android* y los emuladores de dispositivos móviles. Por todas estas ventajas decidimos desarrollar con este IDE.

2.2.3 Herramientas

De acuerdo a la tecnología escogida para el desarrollo de nuestra aplicación móvil, que fue descrita en la sección Procedimiento para este módulo, las herramientas que proyectamos utilizar se listan a continuación:

- SDK (JDK 7u79, JRE 7u79): Kit de desarrollo para entornos *Java* en ordenadores que contiene varios drivers, herramientas y recursos para programar en *Java*.
- IDE: Entorno de desarrollo *Eclipse for Java Developers* que nos provee un entorno para desarrollar y compilar en código Java, que incluye los paquetes: Git client, editor XML, Mylyn e integración para *Maven*.
- SDK *Android* r21: Kit que contiene varios drivers, herramientas y diversos recursos para programar en *Android* 4.2, así como distintas APIs facilitadas por *Google* tanto para el control de las funciones del dispositivo como para la integración de servicios, un depurador, un emulador para testear las aplicaciones y documentación básica.
- *Plugin* ADT (*Android Development Tools*): Un *plugin* para adaptar al IDE Eclipse y que provee funcionalidades para la programación en el sistema operativo.
- Complemento ADT: Usamos un AVD (Dispositivo Virtual Android) como complemento de las herramientas integradas, el cual nos permite emular varios dispositivos móviles tanto comerciales como genéricos para probar nuestras aplicaciones.
- SDK *Targets* de *Android*: Son librerías necesarias para desarrollar en cada una de las versiones concretas de *Android*.

2.2.4 Arquitectura

De acuerdo a las funcionalidades y tecnologías requeridas para nuestra aplicación móvil, establecimos su arquitectura, la cual queda representada por la gráfica de abajo. En la Figura 2.11 se representa tanto la

arquitectura interna de la aplicación móvil como la arquitectura de interacción de la aplicación móvil con los recursos externos de quienes depende.

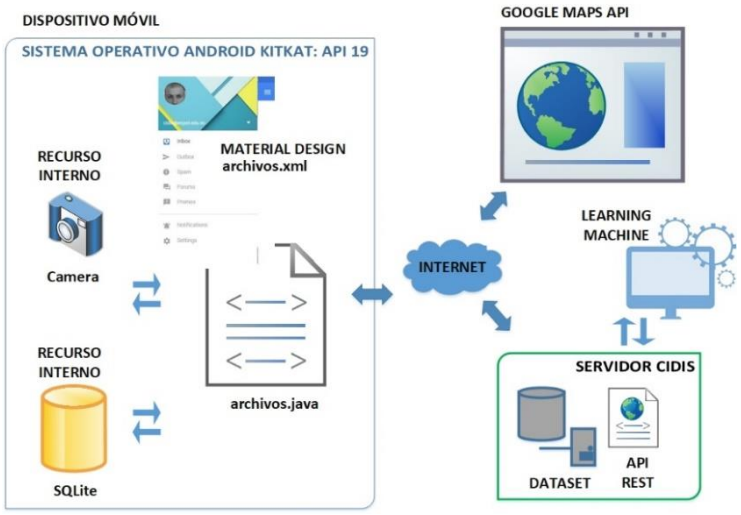


Figura 2.11: Estructura interior del módulo APP

En la Figura 2.11, el rectángulo a la izquierda representa la estructura interna de recursos en el dispositivo móvil, en el podemos apreciar una separación conceptual entre los archivos de diseño (archivos.xml) y los archivos de lógica de negocio (archivos.java). Además, con las flechas azules se representa la comunicación entre los comandos de programación del proyecto y los recursos internos del dispositivo como son: la cámara y la base de datos SQLite. Las pequeñas pantallas a la derecha representan los recursos externos de quienes se sirve la aplicación móvil para completar sus funcionalidades, como son: la API de *Google Maps* para gestionar la incrustación de *Google Maps* en la aplicación, y los recursos alojados en servidor del CIDIS tales como el *data set* y la API REST para el reconocimiento de imágenes.

2.3 Módulo WEB

En esta sección describiremos la metodología, el procedimiento, las herramientas y la arquitectura que han sido establecidas para el desarrollo de nuestra aplicación web montada en ambiente Linux bajo tecnología LAMP, con acceso en línea a través de dirección IP pública, y cuyas principales funcionalidades son: proveer al menos un perfil de usuario que deberá ser administrador el cual tendrá habilitada todas las opciones de la página web, gestión de tabla usuarios proveyendo las funcionalidades necesarias para consulta integra de tabla, inserciones, modificaciones, y eliminaciones lógicas de la misma, gestión de tabla "dataset" proveyendo las funcionalidades necesarias para consulta integra de tabla que incluye la presentación de las imágenes de especies a cuales se referencia en un campo de la tabla y la presentación de una porción de *Google Maps* donde se muestra la ubicación donde fue colectada la especie, inserciones, modificaciones, y eliminaciones lógicas de la misma, gestión de las tablas clasificación y parte proveyendo las funcionalidades necesarias para consulta integra de cada tabla, inserciones, modificaciones y eliminaciones lógicas respectivas, finalmente presentar las páginas web para usuario final con diseños actuales, amigables y que permiten una fácil interacción con el mismo. Para definir la solución de este módulo hemos usado Metodología de Diseño Hipermedia Orientada a Objetos (OOHDM), la cual es una metodología propicia para el desarrollo de sistemas que tienen interacción con usuario final debido a que está basada en etapas de diseño visuales del proyecto permitiendo de esta forma la interacción entre el equipo de desarrollo y el cliente. La idea detrás de esta metodología es primero involucrar al cliente con el producto que ha solicitado y la segunda intención es que el equipo de desarrollo obtenga retroalimentación de parte del cliente para que los cambios o nuevos rumbos en cuanto al proyecto puedan hacerse durante el tiempo de desarrollo del proyecto más no cuando esté terminado.

De la misma forma como fueron estructuradas las sub secciones en la sección anterior: Módulo APP, así se estructuran las sub secciones: 2.2.2 Procedimiento, 2.2.3 Herramientas y 2.2.4 Arquitectura dentro de este módulo.

2.3.1 Metodología de desarrollo OOHDM

La cantidad de aplicaciones de software hipermedia crece de forma acelerada hoy en día, además, se ha estado volviendo cada vez más complejas debido a su alcance y las tecnologías de vanguardia. Es por esto que los equipos de desarrollo de software en la actualidad se han visto necesitados de aplicar metodologías de desarrollo de software que sean sistemáticas, puesto que se necesita que cada fase del proyecto pueda ser estudiada, obtener retroalimentación de parte del usuario final y evolucionar en caso de ser necesario, también aprovechar estas fases de construcción para aplicar reusabilidad utilizando fases de un proyecto en otro.

La Metodología de Desarrollo Hipermedia Orientado a Objetos (OOHDM) cubre los requerimientos de un proceso de desarrollo sistemático puesto que es basado en otra metodología sistemática como HDM (Hypermedia Develop Method) pero con el valor agregado de habérsela orientado a objetos para contemplar abstracciones visuales - modelos del proyecto y la independencia de los elementos en sistemas hipermedia. OOHDM está compuesto por 5 etapas en total aunque la primera muchas veces no es tomada en consideración por ser fase de levantamiento de información. Sus 4 etapas siguientes y cuales son aplicadas a nuestro proyecto son: diseño conceptual, diseño de flujo de navegación, diseño de interfaces abstractas e implementación.

En la etapa de diseño conceptual se busca obtener una correcta abstracción semántica del dominio del proyecto valiéndose de diagramas de modelado estructurales comúnmente usados en orientación a objetos. Las metodologías tradicionales de ingeniería de software no han sabido realizar buenas tareas de abstracción que faciliten la tarea de especificar aplicaciones hipermedia, lo cual se procura corregir con esta fase. En la etapa de diseño de flujo de navegación se plantea un contexto de flujo de navegación, el cual es un espacio de estudio de un escenario específico

de ejecución del proyecto y está formado por nodos, enlaces, clases de contextos, y otros contextos de navegación, cuya principal tarea es organizar el espacio de flujo de navegación en conjuntos convenientes que puedan ser recorridos en un orden particular y que deberían ser definidos como caminos que le permitan al usuario conocer bien donde puede ir y como llegar al lugar deseado. En la etapa de diseño de interfaces abstractas se plasman las decisiones de diseño visual para el proyecto teniendo en consideración que deben ser tan independientes que los diseños debidos a la estructura visual para el flujo de navegación pueden ser desacoplados de aquellos debidos a la estructura visual para el modelo del dominio, pero trabajar de manera sinérgica al mismo tiempo. Finalmente en la etapa de implementación se debe especificar de manera detallada las decisiones en cuánto a aspectos técnicos en general como son: las tecnologías que serán usadas en cada fase del proyecto, las herramientas tecnológicas para la construcción del proyecto, las arquitecturas internas e incluso los procedimientos que se deberán seguir durante su desarrollo. [6]

2.3.1.1 Diseño conceptual

En esta fase se deben crear esquemas conceptuales que representen a los objetos del dominio, sus relaciones y subsistemas. Básicamente se representa un diseño conceptual de un proyecto a partir del modelado de datos semánticos estructurales, idéntico al modelado estructurado de datos que se realiza en UML. En OOHDM se construye este modelado estructurado con elementos de clases, relaciones y subsistemas predefinidos y que son similares a UML excepto tal vez por los atributos de las clases que pueden ser de múltiples tipos para representar perspectivas diferentes de los mismos objetos del mundo real.

Usualmente son usados recursos agregados a este modelado a medida que el sistema aumenta su complejidad. [6]

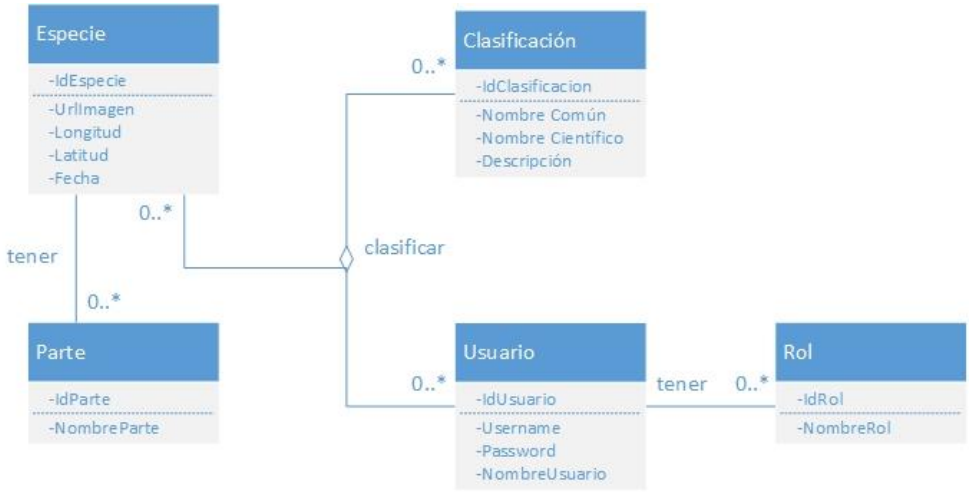


Figura 2.12: Diagrama de clases módulo WEB

La Figura 2.12 muestra un modelado conceptual en UML para el módulo WEB. Este es el modelo de negocio del proyecto y el pequeño rombo indica su relación principal clasificar, la cual es ternaria, e indica que una entidad Usuario puede clasificar una o muchas entidades Especie en una o muchas entidades Clasificación. Luego, una entidad Especie puede tener una o muchas entidades Parte y una entidad Usuario tiene una o muchas entidades Rol.

2.3.1.2 Diseño de flujo de navegación

En OOHDM, la navegación es considerada un proceso crítico en el diseño de aplicaciones. Un modelo de flujo de navegación debe presentar una o varias vistas del modelo conceptual, debe ser una instancia del modelo conceptual en tiempo de ejecución. Además se puede contemplar modelos diferentes de acuerdo al usuario cual sea el caso de estudio. Un Diseño de Flujo de Navegación puede ser expresado en dos tipos de esquemas: el esquema de clases para flujo de navegación y el esquema de contexto para el flujo de navegación. Para el modelamiento de nuestro proyecto nos hemos servido del esquema de contexto para el flujo de navegación. Para el esquema de clases, OOHDM provee un conjunto de tipos predefinidos de clases para flujo de navegación: nodos, enlaces y estructuras de acceso. En este

contexto el concepto de nodos y enlaces es el mismo de aquellos elementos que pertenecen a las metodologías tradicionales para modelamiento de aplicaciones hipermedia. [6]

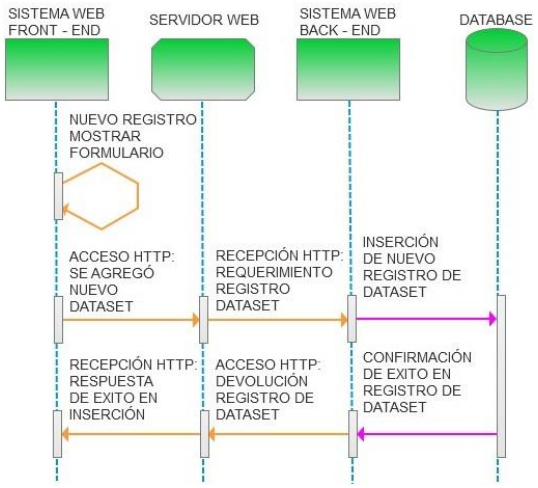


Figura 2.13: Flujo de navegación módulo WEB: funcionalidad registro Dataset

La Figura 2.13 muestra el flujo de actividades en la plataforma web para la funcionalidad de inserción de un registro en tabla *dataset*. Las figuras en color verde representan los entornos que participan. En el entorno SISTEMA WEB FRONT - END la primera actividad se representa con un flujo hacia sí mismo debido a que es una actividad que se realiza dentro de la plataforma: presentar formulario. Mientras que, la última actividad representa la actividad de envío – confirmación de registro de datos ingresados en FRONT – END hacia la tabla Dataset de la DATABASE, a través de su orden de ejecución al BACK – END, valiéndose de un SERVIDOR WEB.

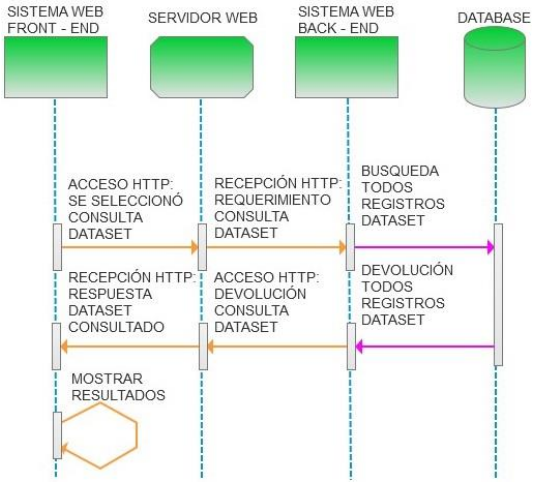


Figura 2.14: Flujo de navegación módulo WEB: funcionalidad consulta Dataset

La Figura 2.14 muestra el flujo de actividades en la plataforma web para la funcionalidad de consulta de registros de tabla *dataset*. Las figuras en color verde representan los entornos que participan. La primera actividad representa la actividad de solicitud – recepción de listado de datos en FRONT - END hacia la tabla Dataset de la DATABASE, a través de su orden de ejecución al BACK – END, valiéndose de un SERVIDOR WEB. La última actividad se representa con un flujo hacia sí mismo debido a que es una actividad que se realiza dentro de la plataforma: presentar resultado de consulta.

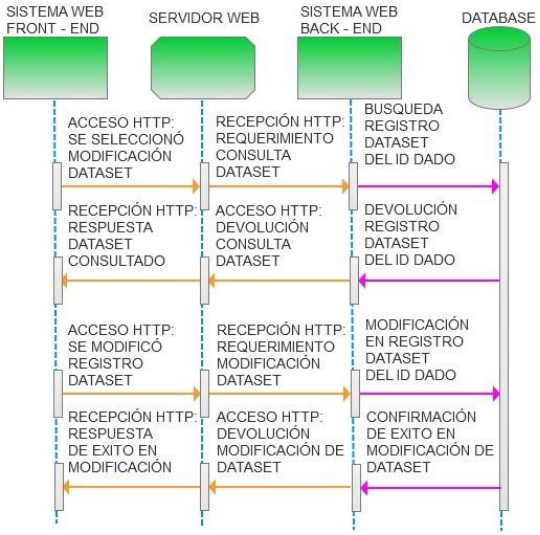


Figura 2.15: Flujo de navegación módulo WEB: funcionalidad modificación Dataset

La Figura 2.15 muestra el flujo de actividades en la plataforma web para la funcionalidad de modificación de un registro de tabla *dataset*. Las figuras en color verde representan los entornos que participan. La primera actividad representa la actividad de solicitud – recepción de datos del registro indicado por el Id en FRONT - END hacia la tabla Dataset de la DATABASE, a través de su orden de ejecución al BACK – END, valiéndose de un SERVIDOR WEB. Mientras que, la segunda actividad representa el envío – confirmación de modificación de los datos ingresados en FRONT – END hacia la tabla Dataset de la DATABASE, siguiendo el mismo proceso descrito anteriormente.

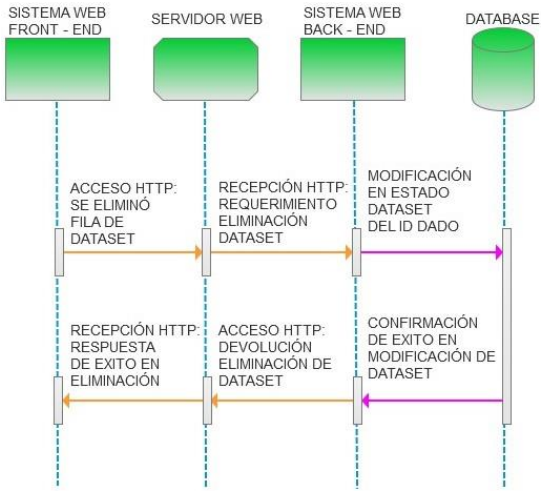


Figura 2.16: Flujo de navegación módulo WEB: funcionalidad eliminación Dataset

La Figura 2.16 muestra el flujo de actividades en la plataforma web para la funcionalidad de eliminación de un registro de tabla *dataset*. Las figuras en color verde representan los entornos que participan. Se representa la actividad de petición – confirmación de eliminado lógico del registro desde FRONT - END hacia la tabla Dataset de la DATABASE, a través de su orden de ejecución al BACK – END, valiéndose de un SERVIDOR WEB.

2.3.1.3 Diseño de interfaces abstractas

En esta fase deben ser definidas las estructuras visuales del proyecto: la forma en la cual aparecen los objetos del flujo de navegación, como los objetos de estas estructuras visuales activarán la navegación y a su vez como estos objetos activarán el resto de la funcionalidad de la aplicación. Si la fase de flujo de navegación y esta fase de diseño de interfaces abstractas están correctamente elaboradas para el proyecto y por ende existe una clara separación entre estas, entonces será posible construir diferentes interfaces visuales para un mismo modelo de flujo de navegación, lo cual permitirá que para el proyecto sea

transparentes las tecnologías de interfaz visual que sean implementadas. En esta fase generalmente son usados elementos para maquetado visual tales como: mock ups, wareframes, y prototipos en general. [6]

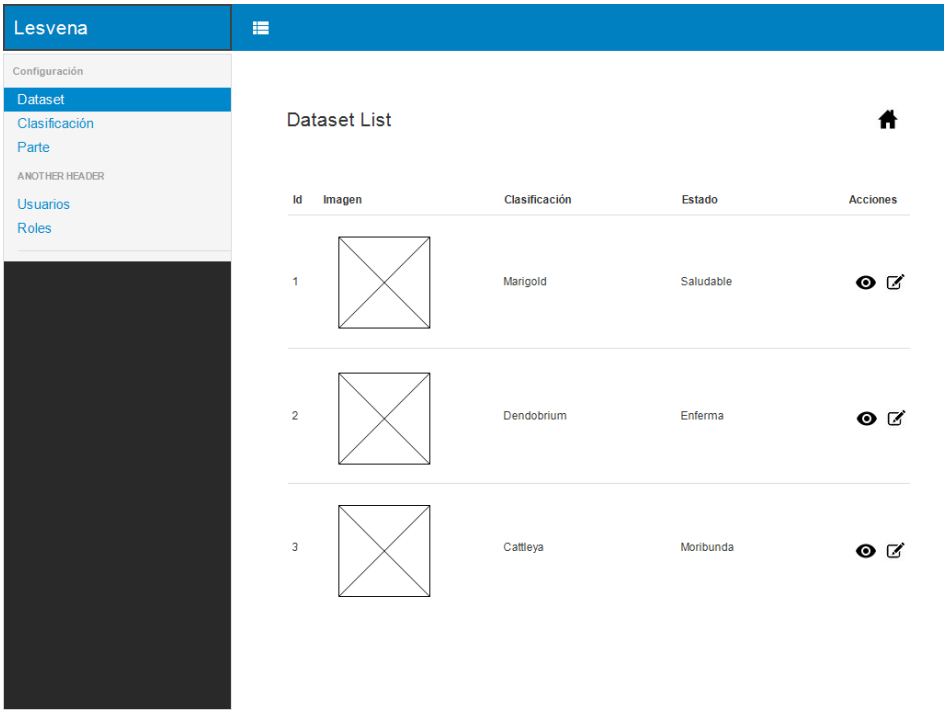


Figura 2.17: Prototipo pantallas página web: pantalla Dataset List

En la Figura 2.17 se presenta el prototipo para pantalla de listado de registros de la tabla Dataset. Esta pantalla muestra un diseño de listado de registros típico de páginas web contemporáneas, donde a cada registro se le incluyen dos botones para: visualización y edición de los datos del registro. Los campos de la tabla Dataset que son mostrados en el listado son: Imagen, Clasificación, Estado y Ubicación para cada especie. Además, si bien en el campo Imagen de la tabla Dataset se guarda el enlace con IP pública de la especie pero en este listado se muestra la foto cargada de la especie traída desde aquella dirección.

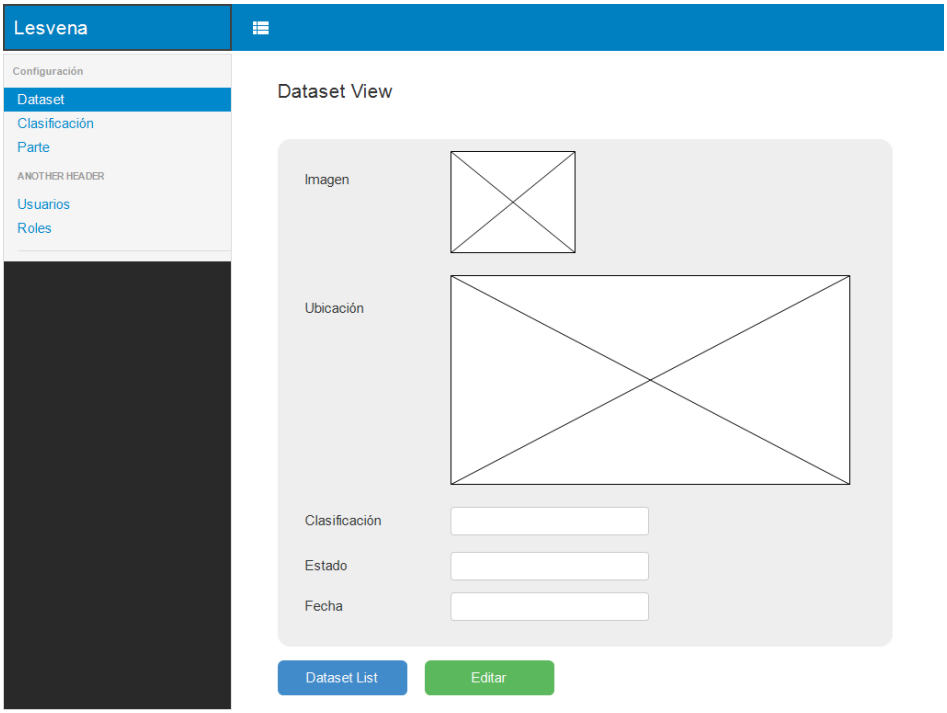


Figura 2.18: Prototipo pantallas página WEB: pantalla Dataset View

En la Figura 2.18 se presenta el prototipo para pantalla de consulta de la tabla Dataset. Esta pantalla muestra los campos de un solo registro específico de la tabla Dataset, el cuál ha sido seleccionado por el usuario, con una disposición extendida a lo largo del espacio de pantalla. Los campos mostrados son: Imagen, Clasificación, Especie, Fecha, y un gráfico de ubicación. Además, es de notar que el gráfico denominado Ubicación es una porción de Google Maps, en la cual se muestra un marcador de posición determinado por los valores de los campos Latitud y Longitud de la tabla Dataset. Finalmente, se agregan dos botones en la parte inferior: Dataset List el cual dirige hacia la página Listado Dataset y Edit el cual dirige hacia la página para edición de un registro de la tabla Dataset.

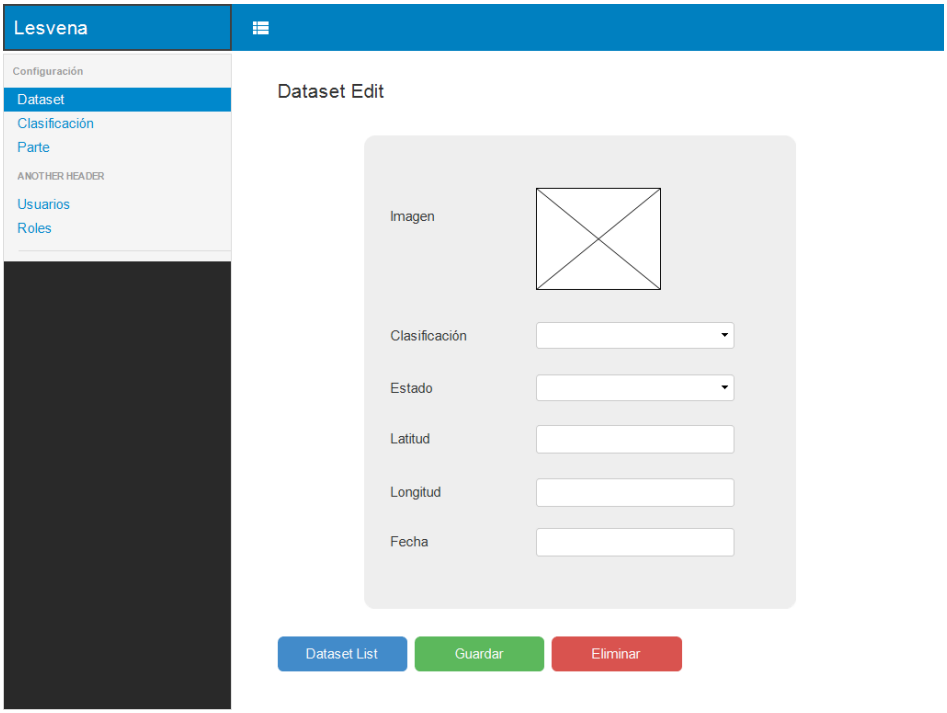


Figura 2.19: Prototipo pantallas página web: pantalla Dataset Edit

En la Figura 2.19 se presenta el prototipo para pantalla de edición de la tabla Dataset. Esta pantalla muestra los campos de un solo registro específico de la tabla Dataset, el cuál ha sido seleccionado por el usuario, con una disposición extendida a lo largo del espacio de pantalla, pero a diferencia de la página Dataset View en esta página no hay el gráfico de Google Maps para la ubicación de la especie sino los campos de la tabla tal cual están en la base. Los campos presentados son: Clasificación, Especie, Latitud, Longitud y Fecha. Finalmente, se agregan tres botones en la parte inferior: Dataset List el cual dirige hacia la página de listado de registros de Dataset, Guardar el cual graba los cambios realizados a los datos de los campos y Eliminar el cual hace un borrado lógico del registro y dirige hacia la página de inicio.

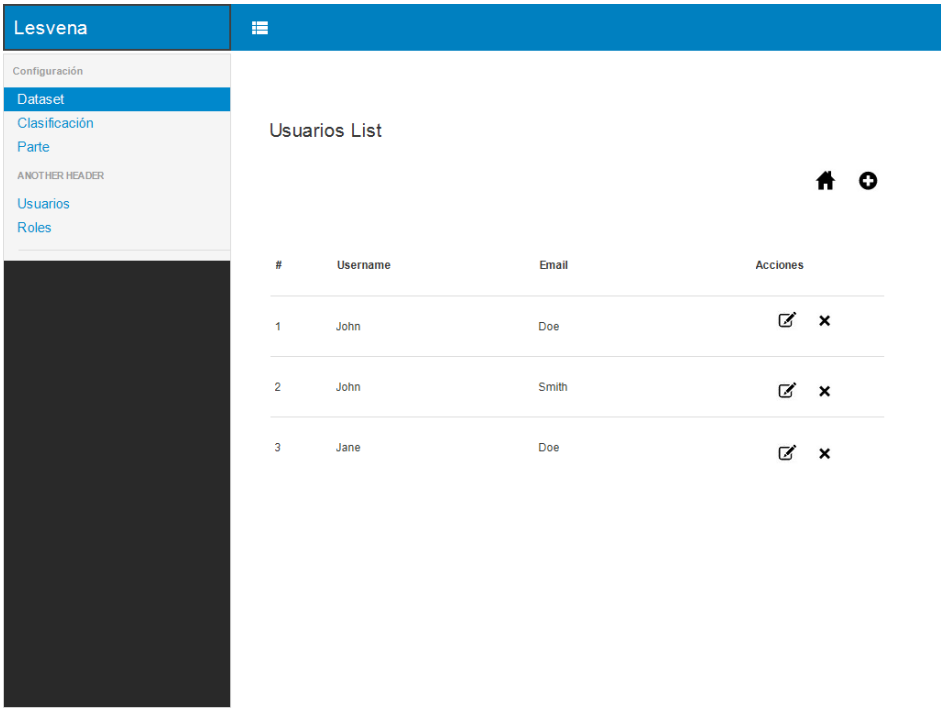


Figura 2.20: Prototipo pantallas página web: pantalla Usuarios List

En la Figura 2.20 se presenta el prototipo para pantalla de listado de registros de la tabla Usuario. Esta pantalla muestra un diseño de listado de registros típico de páginas web contemporáneas, donde a cada registro se le incluyen dos botones para: edición y eliminación de los datos del registro. Todos los campos de la tabla Usuario son mostrados en el listado, excepto Password. Este diseño de listado (sin figura de la especie) es el mismo usado en las tabas Clasificación, Parte y Rol. Además, en este diseño se incluyen los botones: Home (ícono de casa), que dirige hacia la página de inicio y Add Register (ícono de suma), que dirige hacia la página para inserción de un nuevo registro a la respectiva tabla.

2.3.1.4 Implementación

En esta fase el diseñador debe especificar los elementos y procedimientos para lograr implementar el diseño que ha plasmado en las fases anteriores. Se debe especificar un entorno particular en el cual va a correr la aplicación y en base a este entorno se deberán además especificar con detalle: las herramientas tecnologías para la implementación del proyecto, la arquitectura interna que tendrá, y el procedimiento que se debe seguir durante el desarrollo del proyecto hasta su puesta en marcha. [6]

2.3.2 Procedimiento

Antes de escoger las tecnologías que serán usadas en el desarrollo de nuestro sistema web tuvimos que considerar las circunstancias, las demandas tanto de rendimiento como funcionalidad entre otras, para la aplicación, y el entorno en general.

El sistema en general debía ser de tipo administrativo porque sólo soportará funcionalidades de CRUDS para las tablas del módulo DATASET, además de funcionalidades para manejo de autenticación por usuario. Otro factor influyente en la selección de las tecnologías para nuestro sistema fue el corto tiempo del que disponíamos para el desarrollo.

Teníamos conocimiento que en nuestro entorno al momento las tres mejores tecnologías en el mercado para desarrollo web son: .NET, Java y PHP. Cualquiera de las tres tecnologías podía cumplir con las demandas para el sistema pero ninguno de los dos compañeros teníamos conocimiento de .NET en tanto que los dos dominábamos tanto Java como PHP, así que descartamos la tecnología .NET por el costo que hubiésemos tenido para aprenderla.

Las opciones de tecnología que quedaban fueron Java y PHP. Java tiene algunas ventajas como robustez y fiabilidad pero nuestro sistema no tendría altas demandas de robustez, ni alto tráfico de usuarios, en cambio sí necesitaba ser rápido, libre y versátil, características que en Java se consideran como desventajas debido a que sus tiempos para compilado retardan tanto el desarrollo como ejecución en cliente. Pero estas necesidades que no son características de Java son en cambio ventajas conocidas de la tecnología PHP. Por ende terminamos por decidirnos por desarrollar nuestro sistema web con las tecnologías de PHP.

Finalmente, la tecnología PHP tiene multitud de módulos, soportes y herramientas como *frameworks* que permiten simplificar tareas. De esta forma escogimos el *framework Symfony* para el desarrollo. Entre las ventajas que encontramos para trabajar con este *framework* en lugar de desarrollar con código nativo están: El proyecto queda ordenado según la estructura del *framework*, permitiendo un mantenimiento más fácil y rápido, lo cual permite una mejor continuidad del proyecto para desarrollos futuros; otra ventaja al usar este *framework* es que permite hacer reutilización de librerías para tareas comunes, actualización de la plataforma a las nuevas tecnologías o ayuda en el desarrollo por la comunidad de desarrolladores.

2.3.3 Herramientas

De acuerdo a la tecnología escogida para el desarrollo de nuestro sistema web, que fue descrita en la sección Procedimiento para este módulo, las herramientas que proyectamos utilizar se listan a continuación:

- PHP 5 (se instala junto al ambiente LAMP, en módulo DATASET)
- *Framework Sympony*
- Composer
- SubLime Text 3
- HTML5
- Postman

2.3.4 Arquitectura

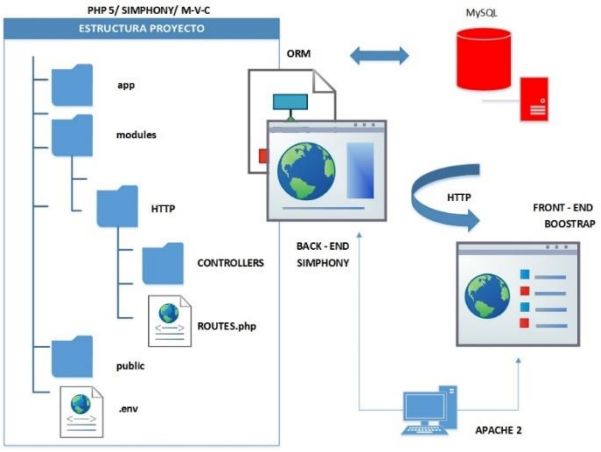


Figura 2.24: Estructura interior del módulo WEB

El recuadro con banda azul superior representa la estructura de carpetas MVC de nuestro proyecto que está desarrollado en *framework Symfony*. Las pequeñas pantallas con símbolo de planeta representan las secciones *Back – End* y *Front – End* según indica su texto. La flecha curva indica el protocolo Http que es usado para comunicar las dos secciones, teniendo como soporte al servidor web *Apache* 2 representado por la figura del ordenador debajo de las pantallas.

2.4 Módulo DATASET

Básicamente es la matriz del proyecto debido a que: Este módulo abarca nuestro servidor como espacio físico de almacenamiento, contiene a la base de datos que está formada por la información recabada de las especies, además de interactuar con todos los demás módulos de nuestro sistema.

2.4.1 Metodología para el desarrollo del módulo

En esta sección describiremos la metodología que fue establecida para el desarrollo de nuestro módulo DATASET que fue separado en tres sub módulos: LOAD, DATABASE, REST.

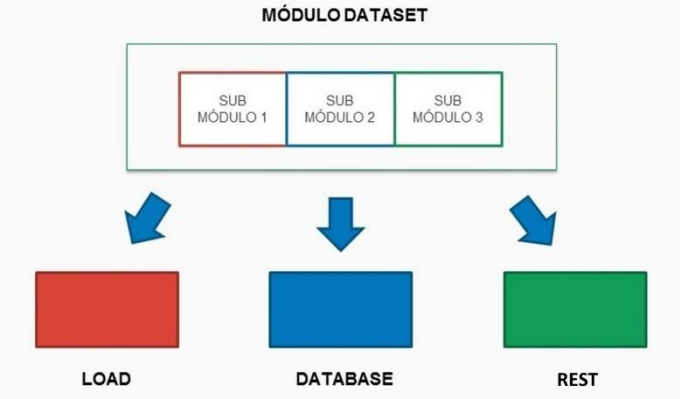


Figura 2.25: División en sub módulos del módulo DATASET

El rectángulo más grande de la parte de arriba representa al módulo DATASET en su totalidad, que a la vez es nuestro servidor con IP pública. Los rectángulos en colores representan las secciones conceptuales en que dividimos a este módulo: LOAD que refiere al levantamiento inicial de fotografías, DATABASE que refiere a la estructura de datos y, REST que refiere a la interacción con un servidor para reconocimiento de imágenes.

2.4.2 Sub módulo LOAD

2.4.2.1 Metodología

La metodología que se sigue en este sub módulo es enlistar las tareas específicas que se deben realizar y en base a estas especificar los procedimientos, herramientas y arquitectura necesarios para su implementación.

2.4.2.2 Procedimiento

Se tuvo que gestionar una carga inicial de datos antes de poner a disposición de los usuarios a nuestra aplicación móvil debido a que en el modo consulta de nuestra aplicación se hace una comparación de las fotografías capturadas por el usuario con fotografías almacenadas en nuestra base de datos, a manera de referencia para el usuario colaborador, surgió por tanto la

necesidad de un conjunto de datos iniciales en nuestra base contra quienes hacer la comparación.
Esta tarea de carga de un conjunto inicial de datos de especies vegetales nativas se ideó para que sea hecha de forma manual debido a que nuestras herramientas tanto administrativa (WEB) como de recolección (APP) entrarían a funcionar después de que estos datos estuviesen cargados en nuestra base de datos.

Inicialmente se planteó gestionar una recolección, tanto de fotografías como de datos del nombre de la especie, su ubicación por GPS y estado de conservación, para tres tipos de especies y en una cantidad de 150 fotografías de individuos por cada especie, dando un total de 450 fotografías de individuos lo necesitado.
Luego, sabiendo el tipo y la cantidad de información que se debía recolectar, se planteó hacer la recolección en el Campus Prosperina de la ESPOL. De esta forma nos dirigimos en un principio hacia los invernaderos ubicados en la actual Facultad de Ciencias de La Vida y el CIBE para revisar especies disponibles pero pudimos notar que no existía suficiente variedad de especies, en la ESPOL en general, así que pudimos proyectar que una recolección en ese lugar nos tomaría mucho mayor tiempo del programado.

Entre las alternativas de lugares con especies vegetales nativas que habíamos investigado estaban ESPOL, viveros de Vía a la Costa saliendo de la ciudad de Guayaquil, viveros en la Vía a Yaguachi cerca de la ciudad de Milagro. Jardín Botánico de Guayaquil, Cerro Blanco en la ciudad de Guayaquil, entre otros. Habiendo descartado la ESPOL como lugar de recolección tuvimos que programar viajes a todos los otros viveros mencionados.

2.4.2.3 Herramientas

De acuerdo a las tecnologías escogidas para la implementación de este sub módulo, que fue descrita en la sección Procedimiento, las herramientas que proyectamos utilizar se listan a continuación:

- Cámara fotográfica EOS 70 D
- Micro SDHC 16 GB
- Computador personal – Windows 10

2.4.2.4 Arquitectura

De acuerdo a los requerimientos establecidos y recursos disponibles para la implementación del sub módulo LOAD, que fue descrita en la sección Procedimiento, la arquitectura que proyectamos utilizar se representa mediante el gráfico:

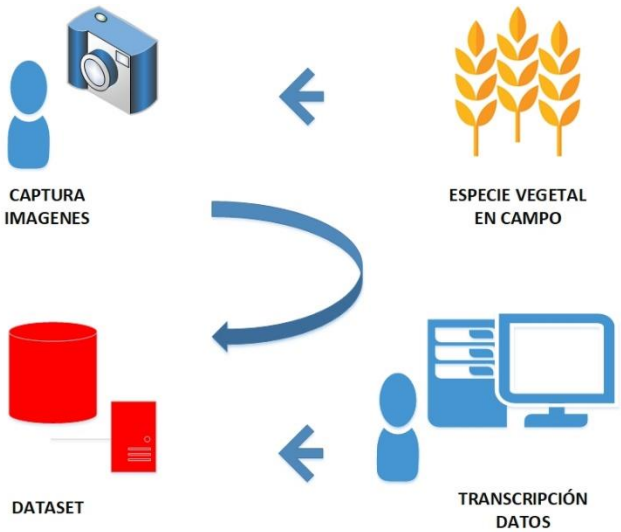


Figura 2.26: Arquitectura levantamiento *data set*

En la Figura 2.26 se maquetan dos procesos en la alimentación con datos a nuestro sistema de base de datos. La flecha azul superior indica el proceso de capturas de fotos de las especies vegetales, en campo; mientras que la flecha azul inferior indica el proceso de digitar y transcribir los datos de fotografías al DATASET. La flecha azul curva indica el traspaso de información de la cámara digital al ordenador.

2.4.3 Sub módulo DATABASE

2.4.3.1 Metodología

La metodología que se sigue en este sub módulo es enlistar las tareas específicas que se deben realizar y en base a estas especificar los procedimientos, herramientas y arquitectura necesarios para su implementación.

2.4.3.2 Procedimiento

Para el almacenamiento de los datos de las especies vegetales nativas, necesitábamos de una estructura física para almacenamiento de datos informáticos que sea escalable, soportando posibles incrementos posteriores tanto de datos como de conceptos, que permita una relativa rapidez y facilidad de acceso a la información y que represente la información de manera estructurada tal cual son los conceptos de especies vegetales en la realidad.

Basados en estos requerimientos escogimos un sistema de base de datos relacional para albergar la información de nuestras especies vegetales puesto que una estructura de datos de este tipo cumple con todas estas especificaciones y muchas otras como seguridades e integridad de los datos, etc. A la fecha existen motores de bases de datos con costos de licencia anuales para su uso, como Oracle o Microsoft SQL Server, pero también hay motores de bases de datos sin costo de licencia,

como MySQL o PostgreSQL. Nos inclinamos por el uso de un motor de base de datos sin costo de licencia porque no tenemos grandes demandas de datos, seguridades ni soporte.

Otro factor muy importante a considerar fue el lugar de alojamiento para nuestro sistema de base de datos. Debido a que este sistema es parte de un proyecto para investigación se consideró que debe tener un sitio digital dedicado, accesible desde muchos sitios y con suficiente espacio para su posterior crecimiento. Siendo ese el caso concluimos en establecer un servidor con acceso de internet a través de IP pública para el alojamiento de nuestro sistema de bases de datos.

Tuvimos dos alternativas luego de consultar con nuestro tutor: alquilar un sitio Host durante unos meses, o tomar un espacio en un servidor proporcionado por el CIDIS en nuestra institución. Escogimos la segunda opción para ahorrar costos y cualquier necesidad de traslado de sitio o integraciones posteriores.
Como sistema operativo para nuestro servidor fue escogida la plataforma Linux, con Ubuntu versión 14.04, esto debido a dos condiciones: Capa de seguridad adicional en el acceso al sistema de bases de datos y capacidad de sincronización entre las tecnologías PHP y Python que debían funcionar simultáneamente para el sistema web y el programa .bash de comunicación con el servidor externo para reconocimiento de imágenes respectivamente.

Puesto que debíamos trabajar con un motor de base de datos sin costo de licencia, ya habíamos definido a Ubuntu como el sistema operativo a funcionar en nuestro servidor , sabíamos que debíamos trabajar con lenguaje PHP y servidor web Apache para nuestro sistema web, entonces escogimos a MySQL como

nuestro motor de bases de datos para terminar de engranar las piezas de un sistema LAMP (Linux, Apache, MySQL, PHP) que típicamente es brindado a la fecha en el mercado como solución que trabaja de forma sinérgica para ambientes Linux con tecnología PHP.

2.4.3.3 Herramientas

De acuerdo a las tecnologías escogidas para la implementación de este sub módulo, que fue descrita en la sección Procedimiento, las herramientas que proyectamos utilizar se listan a continuación:

- Servicio de Host con acceso público (.com)
- LAMP: Ambiente dispuesto para servicios eb basado en las tecnologías Linux con Ubuntu 14.04, Apache 2, MySQL, y PHP 7.
- PHPMyAdmin, para administrar la base de datos.
- Modelo de Base: Entidad – Relación

2.4.3.4 Arquitectura

De acuerdo a los requerimientos establecidos y recursos disponibles para la implementación del sub módulo DATABASE, que fue descrita en la sección Procedimiento, la arquitectura que proyectamos utilizar se representa mediante el gráfico:

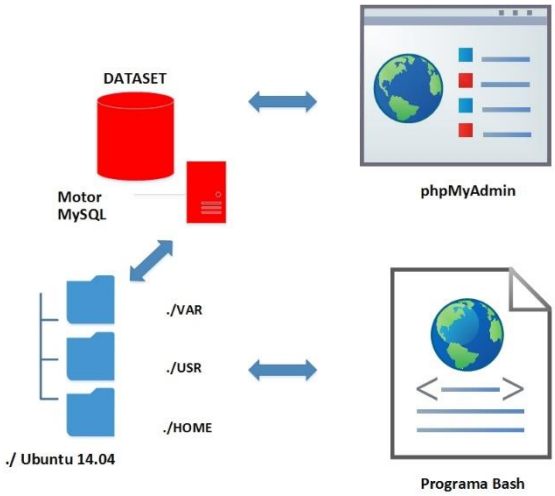


Figura 2.27: Arquitectura interna sub módulo DATABASE

Se maqueta la estructura interior en Linux para nuestro módulo DATASET. Podemos notar que el módulo se compone de dos partes: El conjunto de datos estructurados en la base MySQL y administrados a través de PHPMyAdmin; y el programa BASH para la comunicación entre los datos de la aplicación móvil y el servidor de reconocimiento de imágenes.

2.4.4 Sub módulo REST

2.4.4.1 Metodología

La metodología que se sigue en este sub módulo es enlistar las tareas específicas que se deben realizar y en base a estas especificar los procedimientos, herramientas y arquitectura necesarios para su implementación.

2.4.4.2 Procedimiento

A pesar de que ya poseíamos interface visual (en *Bootstrap*) y sistema web (en *Symfony*) para gestionar los datos del DATASET pero estos datos podían ser gestionados de manera manual a través de la interface visual, más no podían ser gestionados de manera remota desde nuestra aplicación móvil,

de allí partió la necesidad de poseer una interface para la gestión de datos y comandos remotos desde nuestra aplicación móvil, además de la disponibilidad para trabajar con la máquina de reconocimiento de imágenes.

Debido a que la comunicación entre la APP y nuestra API debía establecerse a través del internet, aprovechamos nuestro servidor web Apache que sirve para el sistema web para que sirva también para la disponibilidad de esta API, a través del protocolo HTTP.
Sabiendo que debía permitirse una comunicación remota entre nuestra APP y esta API, también que esta comunicación debía hacerse a través del internet con protocolo HTTP, tuvimos dos opciones de esquema para la implementación de esta comunicación: SOAP y REST, que son las dos tecnologías que permitirían hacer estas tareas, a la actualidad.

En SOAP, se pone el énfasis en la diversidad de operaciones del protocolo, o verbos; por ejemplo una aplicación SOAP podría definir operaciones como: getUser(); addUser(). En cambio en REST el énfasis se pone en los recursos, o sustantivos. Por ejemplo, una aplicación REST podría definir tipos de recursos asignándoles nombres como: Usuario {}.
Este enfoque hacia el recurso que tiene una analogía al concepto de objetos al igual que el lenguaje Java usado en nuestra APP, además de ventajas como: Separación entre el cliente (APP) y el servidor (API); una API REST siempre es independiente del tipo de plataformas o lenguajes (Java en APP y PHP en servidor, en nuestro caso), nos llevó a decidirnos por implementar nuestra API con esquema REST.

En resumen, esta API consta de comandos en lenguaje PHP versión 7 para gestionar la comunicación entre los participantes, además, está alojada en el mismo ambiente LAMP del módulo DATASET, dentro del servidor web Apache levantado para la gestión de transacciones vía HTTP, con estructura de comunicación web REST.

2.4.4.3 Herramientas

De acuerdo a las tecnologías escogidas para la implementación de este sub módulo, que fue descrita en la sección Procedimiento, las herramientas que proyectamos utilizar se listan a continuación:

- PHP 7
- SubLime 3 Text
- Git (.bash)
- Cuenta en BitBucket
- LAMP (montado sobre host provisto por CIDIS)
- Postman
- Tener disponible el archivo.py provisto por CIDIS que ejecuta comandos de forma remota a la máquina de aprendizaje automático.

2.4.4.4 Arquitectura

De acuerdo a los requerimientos establecidos y recursos disponibles para la implementación del sub módulo REST, que fue descrita en la sección Procedimiento, la arquitectura que proyectamos utilizar se representa mediante el gráfico:

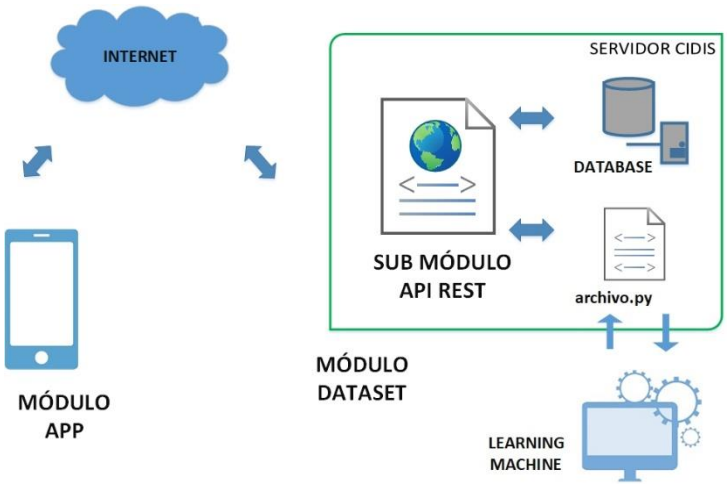


Figura 2.29: Arquitectura interacción sub módulo REST

Se maquetan los componentes del sistema que interactúan con nuestro API REST. La figura del teléfono (MÓDULO APP) representa la aplicación móvil cuando interactúa con la DATABASE o la LEARNING MACHINE a través de la interface del API REST representada por la hoja web (SUB MÓDULO API REST). El rectángulo grande representa el servidor que contiene a todo el módulo DATASET y las flechas azules representan las interacciones internas en el módulo DATASET, desde el sub módulo API REST, tanto con la DATABASE como con la LEARNIG MACHINE, a través esta última de un archivo.py “embebido” en nuestro servidor.

CAPÍTULO 3

3. IMPLEMENTACION DE LA SOLUCIÓN.

En este capítulo describiremos la implementación de la solución según la metodología que fue detallada a manera de secciones para cada módulo del proyecto, en el Capítulo 2.

3.1 Implementación del módulo APP

En esta sección describiremos los detalles de la implementación de la solución para el módulo APP. Se ha considerado dar detalle sobre los aspectos que fueron críticos durante la implementación de este módulo como por ejemplo: las configuraciones para establecer enteramente el entorno para desarrollo; las configuraciones para la obtención de recursos internos del dispositivo como la cámara o el almacenamiento de datos en base *SQLite*; los códigos para gestiones de comunicación a través de protocolo HTTP; las gestiones para integración de diseño de menú y diseño de pestañas en una sola interfaz visual. Finalmente, indicamos el diseño final de interfaz de usuario que quedó establecido en el proyecto, a pesar que en la metodología del capítulo dos se estableció un diseño diferente.

3.1.1 Configuraciones de entorno para el desarrollo de la aplicación móvil

De acuerdo a las tecnologías escogidas para la implementación de este módulo, las tareas necesarias para desplegar tanto el entorno de desarrollo como las herramientas tecnológicas, que hacen viable el desarrollo del proyecto, se listan a continuación:

Descarga del IDE. Nos dirigimos a la página web oficial de *Android* versión *Developers* provista por *Google* a la fecha, y dentro de esta nos dirigimos a la sección *Download* y visualizamos una imagen grande junto con un botón “Descarga” alusivos al entorno de desarrollo *Android Studio*. Damos clic en el botón, aceptamos los términos e inmediatamente

descargamos el archivo ejecutable del IDE. El proyecto se desarrolló con la versión 3.0.1 del IDE para *Windows*.

Instalación de *Android Studio*. Para la instalación del IDE en *Windows* se ejecuta el archivo de extensión ".exe" descargado y luego toda la instalación es gestionada y guiada a través de pantallas tipo *wizard,* de donde se deben escoger: los paquetes adicionales a instalar (como el SDK), el directorio para la instalación del programa, aceptar los términos de uso, y aceptar (o modificar) el resumen final de instalación.

Creación del proyecto en *Android Studio*. Una vez instalado el programa ya podemos crear nuestro proyecto base sobre el cuál se van agregando las pantallas (*Activity*), archivos para programación (".java") y toda dependencia necesaria para su ejecución. Ejecutamos el programa *Android Studio* y clic en pestaña "Nuevo" y luego en "Proyecto", y entonces nos aparece un conjunto de pantallas tipo *wizard* que gestionan la creación del proyecto junto con sus dependencias. Durante la creación del proyecto se especifican tres datos importantes: 1.- Tipo de dispositivo final, donde se ejecutará nuestra aplicación, que pueden ser: teléfono móvil, televisor, o electrodoméstico con sistema operativo *Android*. 2.- Mínimo SDK, que indica la mínima versión de sistema operativo *Android* sobre la cual deberá correr nuestra aplicación, en nuestro caso fue escogida la API 19. 3.- Tipo de *Activity* matriz para el proyecto.

Configuraciones iniciales del proyecto. Debido a que nuestro proyecto es desarrollado sobre lenguaje *Java*, uno de los requisitos básicos para ejecutar *Android Studio* en *Windows* es la *Java Virtual Machine*, así que debemos tener instalado una versión actualizada de JDK en nuestro ordenador. A pesar de tener instalada la JDK, *Android Studio* podría presentar complicaciones en algunos sistemas (*Windows*) si este no sabe el directorio donde está instalado el JDK de *Java*, por este motivo es recomendable realizar una configuración adicional antes de ejecutar

nuestro proyecto, lo cual es configurar la variable de entorno *Java* para el sistema. El proceso es sencillo y fácil de encontrar en internet.

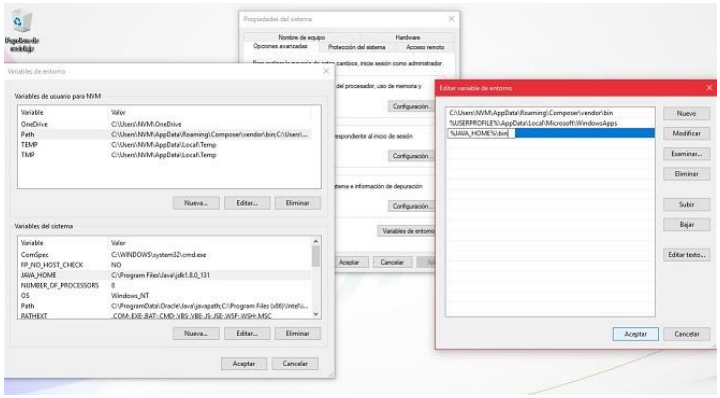


Figura 3.1: Captura pantalla de configuraciones variable de entorno

La Figura 3.1 muestra una captura de pantalla de la configuración de la variable de sistema “JAVA_HOME” en *Windows*. La pantalla del centro es la ventana “Propiedades del sistema”, la pantalla de la izquierda es la ventana “Variables de entorno”, en la cual creamos la variable de sistema “JAVA_HOME”. Luego, escogemos la variable “PATH” y damos clic en botón editar y nos aparece la pantalla de la derecha “Editar variable de entorno”, y agregamos la variable “JAVA_HOME” creada.

Configuraciones de dependencias del proyecto. Un programa desarrollado para ejecutarse en dispositivos móviles hace uso de varias dependencias como vimos en el capítulo 2. Pero *Android Studio* ha facilitado la tarea de integrar manualmente las dependencias al proyecto a través de proveer un administrador de dependencias llamado *Gradle.* Luego de crear nuestro proyecto, el administrador *Gradle*, quien funciona de forma automática y simultánea, nos indica las dependencias necesarias para la ejecución del proyecto. *Gradle* se encarga de las instalaciones y nosotros solo debemos autorizar los requerimientos que son presentados a través de interfaces amigables e intuitivas.

Entorno virtual para pruebas del proyecto. Cuando debemos probar funcionalidades completas del proyecto es recomendable ejecutarlo en algún dispositivo móvil compatible, pero durante el tiempo de desarrollo es recomendable hacer ejecuciones del proyecto en un entorno virtual (*Android Virtual Device* ó AVD) debido a la versatilidad de opciones de entornos emulados tanto en versiones de dispositivos móviles como en versiones de API de *Android*. Es así que para completar el entorno de desarrollo para un proyecto de aplicación móvil es indispensable contar con uno o varios entornos virtuales para probar su ejecución. Para esto debemos realizar dos pasos: 1.- Crear entorno virtual *Android*: nos dirigimos a la pestaña "*Tools"*, "*Android"*, clic en "AVD *Manager"*, y se presenta una ventana con pantallas tipo *wizard*, donde se debe seleccionar el tipo de dispositivo móvil a emular; la versión del sistema operativo *Android*; y además podemos configurar ciertos recursos del emulador como la cámara, la memoria asignada, etc. 2.- Ejecutar proyecto en AVD: damos clic en "Ejecutar" (botón flecha verde) y nos aparece una ventana para escoger el entorno virtual para ejecutar nuestro proyecto, donde, aquí aparecen los entornos virtuales creados en paso previo, seleccionamos uno de ellos y clic en "OK", y se presenta nuestra aplicación en el dispositivo emulado.

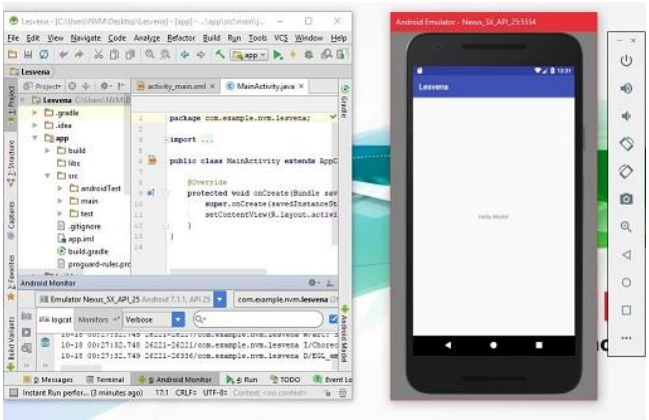


Figura 3.2: Captura pantalla de ejecución de proyecto en AVD

La Figura 3.2 muestra una captura de pantalla de una ejecución de nuestro proyecto en el “Entorno Virtual Android” (*Android Virtual Device*) que hemos creado mediante la pantalla AVD *Manager*. Podemos tener varios AVD pero sólo podemos ejecutar nuestro proyecto en uno de ellos a la vez.

3.1.2 Configuraciones para inclusión de una base de datos *SQLite* al proyecto

Para esta aplicación tuvimos la necesidad de contar con un esquema de base de datos de tipo relacional, debido a que este sistema está pensado para el almacenamiento interno de datos en el dispositivo móvil mientras se realizan las gestiones de sincronización. Para responder a esta necesidad se gestionó la creación, carga y administración de la base a través de *SQLite*, haciendo uso de un procedimiento conocido como: “Copia de base de datos externa a nuestra aplicación”; y, cuyos pasos se describen a breves rasgos a continuación: [7]

Paso 1: crear la base de datos con una herramienta externa, para nuestro caso hemos utilizado la herramienta *DB Browser for SQLite*.

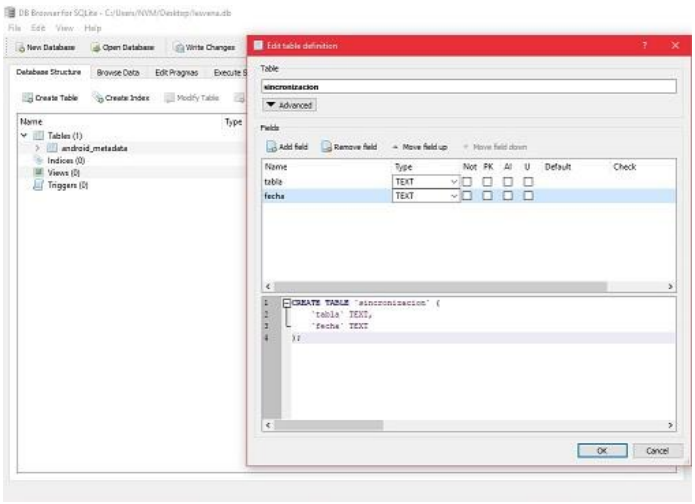


Figura 3.3: Creación de base con *DB Browser for SQLite*

La Figura 3.3 muestra una captura de pantalla del programa libre de licencia *DB Browser for SQLite*, en el momento de la creación de la base de datos Dataset, que es comprimida en el archivo lesvena.db.

Paso 2: copiar la base de datos en el directorio *assets* de nuestro proyecto.

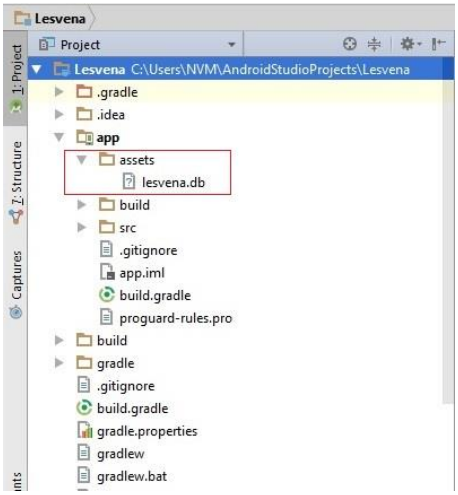


Figura 3.4: Archivo lesvena.db dentro de nuestro proyecto

La Figura 3.4 muestra una captura de pantalla de los directorios de nuestro proyecto, donde se aprecia la carpeta *assets* cuyo contenido es el archivo de base de datos de tipo *SQLite*: lesvena.db. Este archivo será cargado al aplicativo móvil desde ese directorio, mediante código.

Paso 3: inicializar la base de datos y copiar la nuestra.

```
public class AdminSQLiteOpenHelper extends SQLiteOpenHelper {
        public static final int DATABASE_VERSION = 1;
        public static final String DATABASE_NAME = "lesvena";

        private Context mContext;
        public AdminSQLiteOpenHelper(Context context) {...}

        public void createDataBase()  {
            File pathFile = mContext.
                    getDatabasePath(DATABASE_NAME);
            boolean dbExist = checkDataBase(
                    pathFile.getAbsolutePath());
            if(!dbExist) {
                this.getReadableDatabase();
                try {
                    copyDataBase(pathFile);
                } catch (IOException e) {
                }
            }
        }

        /* ... */
```

Figura 3.5: *Script* para copia de la base de datos

La Figura 3.5 muestra una captura de pantalla del *script* principal que: obtiene el archivo desde la carpeta *assets*; hace un *check* para verificar la existencia de alguna otra base de datos; y finalmente copia el archivo desde la carpeta *assets* hacia el aplicativo móvil.

Paso 4: referenciar dentro de nuestro código al *script* para copia de la base de datos.

```
public void createLesvenaDataBase() {
    AdmSQLiteOpenHelper admin;

    try {
        admin = new AdmSQLiteOpenHelper( context: this);
        admin.createDataBase();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        Toast.makeText( context: this,  text: "ERR-createBAse",
                Toast.LENGTH_SHORT).show();
        System.out.println("ERR-createBAse"
                + e.getMessage());
    }
}
```

Figura 3.6: *Script* para llamada a código de copia de base

La Figura 3.6 muestra una captura de pantalla del código en nuestro proyecto para ejecutar los *script* de copia de la base de datos.

3.1.3 Configuraciones para obtención de recurso cámara del dispositivo móvil desde la aplicación

Debido a que la captura de fotografías es una funcionalidad básica para el funcionamiento de nuestra aplicación tuvimos que agregar una etiqueta en archivo manifiesto de nuestra aplicación que accionará una validación que permitirá que sólo los dispositivos que posean cámara puedan descargarla desde *Google Play*.

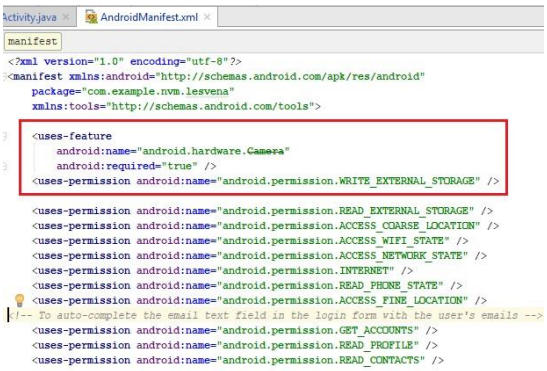


Figura 3.7: Etiquetas en archivo AndroidManifest.xml

La Figura 3.7 muestra una captura de pantalla de una porción de código en el archivo de configuración AndroidManifest.xml de nuestro proyecto. En este archivo se especifican los permisos que deseamos darle a muestra aplicación para acceder a recursos internos del dispositivo móvil, específicamente se establece permisos de cámara y escritura para el uso de recurso cámara.

La forma de delegar acciones a otros recursos del dispositivo, en *Android*, es invocar a un elemento *Intent*, el cual debe ser descriptivo sobre la tarea que va a desarrollar. Este proceso involucra 3 partes: El elemento *Intent* en sí mismo, una petición a la actividad externa requerida y, un código para gestionar la imagen cuando sea devuelta por el recurso cámara del dispositivo. [8]

La siguiente función muestra, de manera general, el código para gestionar el proceso de llamada al recurso cámara del dispositivo:

```
static final int REQUEST_IMAGE_CAPTURE = 1;

public VisualizeRFragment() {
    // Required empty public constructor
}

private void aceptarFoto() {
    takePictureIntent = new Intent(
            MediaStore.ACTION_IMAGE_CAPTURE);
    startActivityForResult(takePictureIntent,
            REQUEST_IMAGE_CAPTURE);
}
```

Figura 3.8: *Script* para gestión de recurso interno cámara

La Figura 3.8 muestra una captura de pantalla del código en nuestro proyecto para acceder al recurso interno cámara del dispositivo móvil. *Intent* es el nombre de la clase que representa recursos internos del dispositivo.

3.1.4 Configuraciones para incrustación de *Google Maps* al proyecto

Nuestro sistema implementa la interfaz de *Google Maps*, en su menú principal opción “Ver Mapa”, para visualizar las coordenadas GPS del sitio donde se realiza la recopilación de datos de especies. Debido a la estructura de interface de usuario de nuestro proyecto, el “Fragmento *Maps*” se incrustó dentro de un *Fragment* en lugar de ser incrustado, como comúnmente se lo hace, dentro de un *Activity*. [9]

Para integrar el SDK de *Google Maps* dentro de nuestra aplicación *Android* hemos debido realizar 4 procedimientos, los cuales son especificados a breves rasgos a continuación:

Paso 1: instalar paquetes de *Google Maps* para APPs.

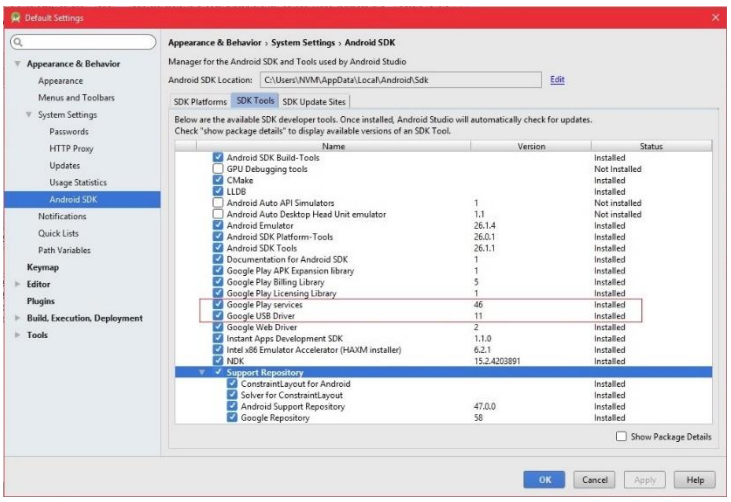


Figura 3.9: Instalación paquetes de *Google Maps*

La Figura 3.9 muestra una captura de pantalla de la ventana para administración de paquetes en *Android Studio*. Se enmarca con línea roja los dos paquetes necesarios para el funcionamiento de *Google Maps* en una APP: *Google Play Services* y *Google USB Driver* (necesario para desarrollos en *Windows*).

Paso 2: obtener clave de API desde *Google API Console*.

Para que el proyecto cargue las API de *Google Maps*, este debe tener un código de API que es provisto por *Google* a través de su página *Google API Console*. Debemos buscar *Google API Console* a través de cualquier motor de búsqueda y autenticarnos con una cuenta de usuario de *Google*. En esta página deberemos crear un proyecto, asignarle una credencial (como procedimiento opcional podemos restringir esa credencial para aplicaciones *Android*) y habilitar APIs que faciliten la gestión de las funcionalidades de la plataforma incrustada.

Figura 3.10: Obtener clave de API

La Figura 3.10 muestra la página de *Google API Console* pestaña "Credenciales", donde podremos copiar la "Clave de API" que es solicitada por el proyecto en el archivo google_maps_api.xml (*String Tag*) que se genera al crear un *Activity* de tipo *Maps*.

Paso 3: crear archivo google_maps_api.xml mediante *Google Maps Activity*.

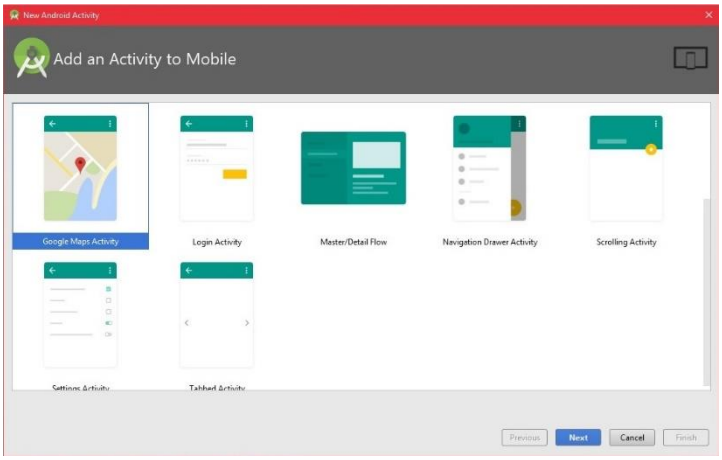


Figura 3.11: Creación de archivos para *Google Maps* en proyecto

La Figura 3.11 muestra una pantalla de *Android Studio*. Al crear un nuevo *Activity* de tipo *Google Maps*, se crean en conjunto tres archivos: El archivo ".java" que hereda de *Google Maps*, el archivo *layout* ".xml" para

el diseño visual y el archivo google_maps_api.xml. En este último (en *String Tag*) es donde se copia la clave de API obtenida desde *Google API Console*.

Paso 4: activar APIs de *Google Maps*

Es recomendable habilitar algunas API desde *Google API Console*. La opción "*Google Maps Android API"* es básica para el correcto funcionamiento de *Google Maps* en nuestras aplicaciones móviles *Android* pero para nuestro sistema hemos habilitado tres opciones adicionales para mejorar el rendimiento y la funcionalidad de *Google Maps*. Son las siguientes: "*Google Places API for Android"*, que nos servirá para actualizar ubicaciones en nuestra aplicación *Android*; "*Google Maps Geocoding API"*, que nos servirá para convertir ubicaciones en coordenadas geográficas; "*Google Maps Geolocation API"*, que nos ayudará en la gestión de geolocalización.

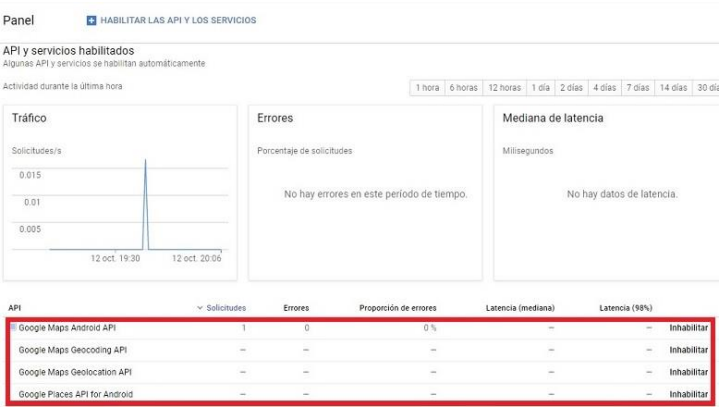


Figura 3.12: Panel de *Google API Console* con APIs habilitadas

La Figura 3.12 es un *print screen* de la página *Google API Console*. En la pestaña "Biblioteca" se selecciona la API que se desea habilitar y clic en botón "Habilitar". En la pestaña "Panel" se muestra un resumen de las API habilitadas para el proyecto de *Google Console* seleccionado.

3.1.5 Gestiones para integración de *Navigation Drawer Activity* con *Tabbed Activity*

Las condiciones del proyecto nos hicieron necesario gestionar las funcionalidades de la aplicación mediante menú de opciones, debido a que el proyecto tiene cuatro principales categorías: recolección, reconocimiento, presentación en *Google Maps*, y vista de galería. Sin embargo, en cualquiera de las dos principales funcionalidades: recolección y reconocimiento, se precisa realizar varias tareas para completar su función, así que también se necesitó gestionar ciertas tareas mediante diseño de pestañas deslizables. Es así como surgió la necesidad de tener una interfaz integrada por diseño de menú con diseño de pestañas. Existen ejemplos publicados en internet sobre la forma como hacer una integración básica de estos dos tipos de diseño. Nosotros nos basamos en un ejemplo publicado en la cuenta de un usuario del sitio web YouTube. [10]

El concepto detrás de esta integración es incrustar las pestañas deslizables dentro de los archivos que gestionan el menú de opciones, luego, gestionar las pestañas y su contenido mediante dos tipos de archivos: un contenedor que funciona como maqueta para la pestaña de turno, y un adaptador que funciona como administrador del contenido a ser cargado en el contenedor. En el adaptador se usan *Threads* para gestionar el cambio de contenido para la pestaña de turno.

Figura 3.13: Distribución de archivos usados para la integración

En la Figura 3.13 se muestra un esquema representativo de la distribución de archivos para la integración de los dos estilos. Los archivos de abajo representan los archivos *Fragment* (archivo ".xml" con archivo ".java") utilizados para formar las pestañas deslizables, y la flecha azul que los apunta indica que los archivos contenedor y adaptador contienen a estos.

3.1.6 Gestiones para envío/ recepción de datos mediante protocolo HTTP hacia el servidor principal

En nuestro proyecto al menos dos funcionalidades principales dependen de una comunicación hacia el servidor en *Linux* mediante protocolo HTTP: reconocimiento de especies que depende del módulo *Machine Learning*, y sincronización de datos de la base *SQLite* interna con la base de datos central. La comunicación es establecida con arquetipo REST haciendo peticiones *get* y *post*, y hemos usado funciones asíncronas en estas peticiones puesto que la tarea de sincronización no debe detener el flujo normal del programa. Pero gestionar el envío y recepción de datos en estas funciones asíncronas de forma sinérgica con el programa se volvió algo complejo, aunque finalmente se resolvió con éxito las dificultades.

Dividimos en tres pasos las tareas que fueron necesarias para gestionar satisfactoriamente este envío y recepción de datos y las presentamos a continuación:

Paso 1: hacer petición asíncrona.

```
public JSONObject saveResponseToken(OnJSONObjectResponse
                                        callback){
    AsyncHttpClient client = new AsyncHttpClient();
    RequestParams requestParams = new RequestParams();

    try {
    requestParams.
            setForceMultipartEntityContentType(true);
    requestParams.put("user", MainActivity.data.getUsername());
    requestParams.put("pwd", MainActivity.data.getPassword());
    }catch (Exception e){
    }

    client.post(urlTokenize, requestParams,
            new AsyncHttpResponseHandler() {
       @Override
       public void onSuccess(int statusCode, Header[] headers,
                             byte[] responseBody) {
       }

       @Override
       public void onFailure(int statusCode, Header[] headers,
              byte[] responseBody, Throwable error) {
       }
    });
    return new JSONObject();
}
```

Figura 3.14: *Script* de envío / recepción de datos en función asíncrona

La Figura 3.14 muestra una captura de pantalla del código de implementación básico para una función asíncrona, en la cual se envían parámetros en la petición y se ha modificado el parámetro de *header*: "*Content-Type*", al valor de: "*multipart/form-data*". [11]

Paso 2: gestionar la respuesta de la función asíncrona mediante una función *callback*.

```
public interface OnJSONObjectResponse {
    void onJSONResponse(boolean success,
                        JSONObject response);
}

public JSONObject saveResponseToken(OnJSONObjectResponse
                                    callback){
    callbackClase = callback;

    client.post(urlTokenize, requestParams,
        new AsyncHttpResponseHandler() {
            @Override
            public void onSuccess(int statusCode, Header[] headers,
                                  byte[] responseBody) {
                if ( responseBody!=null && responseBody.length>0 ){
                    try {
                        JSONObject jsonObj = new JSONObject(
                                new String(responseBody));
                        callbackClase.onJSONResponse(true, jsonObj);
                    } catch(Exception ex){
                    }
                }
            }

            @Override
            public void onFailure(int statusCode, Header[] headers,
                    byte[] responseBody, Throwable error) {
                try {
                    callbackClase.onJSONResponse(false,
                            new JSONObject());
                } catch(Exception e){
                }
            }
        });
    return new JSONObject();
}
```

Figura 3.15: *Script* para gestión de respuesta de función asíncrona

La Figura 3.15 muestra una captura de pantalla del código de implementación básico para la gestión de las respuestas de nuestra función asíncrona. Para esto hemos creado una interface la cual contiene un parámetro de estado, y es establecida como parámetro a manera de función *callback* en la implementación de la función asíncrona. Durante la ejecución de la función asíncrona esta le da el estado (*true* o *false*) a la función *callback* cuando termina de recibir la respuesta a las peticiones *get* y *post*. [12]

Paso 3: ejecución de función asíncrona en nuestro código.

Figura 3.16: *Script* para llamada a función asíncrona

La Figura 3.16 muestra una captura de pantalla del código mediante el cual se ejecuta la función asíncrona dentro de nuestro proyecto. Notamos que al ser la función asíncrona una función de una determinada clase entonces su ejecución se realiza con el procedimiento típico de un lenguaje orientado a objetos, es decir de la forma: objeto punto propiedad. [12]

3.1.7 Implementación final de interfaz de usuario

A pesar de que en la metodología de desarrollo para el módulo APP descrita en el capítulo 2 se definieron un conjunto de prototipo de pantallas para la aplicación móvil, que seguían un modelo de transición serial tipo *wizard,* pero el concepto estructural de la aplicación y el flujo de actividades en la aplicación requirieron adaptar un nuevo diseño de interfaz final al proyecto, y se notó que un diseño de interfaz final tipo menú que incluya pantallas deslizables se adaptaba de forma óptima a las necesidades del proyecto. De esta manera, la interfaz final de usuario ha quedado establecida por el conjunto de pantallas que son presentadas a continuación:

- Pantallas iniciales

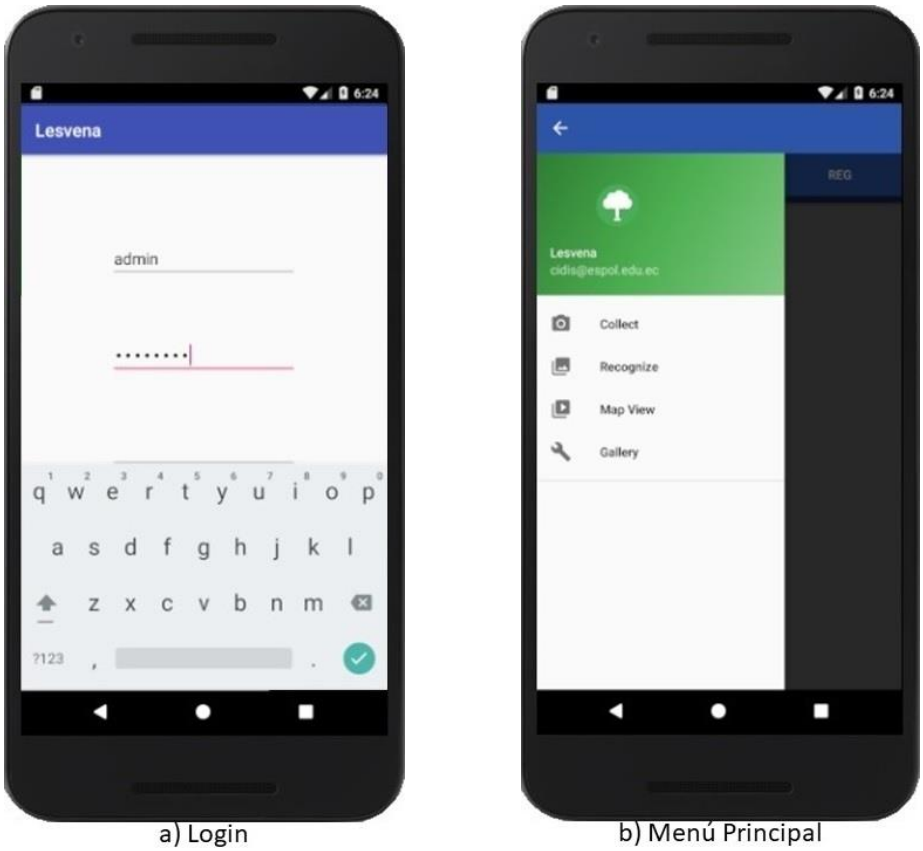


Figura 3.17: Pantallas de interfaz usuario APP: pantallas (a) Login y (b) Menú Principal

En la Figura 3.17 se presentan dos pantallas de interfaz de usuario en APP. En pantalla (a) Login se muestra una pantalla básica para autenticación de usuario con dos campos de ingreso: usuario, contraseña, y un botón de aceptar. En pantalla (b) Menú Principal se muestra una pantalla de interfaz de tipo menú que contiene las cuatro principales funciones de la aplicación: recolectar, reconocer, vista de mapa, y galería. Todos los nombres en la interfaz de usuario han sido expresados en inglés.

- Menú principal: opción “*Collect*”

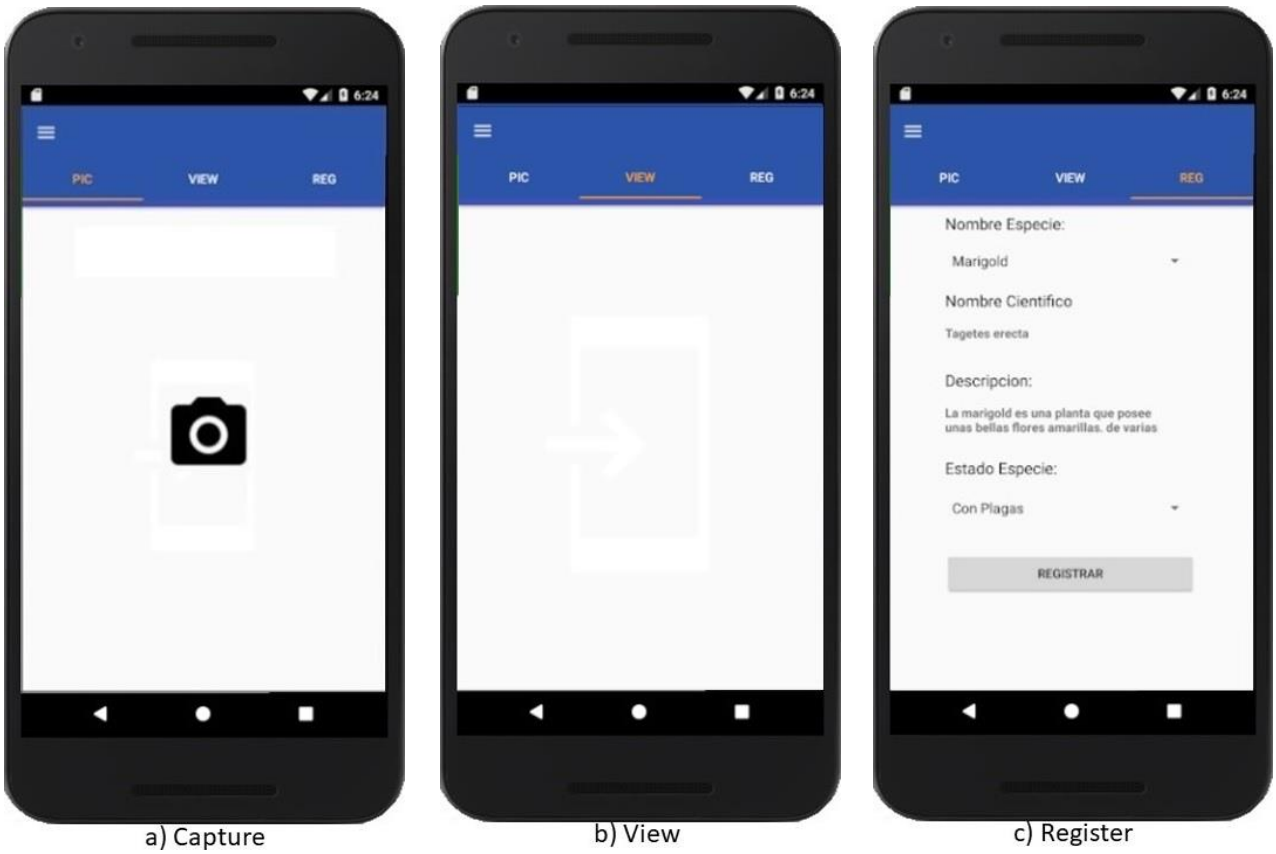


Figura 3.18: Pantallas de interfaz usuario APP: pantallas (a) Capture, (b) View, y (c) Register

En la Figura 3.18 se presentan las tres pantallas del menú “*Collect”* en APP. Son cargadas una a la vez por medio de presionar la pestaña respectiva o en modo de pantalla deslizable. En pantalla (a) Capture se tiene un botón con ícono de cámara que al presionarlo presenta la cámara del dispositivo para la captura de fotos. En pantalla (b) View se muestra un marco para la presentación de la foto capturada. En pantalla (c) Register se muestra un formato de formulario con dos botones de selección múltiple: nombre de especie y estado, y un botón de aceptar, esta pantalla permite registrar en base *SQLite* las selecciones del usuario.

- Menú principal: opción “*Recognize”*

Figura 3.19: Pantallas de interfaz usuario APP: pantallas (a) Capture y (b) View

En la Figura 3.19 se presentan dos primeras pantallas de un total de cinco del menú “*Recognize”* en la interfaz de usuario en APP. Son cargadas una a la vez por medio de presionar la pestaña respectiva o en modo de pantalla deslizable. En pantalla (a) Capture se tiene un botón con ícono de cámara que al presionarlo presenta la cámara del dispositivo para la captura de fotos. En pantalla (b) View se muestra un marco para la presentación de la foto capturada.

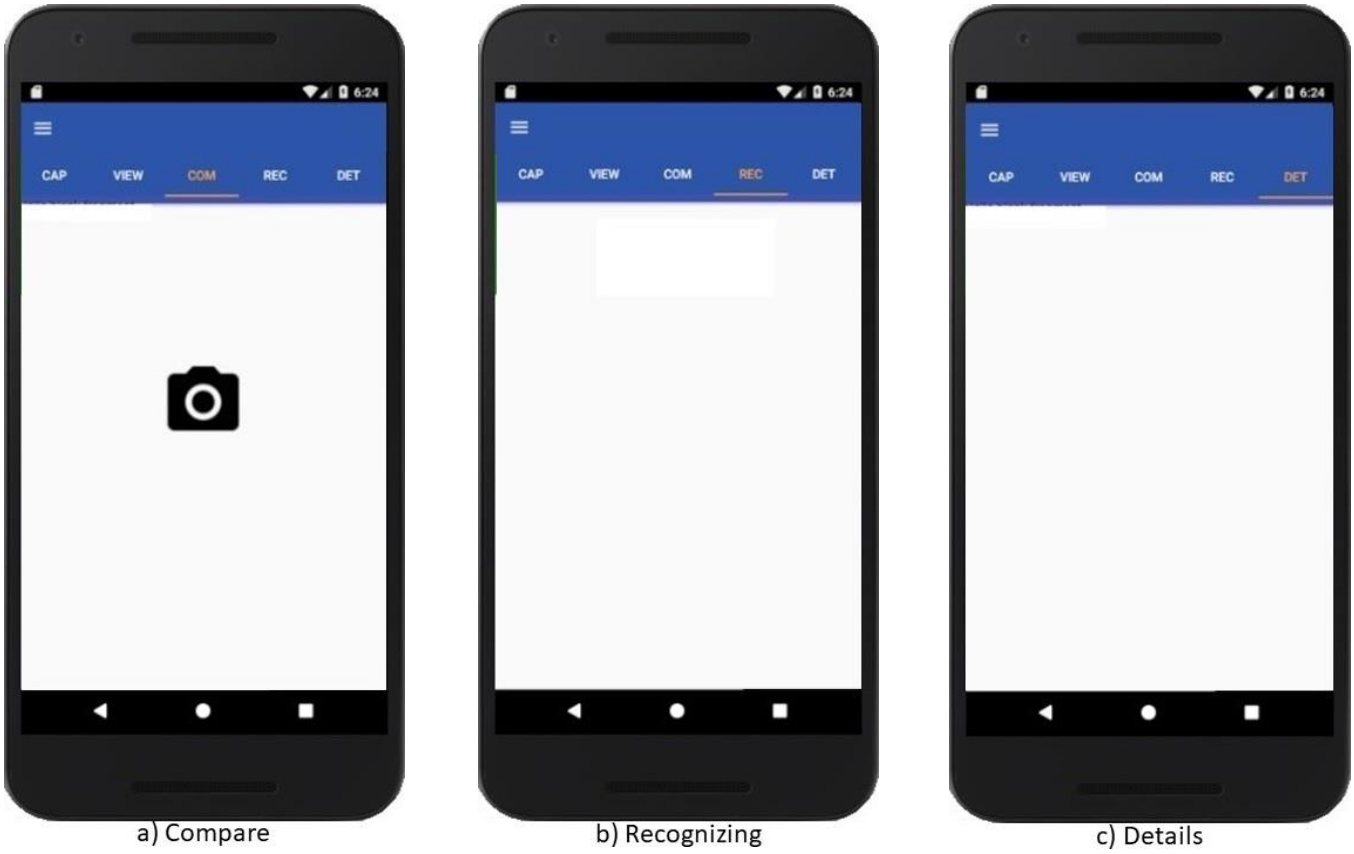


Figura 3.20: Pantallas de interfaz usuario APP: pantallas (a) *Compare*, (b) *Recognizing*, y (c) *Details*

En la Figura 3.20 se presentan tres últimas pantallas de un total de cinco del menú "*Recognize"* en la interfaz de usuario en APP. Son cargadas una a la vez por medio de presionar la pestaña respectiva o en modo de pantalla deslizable. En pantalla (a) Compare se tiene un botón con ícono que al presionarlo ejecuta un código que envía la foto capturada al API de reconocimiento de imágenes, recibe la respuesta, y lanza los resultados a la siguiente pantalla. En pantalla (b) *Recognizing* se muestra un marco para la presentación de los resultados de reconocimiento de especie. En pantalla (c) *Details* se muestra un marco para la presentación de detalles de información de la especie seleccionada al dar clic en un botón de la pantalla (b) *Recognizing* anterior.

- Menú principal: opciones “*Map View”* y “*Gallery”*

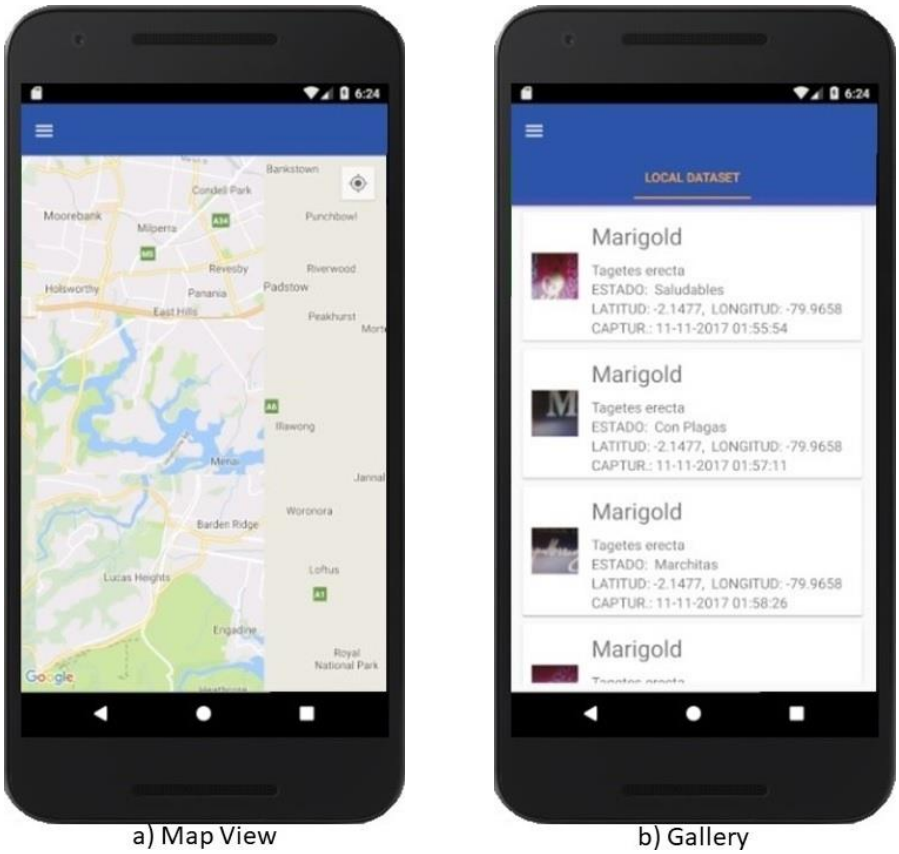


Figura 3.21: Pantallas de interfaz usuario APP: pantallas (a) Map View y (b) Gallery

En la Figura 3.21 se presentan dos pantallas del menú principal en APP. En pantalla (a) Map View se muestra una pantalla que carga una porción de *Google Maps* y donde se coloca un marcador que representa la ubicación GPS de la foto capturada. En pantalla (b) Gallery se muestra un *list card view* que es un diseño de vanguardia para presentación de listados, en nuestro caso listado de especies que han sido capturadas en el dispositivo.

3.2 Implementación del módulo WEB

En esta sección describiremos los detalles de la implementación de la solución para el módulo WEB. Se ha considerado dar detalle sobre los aspectos que fueron críticos durante la implementación de este módulo como por ejemplo: las configuraciones necesarias en el servidor para un correcto funcionamiento del ambiente web; las gestiones para creación del ambiente web del lado del servidor (*back – end*: *Laravel*); las gestiones para creación del ambiente web del lado del cliente (*front – end*: *Bootstrap*); los diferentes niveles de acceso; las configuraciones para creación del API REST; el código implementado en PHP para comunicación con el archivo *Python* de reconocimiento de imágenes incrustado; y finalmente, el código implementado para comunicación con la aplicación móvil.

3.2.1 Configuraciones para el servidor web

El sistema operativo que se ejecuta en el servidor en el cual nuestro sistema está alojado es un ambiente *Linux* (*Ubuntu server*) sin entorno gráfico por lo cual todas las configuraciones e instalaciones se las realizaron por comandos de consola. Este servidor posee la versión de PHP 7.0 instalada lo cual habilita a usar versiones de *Laravel* 5.4 o superior. Con *Composer* se realizó la instalación de *Laravel* 5.5 y para las ediciones de los archivos de configuración se usó VIM, un editor de texto por consola propio de *Linux*.

Paquetes de PHP agregados al ambiente LAMP. Adicional a los paquetes básicos de PHP se agregaron ciertos módulos necesarios para un correcto funcionamiento de *Laravel* 5.5, que son nombrados a continuación:

php-xml; php-mbstring; php-intl

Creación de proyecto en *Laravel*. En la máquina servidor nos ubicamos dentro del directorio: “/var/www/html/”, que es el directorio desde donde el

servidor web *Apache* toma los proyectos y los dispone para su acceso web. Se creó la carpeta “LesvenaLa” en una máquina local haciendo uso de un ambiente XAMP para emular el ambiente LAMP del servidor. A esta carpeta se agregaron un par de archivos de configuración que contienen la tecnología *Laravel*, lo cual nos brinda un proyecto base de *Laravel* sobre el cual montar nuestro código. La carpeta final fue cargada nuevamente al servidor mediante protocolo FTP. Existen tutoriales en internet para la creación y configuración de un proyecto base de *Laravel*.

Archivos configurados del proyecto en *Laravel*. Habiendo realizado la configuración base para el proyecto y habiendo creado la base de datos según se detalló en el módulo DATASET, tuvimos que configurar el archivo “.env”, el cual contiene las configuraciones para la conexión a la base de datos, las claves de autenticación, y los datos de configuración para envío de *e-mail*. Las variables establecidas fueron las siguientes:

```
APP_KEY=clave
APP_URL=http://localhost

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=database
DB_USERNAME=user
DB_PASSWORD=pass
```

Comandos de *Laravel* para iniciar el proyecto. Finalmente, nos servimos de los comandos “php artisan” para crear la clave de autenticación para el proyecto, y el módulo inicial para autenticación de usuarios. Para la instalación del componente de autenticación, nos ubicamos haciendo uso de la consola de comandos en el directorio del proyecto y se hizo uso del comando:

```
php artisan make:auth
```

El cual, genera una estructura base para el módulo de *Login* (autenticación de usuarios). Para la creación de la clave del proyecto (“APP_KEY”), nos ubicamos haciendo uso de la consola de comandos en el directorio del proyecto y se hizo uso del comando:

```
php artisan key:generate
```

Ahora, tenemos configurado el proyecto para que se conecte a la base de datos, para que se pueda acceder a este mediante web en dirección “http://localhost”, y tenemos generada la clave única de proyecto (“APP_KEY”) para cumplir con los protocolos de seguridad de *Laravel*.

3.2.2 Creación de CRUDS

Las funcionalidades básicas de la aplicación web son proporcionar interfaces de CRUDS (*Create Read Update Delete*, por sus siglas en inglés) para las tablas: Dataset, Usuario, Clasificacion, y Parte, las cuales son parte fundamental de la base de datos central de nuestro sistema. Se tiene además, funcionalidades particulares como validaciones de datos en los formularios, y visualización de mapa (*Google Maps*) con marcador nativo, para indicar la ubicación georreferenciada de la foto capturada. La estructura de implementación interna de la aplicación web, que soportan estas funcionalidades del proyecto, es de 3 capas M-V-C (*Model View Controller*, por sus siglas en inglés). Para la implementación de los modelos, controladores, y migraciones (debido a *Laravel*), nos servimos de los comandos “php artisan” propios del *framework*. Para la creación de vistas creamos archivos en un directorio del proyecto. Luego, se procedió a implementar el código necesario a cada uno de los archivos para que se ajusten al diseño del sitio así como a las funcionalidades particulares del mismo.

Creación de vistas (*Views*). Para crear las vistas nos dirigimos al directorio: “/resources/views/”, dentro del proyecto, y creamos un archivo con extensión “blade.php” para cada una de las vistas que deseemos.

```
NombreVista.blade.php
```

Creación de migraciones. Esta tecnología es propia de *Laravel* e involucra 2 pasos: creación de archivos de migración dentro del proyecto, y ejecución del código dentro de los archivos de migración para implementación de las tablas de la base de datos. En los archivos de migración desarrollamos el código para creación de las tablas y sus correspondientes campos. Luego, mediante la consola de comandos ejecutamos el código de los archivos de migración:

```
php artisan migrate
```

Si se presenta algún error al ejecutar la migración, se devuelve la base de datos a su estado original ejecutando el comando:

```
php artisan migrate:rollback
```

Creación de modelos (*Entities*). Una vez creadas todas las tablas en los archivos de migración, se deben crear los archivos modelo (*Entity*) que son reflejo de las tablas de la base de datos, colocando todos los campos que van a ser tomados en cuenta dentro del proyecto. En estos mismos archivos de modelo se crean las funciones necesarias para relacionar los datos de uno a uno, de uno a muchos y de muchos a muchos, se crean las relaciones entre archivos modelo según las relaciones entre tablas de la base de datos. Mediante la consola de comandos ejecutamos el código:

```
php artisan make:model NombreDelModelo -mc
```

Direccionamientos en archivo “routes.php”. Luego de haber creado todos los recursos anteriores, se crearon las rutas de cada recurso interno del proyecto dentro del archivo “routes.php” ubicado en el directorio “/app/Http/” del proyecto. Estas rutas establecen emparejamientos entre los enlaces (URL) externos que apuntan al proyecto y los recursos internos de este, que pueden ser: una determinada vista, o un método de algún archivo controlador.

Creación de *controllers*. La creación de los archivos controladores se puede hacer mediante los comandos “php artisan” como por ejemplo:

```
php artisan make:controller NombreControlador
```

Pero también se pueden crear como archivos con extensión “.php” dentro del directorio “/app/Http/Controllers/” del proyecto. En estos archivos creamos las funciones que relacionan a las vistas y los modelos de base de datos, basados en los emparejamientos establecidos en el archivo “routes.php”. En estas funciones además implementamos las diferentes validaciones del código, y la lógica de programación del proyecto.

3.2.3 Niveles de acceso

El proyecto está dividido en 3 niveles: “Administrador”, “Experto” y “Básico”. Pero, no todos estos roles están implementados sino sólo el rol “Administrador”. Para la implementación de estos niveles de acceso nos ayudamos con los algoritmos de encriptación de ciertas librerías de *Laravel* 5.5, teniendo en cuenta que usaremos estas mismas credenciales para la APP, así como para el API REST.

Tipo de encriptación implementado. El tipo de encriptación que hemos usado es ´AES encryption´ y se lo ha implementado mediante la extensión de PHP: *Mcrypt* (“php-mcrypt”). Para el uso de esta encriptación en la API se crea una ruta con el prefijo de: “api/v1”, donde esté activo un

middleware tal que este envíe un *token* determinado dependiendo del usuario registrado. Esta clave denominada *token* es un *string* único de 60 caracteres, donde un registro de *token* determinado está asociado a cada usuario. Un ejemplo de ruta para la API es la siguiente:

api/v1/Login

3.2.4 Implementación de *Bootstrap*

La versión de *Bootstrap* usada es la 4.0 que contiene muchas funcionalidades importantes para el desarrollo de un sistema adaptable a cualquier dispositivo de donde se ejecute. Se usaron los siguientes recursos de *Bootstrap*: el *header*, menú *dropdown*, barras de navegación, pie de página, formularios con la tecnología de *Material Design*, alertas, paquetes de iconos (*Font Awesome*).

Incrustación del *framework* al proyecto. Se hizo uso de la versión compilada del *framework*, donde los archivos ".css" y ".javascipt" descargados fueron agregados dentro de la carpeta "/public/" dentro del proyecto. Además, la presentación de las correspondientes páginas de vista ha sido establecida en el archivo "routes.php" y el comportamiento del contenido de las vistas está establecido en los diferentes archivos controladores del proyecto.

3.2.5 Implementación de la API REST

La segunda parte de la aplicación web es una API (*Application Programming Interface*, por sus siglas en inglés), con REST como protocolo de intercambio de datos por HTTP y con JSON para el formato de datos que son transferidos a través de la web. La función general de esta API es servir de interface de comunicación entre la aplicación móvil (APP) y las tareas que se realizan en el servidor, tales como: ejecutar el código en *Python* para reconocimiento de imágenes, registrar en la base central los datos enviados por la APP, entre otros. Para la implementación de la API REST se usó PHP 7, y aunque es una aplicación independiente

de la página web para gestión de CRUDS, pero nos servimos del mismo servidor web configurado para esta (incluso la misma carpeta del proyecto web) para utilizar las librerías de acople entre PHP y *Apache* 2.

Para la implementación de esta API se usó un enfoque orientado a objetos, donde, las tareas similares son agrupadas por clases, y estas clases contienen propiedades y métodos dentro de sí para la representación de un objeto dado. A continuación presentamos los detalles de su implementación.

Clase transaccional "REST". Dentro de la carpeta "/api/" tenemos un archivo llamado "Rest.inc.php". Este archivo contiene la clase principal "REST" que es donde están desarrolladas las funciones transaccionales que permiten el funcionamiento de la API REST. Contiene tres grupos de funcionalidades. Funciones de estado HTTP: son funciones que ayudan a gestionar los códigos de respuesta generados por un servidor web, y van desde un mensaje 100 hasta el mensaje de error 505. Funciones de cabecera HTTP: son las funciones que contienen las respuestas a los diferentes peticiones de tareas, que son solicitadas a través de HTTP haciendo uso de métodos POST, GET, PUT, y de formularios de datos en formato JSON. Funciones de limpieza de datos: se hace uso de las funciones "trim" y "stripslashes" de PHP que sirven para realizar limpieza de data por medio de eliminar espacios en blanco y convertir caracteres especiales en caracteres comunes equivalentes.

Estructura interna de la API. Como se ha comentado anteriormente, la estructura de archivos de la aplicación está basada en la metodología de orientación a objetos. Para gestionar los diversos recursos de las clases se ha implementado la clase principal "api.php", en la cual se implementa la lógica transaccional de la API REST. Los recursos que son utilizados en esta clase se especifican a continuación. Conexión con la base de

datos: para gestionar la conexión con la base de datos se carga la librería “mysqli”, y haciendo uso de una función a la cual se especifican los parámetros de servidor, usuario, contraseña y nombre de la base: “(server, user, pwd, db)”. Uso de funciones REST: se carga el archivo “Rest.inc.php” del cual se usan sus funciones a manera de objeto – propiedad. Validación de las URL: se hace uso del archivo “.htaccess” muy conocido en PHP, donde se han creado alias para formar URL amigables, y emparejamientos de determinadas URL externas con direccionamientos a funciones internas requeridas, agregando además lanzamiento de resultado con error 404 (página no encontrada) para aquellos casos donde la URL solicitada desde afuera no exista en nuestros registros. Validaciones de método POST y variables válidas: en cada método transaccional se tienen validaciones del tipo intento – captura (*try* - *catch*) para validar que todas las peticiones sean por método POST caso contrario se devuelve mensaje de error 406 (petición en formato no aceptado), y para validar que toda variable que llega en cada petición HTTP no sea valor vacío caso contrario se devuelve mensaje de error 204 (sin contenido).

Formato peticiones *Request*. Las funcionalidades requeridas desde la APP hacia la API REST son: *login* (autenticación de usuario), sincronizar bases, cargar imagen al servidor, reconocer imagen de especie, cargar registros de tabla Dataset, consultar tabla Clasificacion. Esta comunicación es desarrollada por protocolo HTTP mediante peticiones tipo *Request*, y con formato de envío – recepción de data tipo JSON. Además se establece el *header* (cabecera) de este *Request* en tipo “multipart/form-data” debido a que deben trasladarse datos en estructura de formulario. A continuación especificamos la estructura de la URL (con la IP utilizada a la fecha), y la estructura base del *Request* utilizado.

URL: http://23.98.220.201/lesvena/api/FuncionalidadSolicitada
Header:

Content – Type: multipart / form-data

Body: (parámetros)
Variable 1: valor
Variable 2: valor
Variable n: valor (si es listado con contenido lleva formato de objeto JSON)

Funcionalidad "*login*". La finalidad de esta función es permitir la autenticación de un usuario desde la APP. Recibe dos parámetros: usuario y contraseña ("user", "pwd"), se verifica si las variables no son vacías, y se consulta a la base de datos del servidor por medio de la librería "mysqli" si existen estos valores de parámetro. En caso de coincidencia devuelve: el id del usuario, nombre, y un valor de *token* registrado para el usuario devuelto ("user", "name", "token"). En caso de no coincidir devuelve: un mensaje de error 400 (error en credenciales de acceso). La dirección en la URL para esta funcionalidad es: "/api/login/".

Funcionalidad "validar". Para proteger las seguridades de la conexión entre la APP y el servidor se hace uso de una función privada llamada "validar", la cual se ejecuta antes de ejecutar cualquiera de las funciones especificadas abajo y que son solicitadas por la APP, y cuya finalidad es verificar que el usuario y valor de *token* recibidos desde la APP sean los correctos. Esta función recibe: usuario, *token*, y se constata estos valores en la base de datos. Devuelve: valor de verdadero o falso (*true o false*) según coincidan o no.

Funcionalidad "sincronizar". La finalidad de esta función es entregar a la APP los registros que esta necesita para cargar en sus listas desplegables del formulario de registro de especies y será solicitada cada vez que la APP se cargue en un dispositivo móvil. En cada solicitud a esta

función se ejecuta primero la función “validar”. Recibe dos parámetros: usuario y *token* (“user”, “token”). Luego, se consulta a las siguientes tablas de la base de datos: Clasificacion, y Parte. En caso de coincidencia devuelve: los registros de las tablas mencionadas en formato de objeto JSON. La dirección en la URL para esta funcionalidad es: “/api/sincronizar/”.

Funcionalidad “cargar imagen”. La idea detrás de esta función es que sea subida la foto que fue capturada por el dispositivo móvil al servidor el cual tiene una IP pública, de tal modo que se tenga acceso a la foto en cualquier momento a través del internet. Recibe tres parámetros: usuario, *token*, imagen en formato “Base64” (“user”, “token”, “img”), donde con la función “base64_decode” se convierte este *string* a un archivo, y con la función “file_put_contents” se realiza el guardado de este archivo en formato “.png”, en un directorio del servidor. En caso de éxito devuelve: la ubicación en el directorio del servidor, donde está alojada la fotografía. La dirección en la URL para esta funcionalidad es: “/api/uploadImg/”.

Funcionalidad “reconocer”. Por medio de esta función nos servimos de la tecnología de inteligencia artificial, específicamente de una máquina de aprendizaje automático, puesto que recibe una fotografía de especie desde una APP, se compara esta fotografía contra el grupo de fotografías base ejecutando un archivo de *Python*, y se devuelve un listado de las fotografías que se le parecen. Aunque el proceso se encuentra en un estado inicial la idea detrás es reconocer el tipo de especie basándonos en la fotografía. Recibe tres parámetros: usuario, token, imagen en formato “Base64” (“user”, “token”, “img”), donde con la función “base64_decode” se convierte este *string* a un archivo, y con la función “file_put_contents” se realiza el guardado de este archivo en el directorio: “/reco/img/”, del servidor. Esta imagen es cargada junto al archivo “test_one_img.py” el cual compara la imagen contra una lista que contiene

un tipo de dato *Array* con las diferentes flores existentes. En caso de éxito devuelve: un listado de descripción de las especies que coincidieron con la fotografía, en formato de objeto JSON. La dirección en la URL para esta funcionalidad es: "/api/reconocimiento/".

Funcionalidad "cargar data set". Otro de los procesos realizados por esta API REST es guardar en la tabla: "dataset", de la base de datos del servidor, los registros de la tabla homónima: "dataset", enviados desde la APP, por motivo de mantener sincronizados los datos que se registran en la base nativa de la APP y la base de datos central. Recibe tres parámetros: usuario, *token*, arreglo de datos en formato objeto de JSON ("user", "token", "values"), donde si las credenciales y datos son válidos, se realiza un proceso de guardado. Se decodifica el objeto JSON recibido como parámetro transportando su contenido a un arreglo, y se realiza un proceso cíclico ("foreach") ejecutando una función de inserción mediante el uso de "mysqli" en cada ciclo. En caso de éxito devuelve: la cantidad de registros insertados. La dirección en la URL para esta funcionalidad es: "/api/loadDataset/".

Funcionalidad "consultar tabla Clasificacion". Para finalizar tenemos esta función de consulta, la cual devuelve los registros de la tabla: Clasificacion. También existe la copia de esta función para devolver un registro específico de la tabla mencionada en base a un "id" recibido como parámetro. Recibe dos parámetros: usuario, y *token* ("user", "token"). En caso de éxito devuelve: los registros de la tabla solicitada en formato de objeto JSON. La dirección en la URL para esta funcionalidad es: "/api/clasificacion/".

3.2.6 Comunicación con archivo *Python* incrustado

Dentro de las funcionalidades de la API REST tenemos la funcionalidad de reconocimiento de especies mediante sus fotografías, esta es una función que conlleva una complejidad mayor a las demás debido a que se

debe interactuar con un archivo en lenguaje *Python* (incrustado en la carpeta del proyecto), el cual depende de varias librerías para ejecutar su tarea. A continuación describimos detalles sobre la implementación de este archivo en *Python* y la forma como interactuamos desde nuestro código en PHP con el mencionado archivo.

Comando en PHP para uso de archivo en *Python*. Dentro de la carpeta de nuestro proyecto web, y dentro de la carpeta "/api/" tenemos un archivo con nombre "test_one_img.py", el cual es el archivo en lenguaje *Python* para reconocimiento de imágenes al cual hemos hecho referencia. Este archivo está ubicado un directorio abajo del archivo principal de la API "api.php", desde la cual para ser ejecutado se lo hace por medio de consola de comandos valiéndonos de la función "exec" de PHP. El comando tiene la siguiente estructura: "python test_one_img.py $imagen_resultante /result/$lista_flores". Donde, "$imagen_resultante" es la dirección URL de la imagen cargada al servidor que se solicita reconocer, y "$lista_flores" es la ubicación del archivo con la lista de todas las imágenes de especies (flores) contra quienes se realiza la comparación.

Librería "*Tensor Flow*". Dentro del archivo en lenguaje *Python* se hace uso de varias librerías, las dos principales son: "tensorflow" y "simplejson". *TensorFlow* es una librería de *Python* para aprendizaje automático, basado específicamente en la técnica de inteligencia artificial de redes neuronales. Esta librería permite detectar y descifrar patrones y correlaciones entre un individuo y un grupo de contraste. En nuestro caso específico, para el proyecto se usa *Tensor Flow Image Classifier*, que como su nombre lo indica clasifica imágenes a partir de un listado de objetos existentes. La primera tarea que realiza es obtener la lista de nombres de archivos a relacionar con su función "get_labels". Más abajo reconocimiento de patrones dentro del archivo: "test_one_img.py".

Archivo *Python*: funciones implementadas. Función "predict_on_image": esta función recibe la imagen a la cual se la desea clasificar (encontrar un patrón), la función obtiene acceso a un archivo llamado: "retrained_graph.pb", el cual es un archivo especial usado para búsqueda, que contiene las referencias de búsqueda (índices o directorios) para las imágenes que son usadas como contraste. La función llena al archivo de índices con los valores correspondientes.

Función "GraphDef": esta función, de forma general sirve para imprimir, almacenar o restaurar un gráfico. Para nuestro caso, nos ha servido para importar el archivo "retrained_graph.pb" que contiene ya las predicciones para la imagen solicitada.

Función "run": se hace uso de esta función para ejecutar el proceso de reconocimiento de patrones. La función recibe dos parámetros: los datos arrojados por la función "GraphDef", y la imagen que se solicita comparar, la cual es decodificada por medio del uso de una función llamada "DecodeJpeg". Luego, todos y cada uno de los resultados de las predicciones son agregados a una lista en formato *Array* de datos en *Python*: "msg["RESULT"]".

Función "writeJSON": finalmente, se ejecuta esta función cuyo trabajo es convertir el arreglo de datos que devuelve la función "run" hacia un formato de objeto JSON, este resultado a su vez es entregado a la función original en PHP.

3.2.7 Comunicación con aplicación móvil

El proceso de comunicación de la API REST con cualquier dispositivo móvil donde se ejecute la APP se inicia cuando esta nos solicita el valor de *token* enviándonos como parámetros: un usuario y una contraseña ("user", "pwd"), según el proceso que ya fue detallado en un numeral anterior. Si el proceso de autenticación es correcto entonces la APP podrá

realizar el uso completo de toda la API enviándonos como parámetro en toda petición el *token* que le hemos entregado. Además de haber especificado en un numeral anterior la estructura del *Request* usado, en este numeral indicamos la librería y tarea adicional que se ha tenido que realizar por motivo especifico de la comunicación con un dispositivo móvil.

Librería "*Asynchronous HTTP Client for Android*". Esta función está basada sobre las bibliotecas "HttpClient" de *Apache* pero adaptada al lenguaje *Java* debido a que debemos comunicarnos con un sistema nativo *Android* donde se ejecuta tecnología *Java*. Su funcionalidad es ejecutar y recibir peticiones HTTP asíncronas desde la API REST hacia la APP.

3.3 Implementación del módulo DATASET

En esta sección describiremos los detalles de la implementación de la solución para el módulo DATASET. Se ha considerado dar detalle sobre los aspectos que fueron críticos durante la implementación de este módulo como por ejemplo: verificación y añadidura de configuraciones en el ambiente LAMP para el correcto funcionamiento del proyecto; herramientas adicionales de respaldo; recopilación de datos para la creación del *data set* inicial; creación del *data set* inicial; y entrenamiento de la máquina de aprendizaje automático.

3.3.1 Configuraciones sobre entorno *Linux*

Para el desarrollo de nuestro proyecto se tuvo requerimiento de un espacio físico - digital, el cual fue facilitado por el CIDIS a través de un espacio físico en uno de sus servidores, provisto de acceso a internet a través de la dirección IP pública: 23.98.220.201, el cuál además ya estuvo dotado de un sistema operativo *Linux* y un ambiente básico LAMP (*Linux, Apache, MySQL, PHP*). Se ha estado ingresando al servidor por medios remotos utilizando el programa *puTTY 0.70*, y valiéndonos del protocolo de administración remota *SSH.* Habiendo tenido este escenario, el primer paso fue familiarizarnos con el entorno, verificar instalaciones, y realizar pruebas. A continuación describimos las principales características del entorno.

Verificación del ambiente LAMP. Las versiones instaladas de LAMP que encontramos en el servidor fueron las siguientes:

Linux: $ cat /etc/issue
Ubuntu 16.04 \n\l

Apache: /bin: $ apache2ctl –v
Apache 2.0

MySQL: $ service mysqld start – select version
MySQL 9.0.6
PHP: $ service httpd start – php –v
PHP 7.0.4

Configuración del *Firewall* para permitir el tráfico web. Debimos configurar el firewall de tal forma que permita tanto conexiones HTTP como HTTPS:

$ sudo ufw app list
Muestra “Apache Full” dentro de aplicaciones habilitadas

$ sudo ufw app info “Apache Full”
Muestra a los puertos 80 y 443 habilitados

$ sudo ufw allow in “Apache Full”
Habilita el tráfico entrante para este perfil

Instalación de paquetes PHP. Para mejorar la funcionalidad general de PHP de instalaron paquetes adicionales. Se instalaron varios, se presentan los principales:

$ sudo apt - get install php-all-dev php-dev php-cli

3.3.2 Configuración de soportes tecnológicos adicionales

El ambiente LAMP es la plataforma básica que se debe tener configurada para iniciar con el desarrollo del proyecto. Pero, además de LAMP se necesitan agregar suministros y soportes tecnológicos a nuestra plataforma digital para que el proyecto web y la aplicación móvil puedan ejecutarse de manera idónea. Hemos considerado como soportes tecnológicos para el proyecto al sistema de base de datos que gestiona el almacenamiento central de datos, y hemos agregado herramientas tecnológicas para facilitar el desarrollo de las diversas tecnologías. Los detalles de estos soportes tecnológicos para el proyecto se especifican a continuación.

Instalación de herramientas tecnológicas. Para facilitar el desarrollo del proyecto nos hemos servido de algunas herramientas tecnológicas, tales como:

Postman v5.5.0 API *Development Environment*. Es programa que sirve como entorno de desarrollo de API debido a que ofrece un servicio integral de comunicación por protocolo HTTP mediante envío y recepción de comandos *Request*. Tiene dos versiones: un *plugin* para Google Chrome y programa ejecutable. Nosotros escogimos el programa ejecutable, y lo hemos utilizado como una plataforma para pruebas de envío y recepción de comandos *Request* entre las diferentes plataformas tecnológicas del proyecto.

PhpMyAdmin v4.7.7 *MySQL Database Administrator*. Es una interfaz visual para gestión del sistema de base de datos MySQL. Por medio de este programa hemos creado y gestionado todos los elementos de la base de datos del proyecto.

Git v2.15.1 *Distributed Version Control System*. Es un sistema para el control de versiones de código en el desarrollo de proyectos. En nuestro proyecto lo hemos utilizado en conjunto con la página de gestión de

versiones de código *BitBucket* con el objeto de llevar un registro de las diferentes versiones de código que generamos durante la construcción del proyecto, además, como un repositorio central del mismo del cual nos servimos los desarrolladores y el tutor.

SubLime Text 3 IDE. Es actualmente uno de los más utilizados entornos de desarrollo integrado (*Integrated Development Environment*) para lenguaje PHP. Hemos utilizado este IDE para el desarrollo del código en el proyecto.

Creación de la base de datos. Habiendo instalado de manera satisfactoria las herramientas tecnológicas de soporte, entonces procedimos a crear la base de datos central para el proyecto valiéndonos de la interfaz gráfica *PhpMyAdmin* para administración de bases de datos. La estructura de tablas implementada fue el modelo Entidad – Relación presentado en la metodología de desarrollo del proyecto del Capítulo 2.

3.3.3 Recopilación de información para *data set*

La creación de la *data set* nos ha implicado dos pasos: una carga inicial de datos por medios manuales, y la alimentación constante de la *data set* a través de: la aplicación móvil y la depuración e incremento de datos en la base existente. Esta carga de datos por medios manuales se basa en la captura con cámara digital de aproximadamente 450 fotos de especies: 150 fotos de 3 distintas especies de flores, con la información geográfica de todas y cada una de ellas. Para hacer realidad esto se debió realizar 3 viajes, de donde en el viaje a la vía Yaguachi pudimos conseguir suficiente cantidad de individuos de 3 diferentes especies y recopilamos las fotografías requeridas. Las especies recopiladas fueron las flores: Marigold, Anturio, y Chavelita.

3.3.4 Creación de *data set* inicial

Los datos básicos de cada especie que está registrada en la *data set* son: su foto (en formato .JPG), su posición geográfica, y datos característicos básicos como nombre y estado en el que se encuentra el individuo. Con el proceso de obtención de los 450 individuos descrito en el punto anterior obtuvimos la materia prima para elaborar una *data set* inicial, pero para que estos datos sean información que nos sirva tuvimos que hacer la carga de fotos al servidor en *Linux*, en carpeta dentro del servidor web, de tal forma que todas y cada una de las fotos puedan ser visibles o referenciadas desde cualquier lugar, en cualquier momento puesto que el servidor con IP pública siempre estará disponible a través de la internet.

Conversión de fotos de cámara digital a formato JPG. La recolección de fotos se la realizó con una cámara digital profesional modelo EOS 70, de la cual se obtuvieron fotografías de alta resolución pero los archivos extraídos de esta estuvieron en formato SVG, por tanto se tuvo que realizar una conversión masiva de formato SVG a formato JPG para las imágenes valiéndonos de un programa en línea para tal efecto. [13] Luego se descargó el conjunto de archivos transformados en un solo archivo comprimido (.zip).

Carga de fotos al servidor en Linux. Teniendo una interfaz de línea de comandos en nuestro servidor de *Linux*, las opciones para la transferencia de la gran cantidad de archivos de fotografías de especies hacia este, se hubieron presentado de 3 distintas maneras: por protocolo FTP, con la combinación de protocolos SSH y SCP entre la máquina en *Windows* hacia el servidor, y mediante un programa gratuito llamado WinSCP. Nosotros escogimos la primera opción haciendo uso del programa de licencia gratuita *FileZilla*.

Creación de conjunto de datos de prueba. La base de datos central del proyecto que tiene una estructura de tablas según se indicó en la

metodología en el capítulo 2, fue llenada de los siguientes datos: 1 registro de especie Marigold en tabla Clasificacion, 3 registros de estado en tabla Parte, 2 registros de fecha en tabla Sincronizacion. Esto, por motivo de uso de estos datos para prueba de transferencia y sincronización de datos entre el sistema de base de datos central del servidor en *Linux* y la base de datos nativa en la APP dada en tecnología *SQLite*. Las direcciones URL de las fotografías de especies disponibles en el servidor *Linux* son registradas en la base de datos nativa de la APP en tabla Dataset, y en momento de la sincronización de las dos bases: la nativa de la APP y la base de datos central en servidor, tanto registros como direcciones URL de fotografías son cargadas desde la tabla Dataset de la base nativa hacia la tabla Dataset de la base central del servidor.

3.3.5 Entrenamiento de la máquina de aprendizaje automático

Este proceso es de vital importancia durante el desarrollo de la *data set* pero fue desarrollado por nuestro tutor del CIDIS. El concepto detrás es que la máquina de reconocimiento de imágenes debe tener un patrón inicial sobre la tarea que deberá ejecutar desde su configuración en adelante. De esta manera, este proceso consiste en hacer algunas pruebas iniciales de reconocimiento con las fotografías que tenemos para formar un patrón base en la máquina de inteligencia artificial, tal que en las ejecuciones del proyecto posteriores a esta configuración la máquina de aprendizaje conste de los patrones iniciales sobre quien hacer la comparación a la fotografía de especie que sea enviada. Esta es la manera general en la que funciona una máquina de aprendizaje automático, y a medida que se vaya ejecutando tareas de reconocimiento sobre esta y aumentando su base de fotografías, más precisión tendrá en los reconocimientos de patrones.

CAPÍTULO 4

4. ANÁLISIS DE RESULTADOS.

En este capítulo se hace un análisis general del funcionamiento de nuestro proyecto basándonos en pruebas de ejecución de cada una de las principales funcionalidades que forman parte de este. En cada prueba presentada se muestra el comportamiento de los módulos involucrados en la determinada funcionalidad. Por ejemplo, en la prueba de funcionalidad de “carga de imagen desde la APP al servidor” se muestra una captura de pantalla de la fotografía que aparece capturada en el dispositivo móvil pero también se muestra el archivo de fotografía en formato “.png” que tenemos en el servidor luego de ser cargada la foto. Por último, en cada prueba de funcionalidad se especifican dos cosas: aquello que se espera obtener, y los resultados reales obtenidos, dando pequeñas especificaciones sobre alguna variación entre estos dos aspectos. Las consideraciones de cambios, correcciones, y mejoras a las funcionalidades son expuestas en la sección de Conclusiones y Recomendaciones.

Es importante especificar que para utilizar una APP con tecnología *Android* en un dispositivo móvil lo idóneo es generar un archivo compilado en formato “.apk” y publicarlo en *Google Play Services,* para que pueda ser descargado por cualquier usuario. Luego, nosotros no lo hemos publicado por tratarse de un proyecto de prueba así que hemos descargado el archivo “.apk” desde un repositorio en internet. El archivo “.apk” lo hemos generado a partir de nuestro proyecto por medio de la funcionalidad “*Build APK*” de *Android Studio*. Para realizar las pruebas presentadas en este capítulo nos hemos servido de un dispositivo móvil SAMSUNG J5 modelo SM G570M, con sistema operativo *Android* 5.1 (*Lollipop*), API 22. Para extender la visualización del dispositivo móvil dentro de una pantalla en nuestra computadora se hizo uso del programa de escritorio: *Vysor*. La dirección IP del servidor en el momento de las pruebas fue: 13.65.250.121, pero estará sujeta a cambios mientras el proyecto se encuentre en fase de prueba.

A continuación se presentan las pruebas de ejecución realizadas sobre todo el sistema.

4.1 Listado de funcionalidades del proyecto a ser probadas

CASO DE ESTUDIO	ESCENARIO	MÓDULO DEL SISTEMA QUE PARTICIPA
Ejecución de la APP en un dispositivo móvil		
	Instalación de la APP	APP
	Autenticación de usuario por primera vez	APP y DATASET
	Sincronización de tablas: clasificacion y parte	APP y DATASET
Recolectar especies vegetales		
	Carga de fotografías al servidor	APP y DATASET
	Registro de datos de especie en base nativa de la APP	APP
Reconocer especies vegetales		
	Reconocimiento de flor Anturio	APP y DATASET
	Reconocimiento de flor Chavelita	APP y DATASE
	Reconocimiento de flor Marigold	APP y DATASET
Listar base nativa de la APP		
	Existen datos registrados por usuario	APP
Administrar *data set*		
	Autenticación de usuario al sitio web	WEB y DATASET
	Registro de nuevo usuario	WEB y DATASET
	Modificación de un registro en tabla “dataset”	WEB y DATASET

Tabla 4.1: Listado de funciones a probar

En la Tabla 4.1 se listan las especificaciones de funcionalidad del proyecto, catalogadas en 3 columnas: Casos de Uso, Escenarios involucrados, y módulos del proyecto que interactúan en la respectiva funcionalidad.

4.2 Caso de estudio: ejecución de la APP en un dispositivo móvil

Debemos realizar un pequeño proceso para la utilización de nuestra APP en un dispositivo móvil antes de su uso por primera vez, que consiste de 3 pasos: instalación de la APP en el dispositivo móvil, autenticación en la APP con un usuario previamente registrado en el sistema, y sincronización de datos entre el servidor y la APP. Este caso de estudio se basa en la prueba de que estos 3 escenarios sean exitosos y como resultado la APP quede instalada en el dispositivo y funcionando correctamente.

4.2.1 Escenario: instalación exitosa de la APP

El usuario descarga el archivo ".apk" desde internet, en la ventana de dialogo escoge instalar la aplicación. Luego, se presenta la página tipo *dialog* mostrada en la Figura 4.1 donde se muestra al usuario los recursos del dispositivo a cuales nuestra APP tendrá acceso con las opciones de "INSTALAR" o "CANCELAR", el usuario acepta instalar, y la APP se instala en el dispositivo móvil.

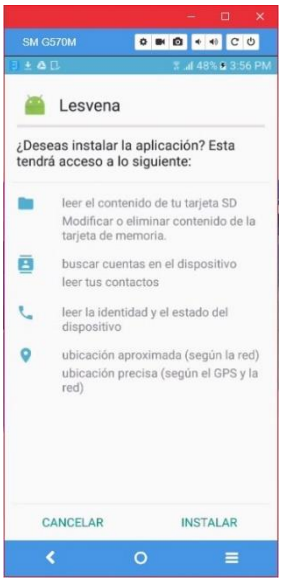


Figura 4.1: Captura de pantalla de la APP: pantalla de instalación de la APP en el dispositivo móvil

En la Figura 4.1 se presenta una captura de pantalla de la APP en el momento de su instalación en el dispositivo móvil. Podemos observar que la pantalla es de tipo *dialog* e informa al usuario sobre los recursos del dispositivo que serán usados por la APP, a la vez que requiere del usuario la concesión de los permisos necesarios para el uso de estos recursos.

4.2.2 Escenario: autenticación exitosa de usuario por primera vez

Luego de la instalación, al usar el usuario la APP por primera vez, se presenta la pantalla de autenticación de usuario. No se observó inconvenientes en esta funcionalidad.

- **Comportamiento del módulo APP**

Figura 4.2: Captura de pantalla de la APP: pantalla de autenticación (*login*) de usuario en la APP, por primera vez

En la Figura 4.2 se presenta una captura de la pantalla en la APP para autenticación de usuarios. Esta pantalla se presenta por única vez en la primera ocasión que el usuario use la APP en su dispositivo móvil puesto

que para el resto de transacciones de este usuario en este dispositivo móvil se usará un valor de *token* que resultará transparente para el mismo.

- **Comportamiento del módulo DATASET**

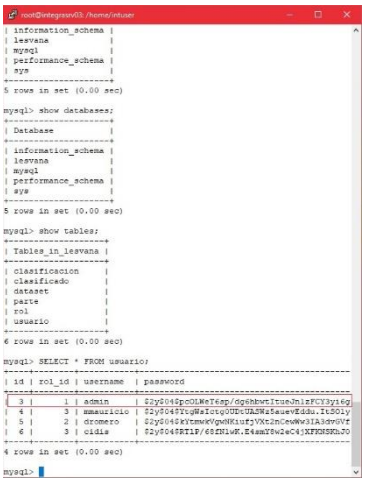


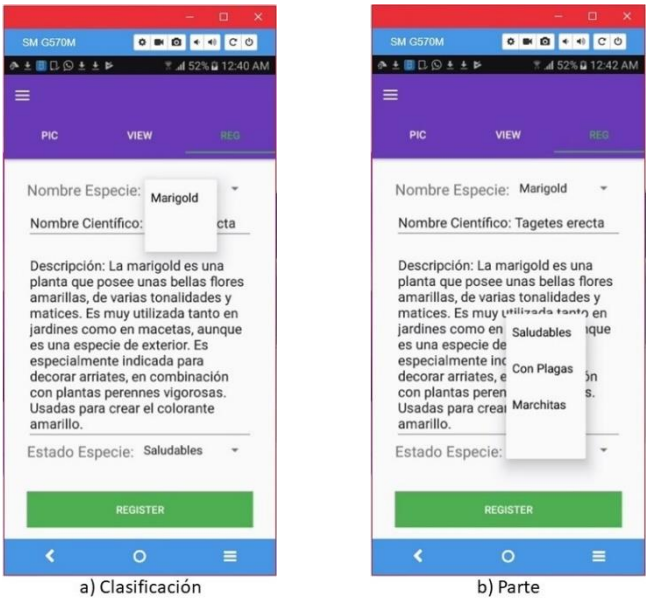
Figura 4.3: Captura de pantalla del servidor: resultado de consulta a la tabla “usuarios” de la base de datos “lesvana” (*data set*)

En la Figura 4.3 se presenta una captura de pantalla en el servidor, de los resultados obtenidos luego de realizar una consulta a la tabla “usuarios”. En esta captura de pantalla podemos identificar que existe el registro de usuario “admin” con el cual estamos autenticándonos en la APP. Este registro de la tabla “usuarios” en el servidor será usado por la API REST para corroborar la existencia del usuario ingresado en la APP y devolverle su correspondiente valor de *token*.

4.2.3 Escenario: sincronización exitosa de tablas: “clasificacion” y “parte”

El usuario se autentica con datos de usuario y contraseña existentes, está conectado a internet, se reciben los datos de tablas: “clasificacion” y “parte”, y los registros de estas tablas son cargados en los respectivos menú desplegables de la APP. No se observó inconvenientes en esta funcionalidad.

- **Comportamiento del módulo APP**



a) Clasificación b) Parte

Figura 4.4: Capturas de pantalla de la APP: pantallas (a) Clasificación y (b) Parte

En la Figura 4.4 se presentan dos capturas de pantalla, de la misma pestaña "*REGISTER*" para registro de especies en la APP. En pantalla (a) Clasificación se muestra el contenido en lista desplegable de tabla "clasificacion" al hacer clic en pestaña superior. En pantalla (b) Parte se muestra el contenido en lista desplegable de tabla "parte" al hacer clic en pestaña inferior.

- **Comportamiento del módulo DATASET**

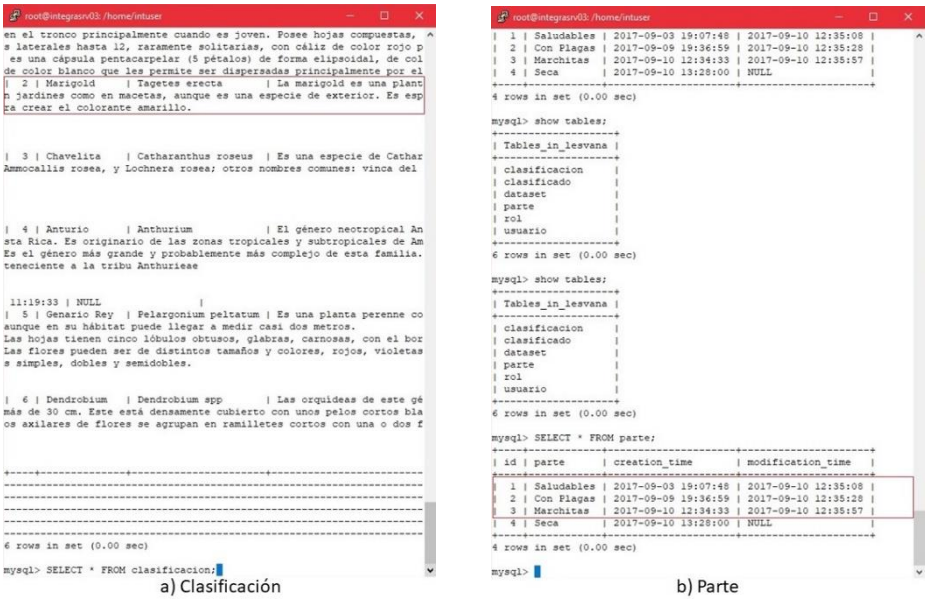


Figura 4.5: Capturas de pantalla del servidor: pantallas (a) Clasificación y (b) Parte

En la Figura 4.5 se presentan dos capturas de pantalla en el servidor. En pantalla (a) Clasificación se muestra los resultados obtenidos luego de consultar a la tabla “clasificacion”. En pantalla (b) Parte se muestra los resultados obtenidos luego de consultar a la tabla “parte”. En cada una de estas pantallas podemos verificar los datos de las tablas correspondientes en la base “lesvana” que son enviados a la APP en el momento de la sincronización inicial.

4.3 Caso de estudio: recolectar datos de especies vegetales

La recolección de fotografías de especies vegetales en nuestra APP consiste en 3 tipos de pantalla con una actividad específica cada una: “*PICTURE*”, “*VIEW*”, y “*REGISTER*”, y son cargadas conforme a la pestaña que el usuario seleccione, según se detalló en el capítulo 3. El proceso de captura de fotografías y datos de especies involucrado dentro de las 3 pestañas mencionadas, requiere de 2 escenarios: carga de fotografía al servidor y registro de datos de especie en base nativa. Este caso de estudio se basa en la prueba de que estos 2 escenarios sean exitosos y como resultado: el enlace URL de la fotografía en el servidor, y los datos escogidos por el usuario, queden registrados en la base de datos nativa en la APP.

4.3.1 Escenario: carga exitosa de fotografía al servidor

El usuario captura la fotografía, el usuario tiene internet, la fotografía se muestra en la pestaña "*VIEW*". Además, se verifica en el servidor y se constata que el archivo de fotografía fue cargado con éxito. Luego de la práctica pudimos constatar que la resolución de la foto capturada desde la APP es bastante baja para las finalidades del proyecto.

- **Comportamiento del módulo APP**

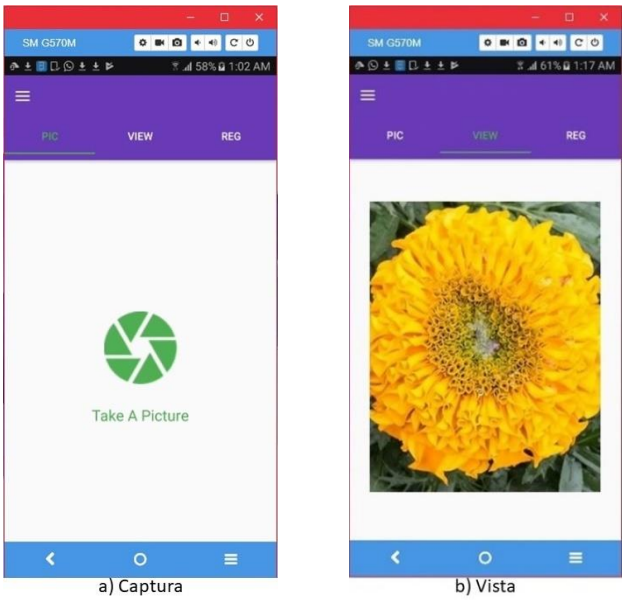


Figura 4.6: Capturas de pantalla de la APP: pantallas (a) Captura y (b) Vista

En la Figura 4.6 se presentan dos capturas de pantalla en la APP. En pantalla (a) Captura se presenta una captura de pantalla de la pestaña "*PICTURE*", en la cual al hacer clic en el ícono verde se muestra la cámara del dispositivo. En pantalla (b) Vista se presenta una captura de pantalla de la pestaña "*VIEW*", en la cual se muestra la fotografía capturada de la especie durante la prueba. Durante la captura y presentación de la

fotografía de la especie, la APP envío la fotografía al servidor y se obtuvo una URL de la ubicación en el servidor para dicha fotografía.

- **Comportamiento del módulo DATASET**

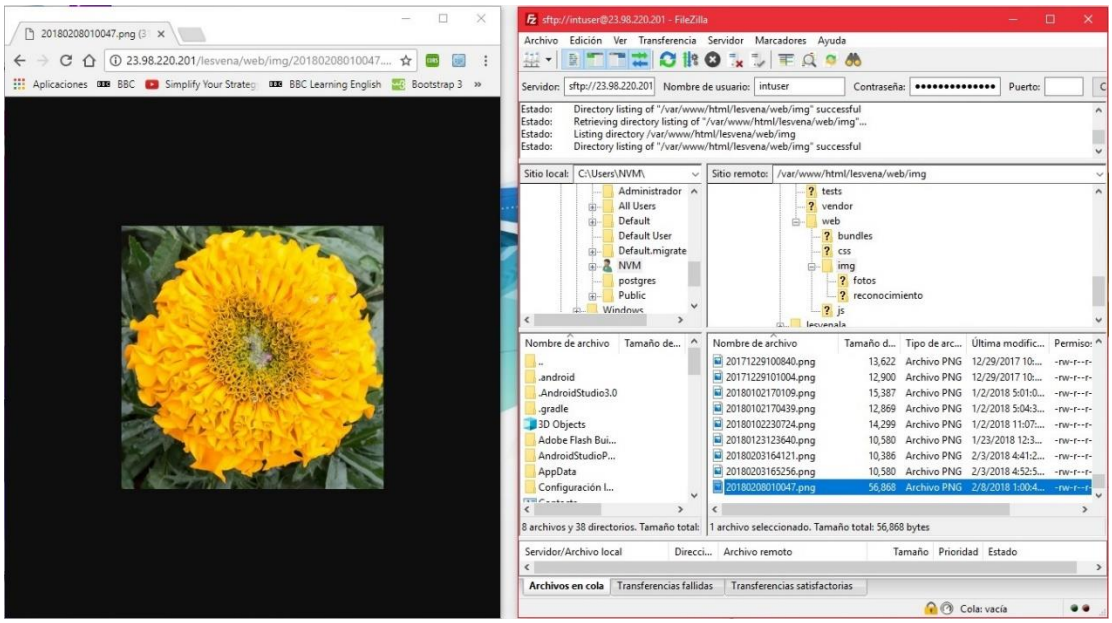


Figura 4.7: Fotografía cargada en el servidor: captura de pantalla del programa "*FileZilla*"

En la Figura 4.7 se presenta una captura de pantalla del programa "*FileZilla*" que muestra los archivos presentes en el servidor. Podemos observar que el archivo sombreado de azul es el archivo en formato ".png" y que corresponde a la fotografía recientemente cargada desde la APP. Para corroborar esta tarea podemos observar que la fecha de modificación de este archivo (sombreado) en el servidor es el mismo de la fecha de captura de la imagen en la APP (los minutos varían). Finalmente, tomamos el nombre del archivo de fotografía y lo adjuntamos a la URL del servidor, y presentamos a través de un navegador web (*Chrome*) la fotografía cargada en el servidor, la cual se presenta en la imagen de la izquierda.

4.3.2 Escenario: registro exitoso de datos de especie en base nativa de la APP

El usuario captura la fotografía, el usuario tiene internet, la fotografía se muestra en la pestaña "*VIEW*", y el usuario escoge los valores del menú desplegable y realiza el registro. Luego de la práctica pudimos constatar que hacen falta: una validación para guardar la fotografía sin necesidad de internet, y una mejor notificación al usuario sobre el evento de registro.

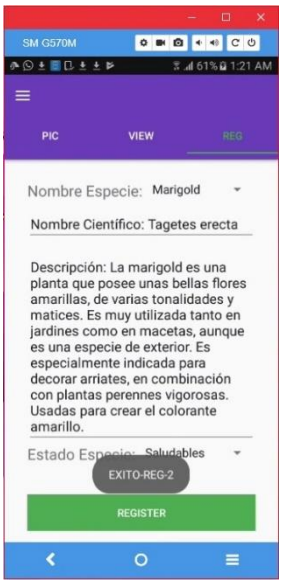


Figura 4.8: Captura de pantalla de la APP: pestaña "*REGISTER*"

En la Figura 4.8 se presenta una captura de pantalla de la pestaña "*REGISTER*" en la APP. La captura de esta pantalla fue realizada luego de haber hecho clic en el botón "*REGISTER*", y podemos observar un mensaje de éxito en el registro (mensaje en cuadro plomo), el cual además nos presenta el índice identificador (id) de registro guardado. Este evento guardó los valores de categoría y estado de especie, escogidos por el usuario, como un nuevo registro en la tabla "dataset" de la base de datos nativa de la APP.

4.4 Caso de estudio: reconocer especies vegetales

El proceso de reconocimiento de especies vegetales basados en su fotografía, en nuestra APP consiste en 5 tipos de pantalla con una actividad específica cada una: “*PICTURE*”, “*VIEW*”, “*RECOGNIZE*”, “*RECONIZING*”, y “*DETAILS*”, y son cargadas conforme a la pestaña que el usuario seleccione, según se detalló en el capítulo 3. El proceso de reconocimiento de especies involucrado dentro de las 5 pestañas mencionadas, requiere de 2 escenarios: carga de fotografía al servidor, y reconocimiento de especie en el servidor basado en la fotografía cargada. Debido a que en el numeral anterior ya se probó con éxito la carga de fotografía al servidor, en este caso de uso nos enfocamos en el escenario de reconocimiento.

Este caso de estudio se basa en la prueba de que este escenario de reconocimiento de especie vegetal sea exitoso y como resultado: se obtenga desde el servidor un listado de *strings* con información básica de las especies que se asemejan a la especie entregada, incluyendo una indicación del individuo de entre la lista que mayor porcentaje de semejanza tenga con la especie original. Luego, por ser esta una funcionalidad sensible, hemos realizado pruebas con 3 tipos diferentes de flor: Anturio, Chavelita, y Marigold, con 10 pruebas de individuos por cada tipo.

A continuación presentamos el resultado más representativo de entre las 10 pruebas de individuos por cada tipo de flor.

4.4.1 Escenario: reconocimiento de flor Anturio

El usuario captura la fotografía, el usuario tiene internet, la fotografía se muestra en la pestaña “*VIEW*”, el usuario presiona el botón reconocer en la pestaña “*RECOGNIZE*”, y se presentan los resultados del reconocimiento en diagrama de barras con flor Anturio como ganadora. Además, el usuario presiona el ícono de flecha alado de la especie y se presenta la descripción de la especie en pestaña “*DETAILS*”. Luego de la

práctica pudimos constatar que: se tiene poco detalle sobre cada especie del listado.

- **Comportamiento del módulo APP**

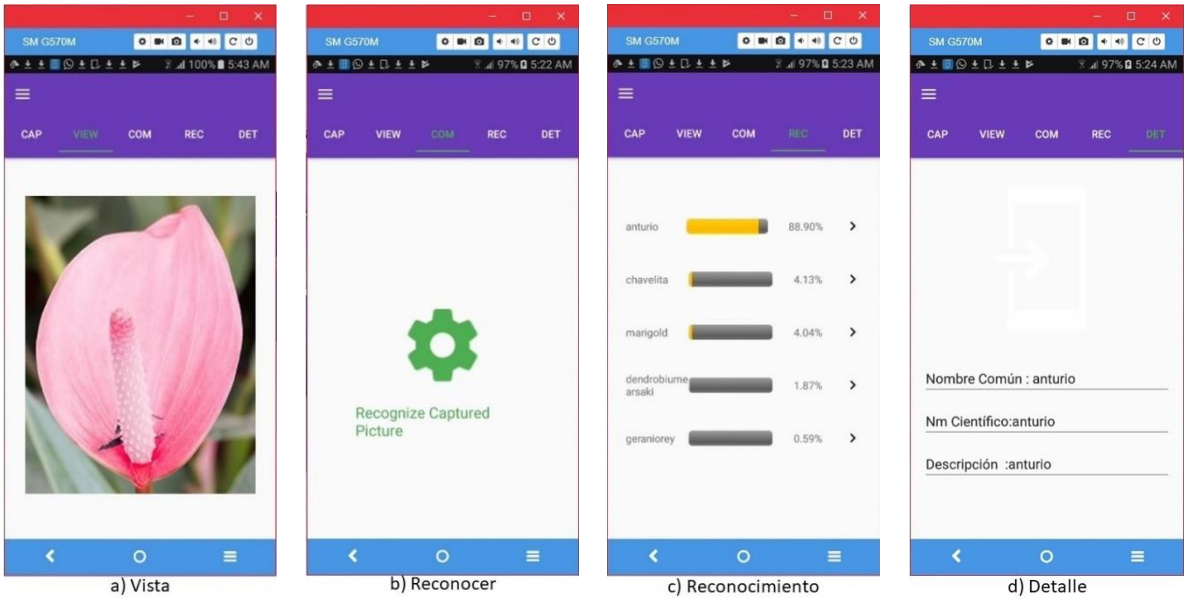


Figura 4.9: Capturas de pantalla de la APP: pantallas (a) Vista; (b) Reconocer; (c) Reconocimiento; (d) Detalle

En la Figura 4.9 se presentan cuatro capturas de pantalla en la APP. En pantalla (a) Vista se muestra una captura de pantalla de la pestaña "*VIEW*" que presenta la fotografía de la especie, capturada con el dispositivo móvil. En pantalla (b) Reconocer se muestra una captura de pantalla de la pestaña "*RECOGNIZE*", en la cual al hacer clic en ícono verde se ejecuta el proceso de reconocimiento de especie mediante enviar la fotografía capturada al servidor. En pantalla (c) Reconocimiento se muestra una captura de pantalla de la pestaña "*RECOGNIZING*", en la cual se presentan mediante un gráfico de barras los resultados del reconocimiento de la especie recibidos desde el servidor. La especie con mayor semejanza en este caso fue la flor Anturio. En pantalla (d) Detalle

se muestra una captura de pantalla de la pestaña “*DETAILS*”, la cual muestra pequeños datos de una especie determinada, la cual fue escogida de entre el listado anterior al hacer clic en el correspondiente ícono de flecha ubicado alado de cada elemento de la lista.

- **Comportamiento del módulo DATASET**

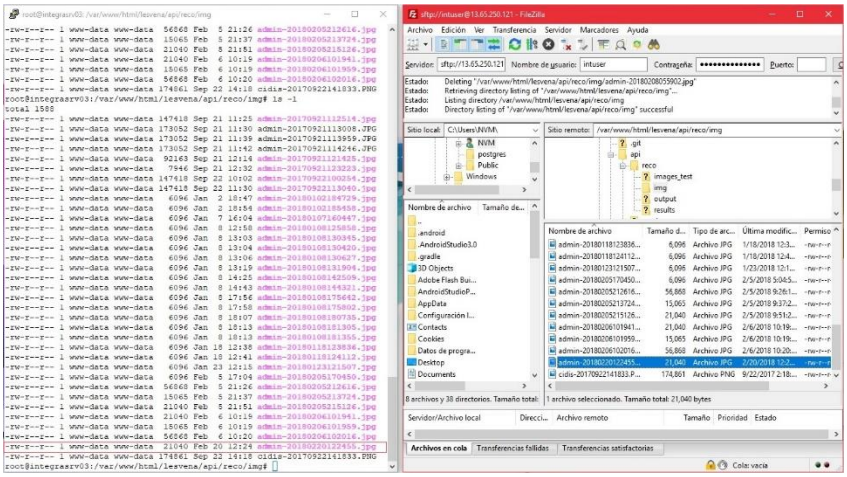


Figura 4.10: Fotografía cargada en el servidor: captura de pantalla del archivo de foto “.jpg” correspondiente a flor Anturio

En la Figura 4.10 se presenta una captura de pantalla del programa “*FileZilla*” que muestra los archivos presentes en el servidor. Podemos observar que el archivo sombreado de azul es el archivo en formato “.jpg” y que corresponde a la fotografía de flor Anturio recientemente cargada desde la APP. Adicionalmente, la imagen de la izquierda es la captura de pantalla del servidor, luego de listar los archivos del directorio: “/api/reco/img/” en el cual se agregan las fotos cargadas para reconocimiento. Podemos notar en el servidor el archivo de foto de flor Anturio recientemente agregado.

4.4.2 Escenario: reconocimiento de flor Chavelita

El usuario captura la fotografía, el usuario tiene internet, la fotografía se muestra en la pestaña "*VIEW*", el usuario presiona el botón reconocer en la pestaña "*RECOGNIZE*", y se presentan los resultados del reconocimiento en diagrama de barras con flor Chavelita como ganadora. Además, el usuario presiona el ícono de flecha alado de la especie y se presenta la descripción de la especie en pestaña "*DETAILS*". Luego de la práctica pudimos constatar que: se tiene poco detalle sobre cada especie del listado.

- **Comportamiento del módulo APP**

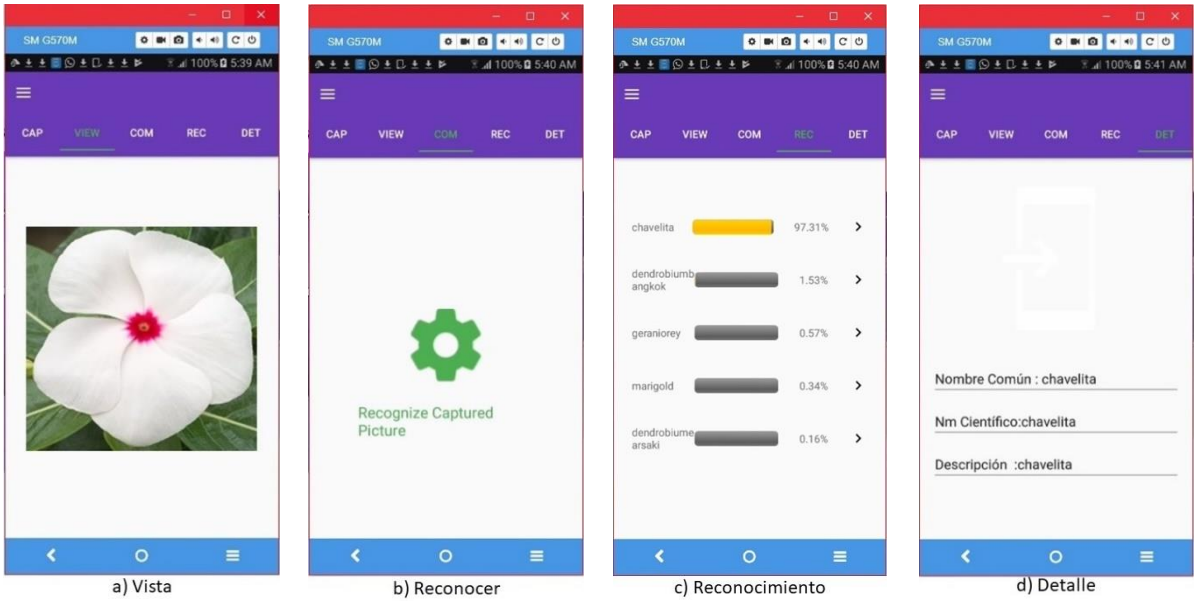


Figura 4.11: Capturas de pantalla de la APP: pantallas (a) Vista; (b) Reconocer; (c) Reconocimiento; (d) Detalle

En la Figura 4.11 se presentan cuatro capturas de pantalla en la APP. En pantalla (a) Vista se muestra una captura de pantalla de la pestaña "*VIEW*" que presenta la fotografía de la especie, capturada con el dispositivo

móvil. En pantalla (b) Reconocer se muestra una captura de pantalla de la pestaña "*RECOGNIZE*", en la cual al hacer clic en ícono verde se ejecuta el proceso de reconocimiento de especie mediante enviar la fotografía capturada al servidor. En pantalla (c) Reconocimiento se muestra una captura de pantalla de la pestaña "*RECOGNIZING*", en la cual se presentan mediante un gráfico de barras los resultados del reconocimiento de la especie recibidos desde el servidor. La especie con mayor semejanza en este caso fue la flor Chavelita. En pantalla (d) Detalle se muestra una captura de pantalla de la pestaña "*DETAILS*", la cual muestra pequeños datos de una especie determinada, la cual fue escogida de entre el listado anterior al hacer clic en el correspondiente ícono de flecha ubicado alado de cada elemento de la lista.

- **Comportamiento del módulo DATASET**

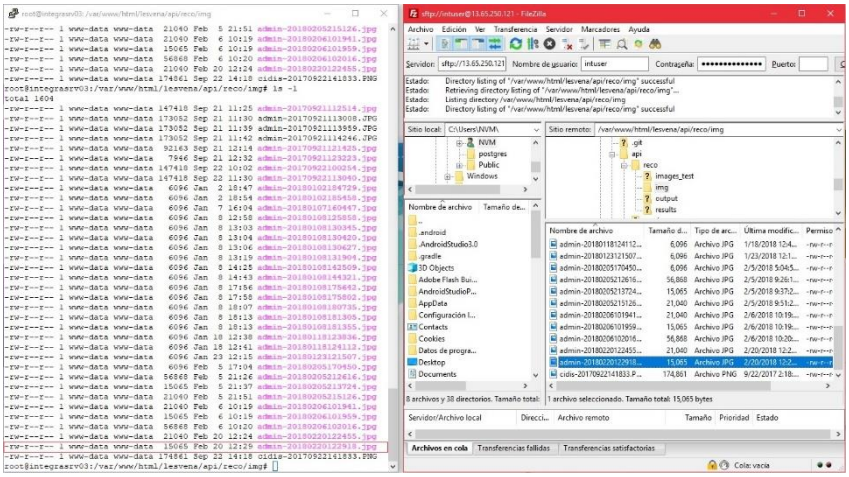


Figura 4.12: Fotografía cargada en el servidor: captura de pantalla del archivo de foto ".jpg" correspondiente a flor Chavelita

En la Figura 4.12 se presenta una captura de pantalla del programa "*FileZilla*" que muestra los archivos presentes en el servidor. Podemos observar que el archivo sombreado de azul es el archivo en formato ".jpg"

y que corresponde a la fotografía de flor Chavelita recientemente cargada desde la APP. Adicionalmente, la imagen de la izquierda es la captura de pantalla del servidor, luego de listar los archivos del directorio: "/api/reco/img/" en el cual se agregan las fotos cargadas para reconocimiento. Podemos notar en el servidor los archivos de foto de las flores Anturio y Chavelita recientemente agregados.

4.4.3 Escenario: reconocimiento de flor Marigold

El usuario captura la fotografía, el usuario tiene internet, la fotografía se muestra en la pestaña "*VIEW*", el usuario presiona el botón reconocer en la pestaña "*RECOGNIZE*", y se presentan los resultados del reconocimiento en diagrama de barras con flor Marigold como ganadora. Además, el usuario presiona el ícono de flecha alado de la especie y se presenta la descripción de la especie en pestaña "*DETAILS*". Luego de la práctica pudimos constatar que: se tiene poco detalle sobre cada especie del listado.

- **Comportamiento del módulo APP**

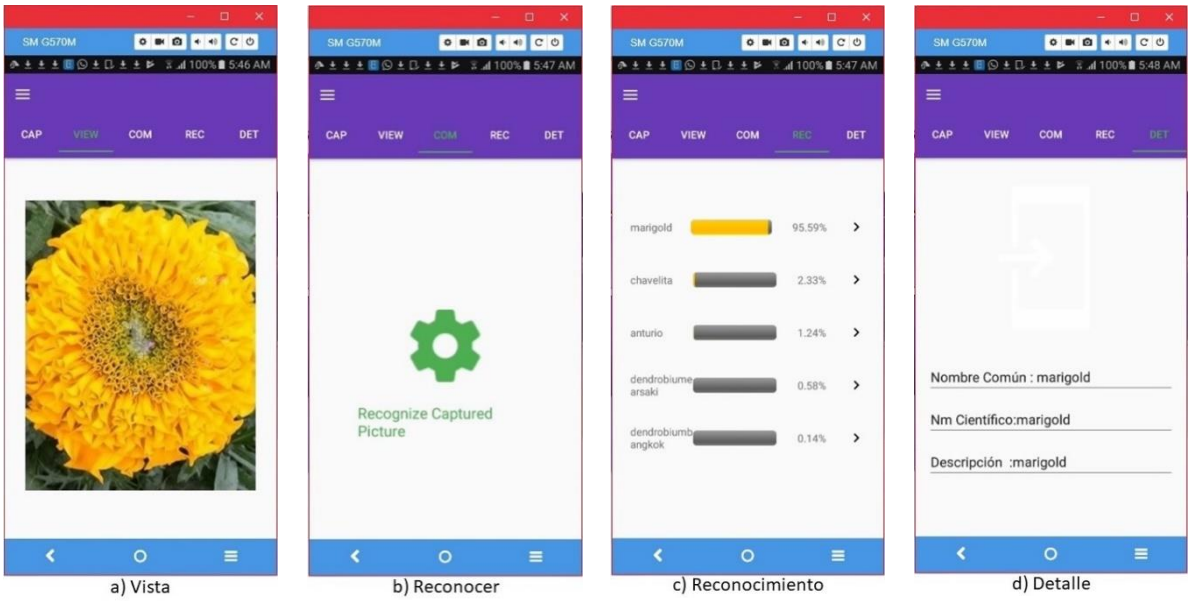


Figura 4.13: Capturas de pantalla de la APP: pantallas (a) Vista; (b) Reconocer; (c) Reconocimiento; (d) Detalle

En la Figura 4.13 se presentan cuatro capturas de pantalla en la APP. En pantalla (a) Vista se muestra una captura de pantalla de la pestaña “*VIEW*” que presenta la fotografía de la especie, capturada con el dispositivo móvil. En pantalla (b) Reconocer se muestra una captura de pantalla de la pestaña “*RECOGNIZE*”, en la cual al hacer clic en ícono verde se ejecuta el proceso de reconocimiento de especie mediante enviar la fotografía capturada al servidor. En pantalla (c) Reconocimiento se muestra una captura de pantalla de la pestaña “*RECOGNIZING*”, en la cual se presentan mediante un gráfico de barras los resultados del reconocimiento de la especie recibidos desde el servidor. La especie con mayor semejanza en este caso fue la flor Marigold. En pantalla (d) Detalle se muestra una captura de pantalla de la pestaña “*DETAILS*”, la cual muestra pequeños datos de una especie determinada, la cual fue escogida de entre el listado anterior al hacer clic en el correspondiente ícono de flecha ubicado alado de cada elemento de la lista.

- **Comportamiento del módulo DATASET**

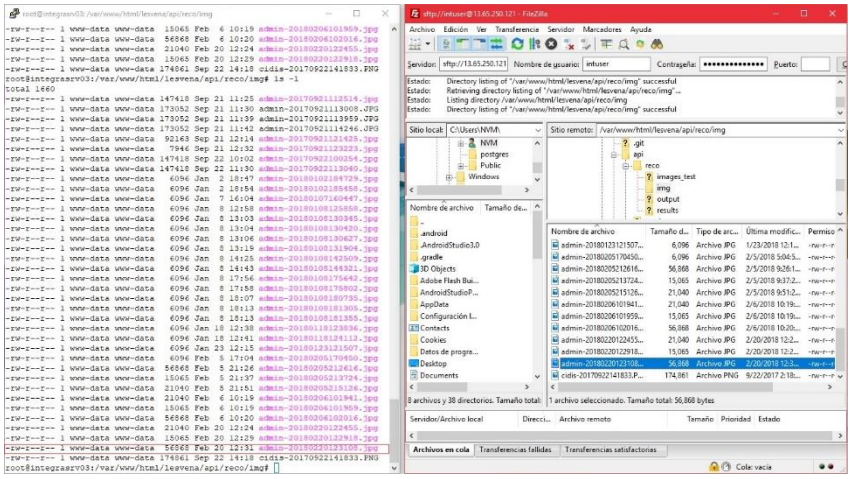


Figura 4.14: Fotografía cargada en el servidor: captura de pantalla del archivo de foto “.jpg” correspondiente a flor Marigold

En la Figura 4.14 se presenta una captura de pantalla del programa "*FileZilla*" que muestra los archivos presentes en el servidor. Podemos observar que el archivo sombreado de azul es el archivo en formato ".jpg" y que corresponde a la fotografía de flor Marigold recientemente cargada desde la APP. Adicionalmente, la imagen de la izquierda es la captura de pantalla del servidor, luego de listar los archivos del directorio: "/api/reco/img/" en el cual se agregan las fotos cargadas para reconocimiento. Podemos notar en el servidor los archivos de foto de las flores Anturio, Chavelita, y Marigold recientemente agregados.

4.5 Caso de estudio: listar base nativa de la APP

Para la presentación de los datos desde la base de datos nativa en la APP, sólo necesitamos de un escenario: existan datos registrados por el usuario en tabla "dataset". Para este efecto el usuario debe haber realizado el proceso de recolección de especies por al menos una vez a partir del momento de la instalación de esta aplicación. En resumen, este caso de estudio se basa en la prueba de que este escenario sea exitoso y como resultado se presente un listado de tipo *List Card View* de las especies registradas en la base de datos nativa, que han sido recolectadas por el usuario con anterioridad.

4.5.1 Escenario: existen datos registrados por el usuario

El usuario ya ha registrado datos en un proceso de recolección de especies, y al seleccionar la opción de *Gallery View* del menú principal, se presentan los datos registrados en la base. No se observó inconvenientes en esta funcionalidad

Figura 4.15: Captura de pantalla de la APP: listado de registros en la base nativa de la APP.

En la Figura 4.15 se muestra una captura de pantalla en la APP del listado de registros existentes en la base de datos nativa de la APP, traída desde la tabla “dataset”. Para presentar esta opción le damos clic a la opción *Gallery View* del menú principal.

4.6 Caso de estudio: administrar *data set*

Corroborar una gestión completa de los registros de la base de datos (*data set*) mediante una interfaz de página web involucra varios escenarios. Así pues, nos hemos enfocado en los 3 más representativos escenarios involucrados en la funcionalidad de nuestra página: autenticación de usuario al sitio web, ingreso de un nuevo registro en tabla “usuario”, y modificación de un registro existente en tabla “dataset”. Este caso de estudio se basa en la prueba de que estos 3 escenarios sean exitosos y como resultado los datos modificados y registros agregados a través de la página web, queden reflejados en la base de datos central del servidor.

4.6.1 Escenario: autenticación exitosa de usuario al sitio web

El usuario ingresa credenciales existentes de nombre de usuario y contraseña, y se le presenta la página web completa puesto que el rol implementado actualmente es de administrador. Luego de realizar la prueba en esta funcionalidad pudimos notar que se necesitan roles de usuarios: básico y experto.

- **Comportamiento del módulo WEB**

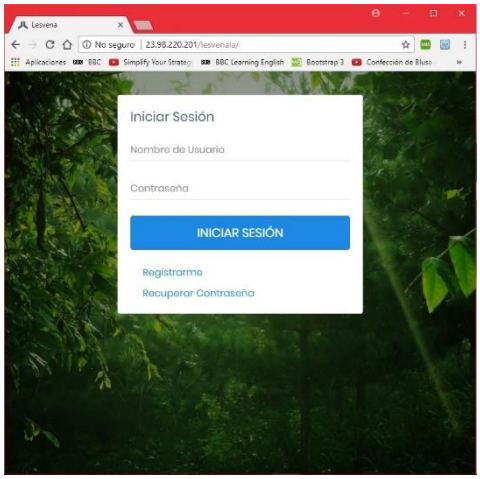


Figura 4.16: Captura de pantalla de la página web: pantalla de autenticación de usuario (*login*)

En la Figura 4.16 se muestra una captura de pantalla de la página de autenticación de usuarios en el sistema web. Si nos dirigimos a la dirección URL actual del proyecto: “http://13.65.250.121/lesvenala”, esta página de inicio será la primera página en presentarse. Podemos observar que tiene una pequeña sección de formulario, que permite el ingreso de datos de usuario y contraseña: si se ingresan los datos correctos, se nos dirige al sistema web completo (o acoplado dependiendo del rol de usuario), caso contrario se presenta un mensaje de datos ingresados erróneos.

- **Comportamiento del módulo DATASET**

Figura 4.17: Captura de pantalla del servidor: resultado de consulta a la tabla “usuarios” de la base de datos “lesvana” (*data set*)

En la Figura 4.17 se presenta una captura de pantalla en el servidor, de los resultados obtenidos luego de realizar una consulta a la tabla “usuarios”. En esta captura de pantalla podemos identificar que existe el registro de usuario “admin” con el cual nos autenticamos para efectos de realizar esta prueba.

4.6.2 Escenario: registro exitoso de nuevo usuario

El usuario da clic en botón “Crear usuario”, se presenta la página con el formulario para ingreso de datos, el usuario da clic en botón “Registrar” luego de agregar los datos, y se agrega el nuevo registro enviándonos a la página de listado de usuarios. No se observó inconvenientes en esta funcionalidad.

- **Comportamiento del módulo WEB**

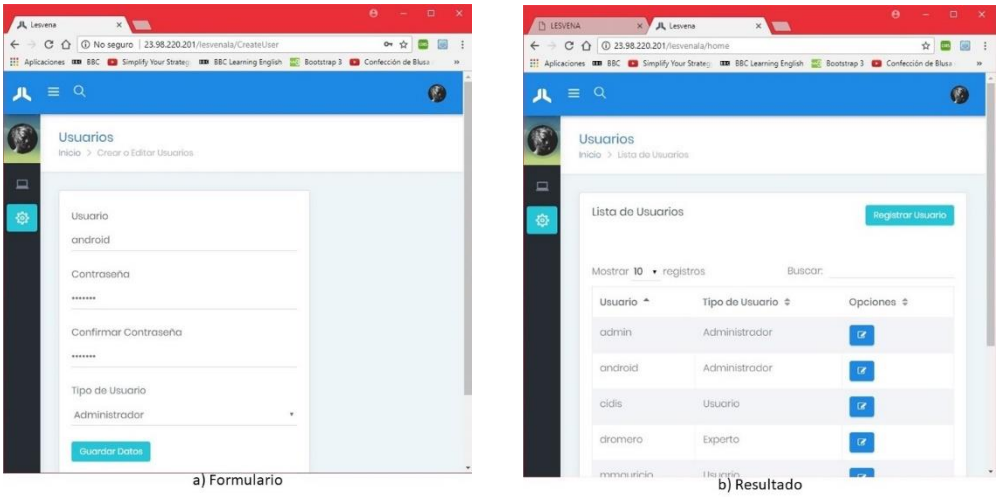


Figura 4.18: Capturas de pantalla de la página web: pantallas (a) Formulario y (b) Resultado

En la Figura 4.18 se muestran capturas de pantalla de las páginas para ingreso de un nuevo registro de usuario. En pantalla (a) Formulario se muestra una captura de pantalla de la página con el formulario para ingreso de datos para un nuevo registro. En pantalla (b) Resultado se muestra una captura de pantalla de la página de listado de usuarios, en la cual podemos observar el nuevo registro de prueba (usuario: “android”) que hemos ingresado.

- **Comportamiento del módulo DATASET**

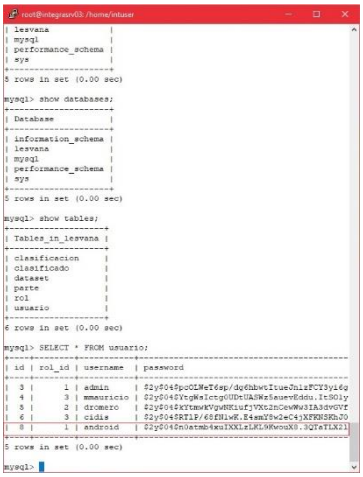


Figura 4.19: Captura de pantalla del servidor: resultado de consulta a la tabla “usuarios” de la base de datos “lesvana” (*data set*)

En la Figura 4.19 se presenta una captura de pantalla en el servidor, de los resultados obtenidos luego de realizar una consulta a la tabla “usuarios”. En esta captura de pantalla podemos observar que ahora existe un registro adicional a los que existían originalmente y que hemos observado en figuras anteriores de este capítulo. De hecho, el registro nuevo tiene los datos de usuario: “android”, que es precisamente el que acabamos de agregar mediante la página web.

4.6.3 Escenario: modificación exitosa de un registro en tabla “dataset”

El usuario se dirige a la página de listado de especies (datos de tabla “dataset”), da clic en botón con ícono de modificación de un determinado elemento de la lista, se presenta la página con formulario para modificación de datos, el usuario da clic en botón “Registrar” luego de modificar alguno de los valores, y se modifica el registro escogido enviándonos a la página de listado de especies. No se observó inconvenientes en esta funcionalidad.

- **Comportamiento del módulo WEB**

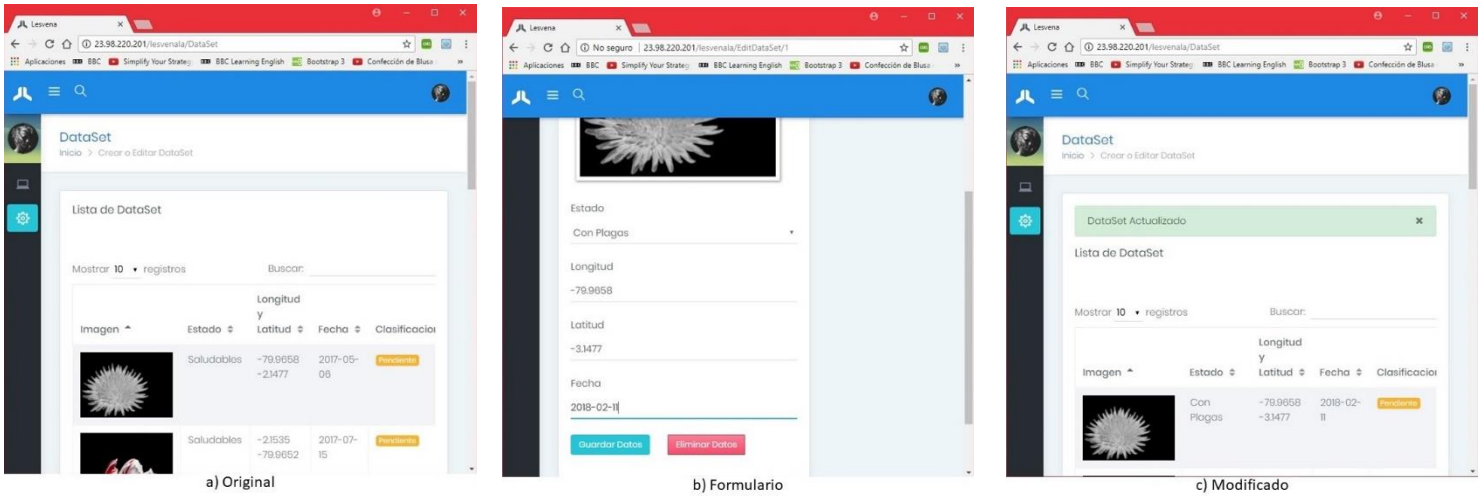
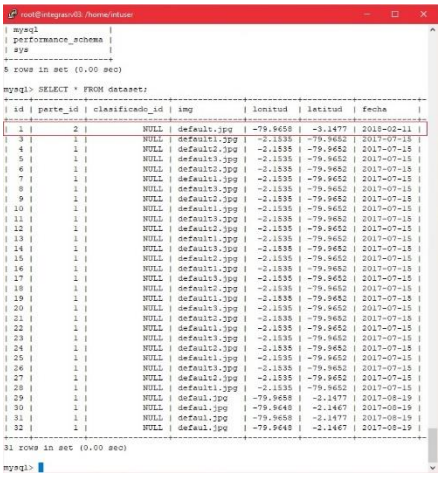


Figura 4.20: Capturas de pantalla de la página web: pantallas (a) Original, (b) Formulario, y (c) Modificado

En la Figura 4.20 se muestran capturas de pantalla de las páginas para modificación de un registro existente de usuario. En pantalla (a) Original se muestra una captura de pantalla de la página de listado de especies (datos de tabla “dataset”), de la cual el primer registro lo vamos a modificar. En pantalla (b) Formulario se muestra una captura de pantalla de la página con el formulario para modificación de datos del registro seleccionado (primer registro de la lista, en orden ascendente por id), en la cual cambiamos valores en los campos: estado y fecha. En pantalla (c) Modificado se muestra una captura de pantalla de la página de listado de especies, cuyos valores de estado y fecha han sido cambiados.

- **Comportamiento del módulo DATASET**



```
root@integrausr03: /home/intiuser
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM dataset;
+----+----------+----------------+-----------+----------+----------+---------------------+
| id | parte_id | clasificado_id | img       | lonitud  | latitud  | fecha               |
+----+----------+----------------+-----------+----------+----------+---------------------+
|  1 |        2 |           NULL | default.jpg | -79.9658 |  -3.1477 | 2018-02-11 |
|  3 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
|  4 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
|  5 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
|  6 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
|  7 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
|  8 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
|  9 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 10 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 11 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 12 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 13 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 14 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 15 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 16 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 17 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 18 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 19 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 20 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 21 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 22 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 23 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 24 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 25 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 26 |        1 |           NULL | default3.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 27 |        1 |           NULL | default2.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 28 |        1 |           NULL | default1.jpg |  -2.1535 | -79.9652 | 2017-07-15 |
| 29 |        1 |           NULL | defaul.jpg   | -79.9658 |  -2.1477 | 2017-08-19 |
| 30 |        1 |           NULL | defaul.jpg   | -79.9648 |  -2.1467 | 2017-08-19 |
| 31 |        1 |           NULL | defaul.jpg   | -79.9658 |  -2.1477 | 2017-08-19 |
| 32 |        1 |           NULL | defaul.jpg   | -79.9648 |  -2.1467 | 2017-08-19 |
+----+----------+----------------+-----------+----------+----------+---------------------+
31 rows in set (0.00 sec)

mysql>
```

Figura 4.21: Captura de pantalla del servidor: resultado de consulta a la tabla “dataset” de la base de datos “lesvana” (*data set*)

En la Figura 4.21 se presenta una captura de pantalla en el servidor, de los resultados obtenidos luego de realizar una consulta a la tabla “dataset”. En esta captura de pantalla podemos observar, en el primer registro de arriba a abajo, que los datos de estado y fecha concuerdan con los datos modificados mediante la página web, y que son presentados en la Figura 4.20.

CONCLUSIONES Y RECOMENDACIONES

Habiendo terminado el proyecto, y habiendo cubierto los objetivos propuestos al inicio del mismo, describimos a continuación las conclusiones y recomendaciones obtenidas al término de la implementación y puesta a prueba del sistema en su totalidad.

Se ha hecho uso de la tecnología de vanguardia y de los principales conocimientos aprendidos en la carrera tales como: Ingeniería de Software, Desarrollo de Aplicaciones Web, o Inteligencia Artificial, para resolver un problema de la vida real, en este caso levantamiento de información sensible y remota.

Se ha aprovechado de un implemento cotidiano en la actualidad como lo es un dispositivo móvil para aportar un beneficio a la comunidad por medio de la ayuda en la preservación de las especies nativas.

Se ha habilitado una opción de recopilación de información de especies vegetales nativas desde lugares remotos como fincas, pueblos, o lugares apartados del perímetro urbano.

Finalmente, luego de las pruebas realizadas podemos corroborar que se tiene construida una aplicación con tecnología *Android* que funciona en dispositivos móviles, la cual, permite recolectar datos de especies vegetales por medio de la captura de fotografías y registro de sus datos en base nativa, permite reconocer una especie vegetal basándonos en su fotografía (capturada con la aplicación), mostrar información de la especie reconocida, listar los datos de especies que han sido recolectadas por el usuario, y administrar la *data set* por medio del sistema web.

A manera de recomendaciones especificamos las observaciones que tuvimos en cuanto al funcionamiento del proyecto. Además, comentamos sobre las posibles mejoras que consideramos que pueden ser realizadas al sistema.

El alcance del proyecto no contempló la implementación de roles en la página web, de esta manera nuestra primera recomendación es que sea implementada en un futuro esta característica en el sitio web, tal que del rol actual: Administrador, existan 2 roles adicionales: Básico y Experto.

Se debe conseguir y mantener un usuario físico colaborador permanente para el sitio web con el rol de Experto, tal que clasifique las fotografías que sean subidas desde la APP, y de esta forma depurar los datos y fotografías que enviamos a reconocimiento a la máquina de aprendizaje automático.

Por motivo del poco tiempo que se dispone en la Materia Integradora, no se alcanzó a culminar todas las validaciones suficientes para el proyecto. Por ende, recomendamos las siguientes mejoras en cuanto a la funcionalidad del sistema: mejorar la calidad de las fotografías que son guardadas en el proceso de recolección mejorando los algoritmos de compresión de estas; implementar una validación que verifique la disponibilidad de internet luego de la captura de la foto de especie, si no hay guarde la foto en memoria interna y se haga una sincronización posterior; implementar una funcionalidad de lanzamiento automático entre pestañas de la APP; y realizar más pruebas para calibrar las validaciones necesarias para mantener al sistema suficientemente estable durante su ejecución en un dispositivo móvil.

BIBLIOGRAFÍA

[1] A. Carrasco V., “Cuarto Informe Nacional para el Convenio sobre la Diversidad Biológica” Min. Ambiente, Quito, Ecuador, ISBN: 978-9978-92-823-3, enero, 2010.

[2] *Cornell Lab. of Ornithology* (2017, Julio 14). *Celebrate Urban Birds* [Online]. Disponible en:
https://celebrateurbanbirds.org/

[3] R. B. Gardens and D. Trust (2017, Julio 17). *The NSW Plant Information Network System* [Online]. Disponible en:
http://plantnet.rbgsyd.nsw.gov.au

[4] J. R. Seiler and J. A. Peterson (2017, Julio 17). *Virginia Tech Dendrology* [Online]. Disponible en:
http://dendro.cnre.vt.edu/dendrology/main.htm

[5] C. Supe. Inv. Científicas (2017, Julio 21). ARBOLAPP [Online]. Disponible en:
http://www.arbolapp.es/

[6] D. A. Silva, B. Mercerat (2002, Enero 29). Construyendo aplicaciones web con una metodología de diseño orientada a objetos [Online]. Disponible en:
https://ldc.usb.ve/~abianc/electivas/OOHDM.pdf

[7] FedeProEx (2011, Noviembre 14). Cómo añadir datos precargados en tus aplicaciones [Online]. Disponible en:
http://androcode.es/2013/01/como-anadir-datos-precargados-en-tus-aplicaciones/

[8] HearTom (2016, Febrero 5). Tomar foto y mostrar en *ImageView* [Online]. Disponible en:
https://www.youtube.com/watch?v=l3MHEjy5A2k

[9] Arshu (2013, Octubre 14). *By using this code we can setup MapView anywhere* ... [Online]. Disponible en: https://stackoverflow.com/questions/19353255/how-to-put-google-maps-v2-on-a-fragment-using-viewpager

[10] Android Developer (2017, Marzo 1). *Navigation Drawer with Swipe Tabs* [Online]. Disponible en: https://www.youtube.com/watch?v=nwRxjJefcaI

[11] M. Arteaga (2017, Marzo 23). RE: ¿Cómo subir una imagen a servidor desde Android? [Online]. Disponible en: https://es.stackoverflow.com/questions/57336/c%C3%B3mo-subir-una-imagen-a-servidor-desde-android

[12] J. Powell (2014, Noviembre 14). RE: *Android - loopJ AsyncHttpClient* ... [Online]. Disponible en: https://stackoverflow.com/questions/26938212/android-loopj-asynchttpclient-return-response-onfinish-or-onsuccess

[13] Convertio (2017, Septiembre 7). Convertidor de vectores en línea [Online]. Disponible en: https://convertio.co/es/vector-converter/