UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso de Profesionales en Actividad

TÍTULO: Implementación del Sistema de Gestión del Documento Único Equino del Ministerio de Agroindustria de la Provincia de Buenos Aires

AUTOR: Manuel Olegario Becerra

DIRECTOR ACADÉMICO: Lisandro Delía

DIRECTOR PROFESIONAL: Marcelo Hermigarate

CARRERA: Licenciatura en Sistemas

Resumen

A partir de la sanción de la Ley Nº 13.627, se implementó en toda la provincia de Buenos Aires, con carácter obligatorio, el Documento Único Equino (DUE) para la identificación y traslado de la especie equina. Por tales motivos, surge la necesidad de desarrollar e implementar un Sistema Informático de Gestión del Documento Único Equino, del Ministerio de Agroindustria de la Provincia de Buenos Aires (MAIBA). La presente tesina está enfocada principalmente en el diseño, desarrollo e implementación de la aplicación móvil, una de las cuales integra dicho Sistema.

Palabras Clave

Documento Único Equino, aplicación móvil, aplicación multiplataforma, Android, React Native, MobX.

Conclusiones

En lo relativo al ámbito técnico-profesional, la participación en este proyecto me permitió conocer y poner en práctica React Native como herramienta para el desarrollo de aplicaciones móviles multiplataforma. Por otra parte, haber formado parte de las capacitaciones brindadas al usuario final en lo referido al uso de la aplicación, y haberlos acompañado en sus primeras interacciones con la misma, me permitió conocer cómo funciona la app en una práctica real y también enriquecerme por el feedback directo de los mismos.

Trabajos Realizados

Inicialmente, se analizó la información recabada por el equipo de analistas funcionales y se procedió a elaborar el diseño y prototipado de la aplicación móvil multiplataforma. A continuación, se seleccionaron las herramientas a utilizar para el desarrollo de la misma. Por último, se procedió con la compilación de la aplicación móvil para Android y se la publicó en su respectiva plataforma de distribución digital. Por otra parte, se realizaron capacitaciones a los usuarios en el uso de dicha aplicación.

Trabajos Futuros

Se proponen diversos puntos de mejora y trabajos a futuro sobre la aplicación móvil desarrollada. La principal mejora pendiente es el desarrollo y publicación de la aplicación móvil para iOS. Otro de los aspectos a mejorar es la interacción del usuario con la app, por ejemplo, al indicar características en los diagramas SVG. Además, se plantean nuevas funcionalidades, como es el módulo de denuncias o la consulta de la información de un DUE a partir de la lectura de un microchip.

Fecha de la presentación: Noviembre 2019

AGRADECIMIENTOS

En primer lugar, quiero agradecerle a la Facultad de Informática, por la posibilidad de realizar mi tesina de Licenciatura en Sistemas a través del programa PAEPA;

a Leandro y Marcelo, quienes confiaron en mí para el desarrollo de la aplicación móvil y me permitieron utilizarla como objeto de la presente;

a mi director académico Lisandro Delía, por dirigirme y hacer esta tesina posible;

a Augusto Villa Monte, por su predisposición, consejo y conocimiento brindado;

a mis compañeros, por su apoyo ante mis consultas y por su ayuda durante el desarrollo de la aplicación;

a Yani, por el amor y la inteligencia.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ACRÓNIMOS

API Application Programming Interface

CLI Command-Line Interface (Interfaz de Línea de Comandos)

CSS Cascading Style Sheets

DOM Document Object Model (Modelo de Objetos del Documento)

DT-e Documento de Tránsito Electrónico

DUE Documento Único Equino

HTML HyperText Markup Language (Lenguaje de Marcas de Hipertexto)

IDE Integrated Development Environment (Entorno de Desarrollo Integrado)

ISO International Organization for Standardization

JDK Java Development Kit

JSON JavaScript Object Notation (notación de objeto de JavaScript)

JSX JavaScript Syntax eXtension

KHz Kilohertz

LTS Long Term Support

MAIBA Ministerio de Agroindustria de la Provincia de Buenos Aires

MVC Model-View-Controller (Modelo-Vista-Controlador)

MVVM Model-View-ViewModel (Modelo–Vista–Modelo de vista)

RENSPA Registro Nacional Sanitario de Productores Agropecuarios

SDK Software Development Kit (Kit de Desarrollo de Software)

SENASA Servicio Nacional de Sanidad y Calidad Agroalimentaria

SPC Sangre Pura de Carrera

SVG Scalable Vector Graphics

UI User Interface (Interfaz de Usuario)

UX User Experience (Experiencia de Usuario)

WS Web Services

WSDL Web Services Description Language

1

INTRODUCCIÓN

En esta sección se presenta la motivación de esta tesina, y se detallan: el objetivo, los desarrollos propuestos y los resultados esperados. Finalmente, se describe la estructura de la misma.

1.1. Motivación

En la actualidad, existen en la Argentina más de dos millones y medio de caballos. La distribución de éstos, según su raza y actividad, es extraordinariamente variada, existiendo ejemplares de la mayoría de las razas reconocidas en el mundo, como así también una gran variedad de animales mestizos (no inscriptos en un registro de criadores) que se utilizan para los más diversos fines.

Entre las distintas actividades reconocidas, se pueden mencionar: caballos deportivos, ya sea de raza pura o mestizos; caballos de eventos tradicionalistas y destreza criolla; los de trabajo, ya sea en el ámbito rural o urbano; los de paseo; los de fuerzas armadas y seguridad; los utilizados en elaboración de medicamentos; los reproductores de todas las razas; aquellos que se envían a remates/ferias y finalmente los que se envían a faena (unos 200.000 al año).

En lo que respecta a la provincia de Buenos Aires, actualmente existen más de setecientos mil equinos. A partir de la sanción de la Ley N° 13.627 [1], se implementó en todo el territorio provincial, con carácter obligatorio, el Documento Único Equino (DUE) para la

identificación y traslado de la especie equina, reemplazando la Guía Única de Traslado provista por la Ley N° 10.891 [3] y el Código Rural de la Provincia de Buenos Aires – Decreto Ley N° 10.081/1983 [4] y modificatorias. La puesta en marcha de esta ley conlleva la generación de varios registros, entre ellos, el de Identificación Equina, el de Médicos Veterinarios Privados y Oficiales Habilitados, el de Funcionarios Municipales Autorizados y el de Proveedores de Dispositivos de Identificación Animal.

Por tales motivos, surge la necesidad de desarrollar e implementar un Sistema Informático de Gestión del Documento Único Equino, del Ministerio de Agroindustria de la Provincia de Buenos Aires (MAIBA).

1.2. Objetivos

El objetivo general es analizar, diseñar, desarrollar e implementar una plataforma informática para la gestión del DUE y de sus registros relacionados.

La presente tesina estará enfocada principalmente en el diseño, desarrollo e implementación de la aplicación móvil, una de las cuales integra el Sistema Informático de Gestión del Documento Único Equino. Para contextualizar dicho enfoque, a continuación, se mencionan cada una de las etapas del ciclo de vida del proyecto:

- **Relevamiento de información**: La etapa de relevamiento de información será realizada por el equipo de analistas funcionales, mediante entrevistas con el personal del Ministerio de Agroindustria, quedando fuera del alcance de la presente tesina.
- **Análisis de información y diseño del sistema:** La información recabada por los analistas funcionales en la etapa anterior será analizada y organizada según los criterios convenidos con el MAIBA, poniendo énfasis en la mirada del encuestado y en su conocimiento sobre los procesos. Utilizando como base dicho análisis, se procederá a elaborar el diseño y prototipado de la aplicación móvil.

- **Desarrollo del software:** En base al diseño y prototipado generado en la etapa anterior, se procederá al desarrollo de los diferentes módulos que conformarán la aplicación.
- **Testing:** Se realizarán las pruebas funcionales y no funcionales pertinentes y se ajustará aquello que corresponda, según los resultados de las mismas. Estas pruebas, como así también la conformación de manuales de ayuda, será realizada por un equipo específico, quedando fuera del alcance de la tesina.
- **Implementación y capacitación:** Se procederá a la implementación definitiva de la aplicación, contemplando tanto la transferencia de conocimiento al área técnica del MAIBA, a través de capacitaciones, como el acompañamiento a los veterinarios para el correcto uso del sistema desarrollado.

1.3. Desarrollos propuestos

Se propone desarrollar un sistema que permita la correcta implementación del DUE, conformado por dos grandes aplicaciones: la aplicación Web, previamente realizada por el equipo de desarrolladores, y la aplicación móvil, la cual será el foco principal de la presente tesina:

- **Aplicación Web:** Será desarrollada e integrada a la Plataforma Web existente del MAIBA como un nuevo subsistema, y estará destinada principalmente a centralizar la administración de los DUE. También gestionará la información personal de los veterinarios habilitados para realizar DUE. Administrará la información personal de los propietarios de los equinos y gestionará a los proveedores de microchips habilitados por el MAIBA. Se controlarán las solicitudes de DUE por parte de los propietarios y las características equinas (razas, pelajes y particularidades).
- **Aplicación móvil:** Estará destinada a la identificación equina y posterior generación del DUE por parte de los veterinarios habilitados. La carga de los datos podrá ser

realizada de forma offline, sincronizándose manualmente con la plataforma Web cuando se cuente con conexión a Internet. Dicha identificación no podrá ser realizada desde la aplicación Web, por lo que la instalación y uso de la aplicación móvil será necesaria.

1.4. Resultados esperados

Como resultado general se espera el desarrollo y la implementación exitosa del Sistema de Gestión del Documento Único Equino del Ministerio de Agroindustria de la Provincia de Buenos Aires, aplicando e integrando tanto los conocimientos adquiridos a lo largo de la carrera de Licenciatura en Sistemas, como las experiencias e investigaciones realizadas previamente en el ámbito laboral.

Los resultados particulares esperados varían según cada una de las etapas del proceso de diseño e implementación de la aplicación móvil:

Análisis de información y diseño del sistema: Como resultado de esta etapa se espera generar el diseño y prototipado de la aplicación móvil:

- Análisis completo de los requerimientos recabados por el equipo de analistas.
- Configuración y puesta a punto del framework móvil.
- Prototipos de la aplicación móvil (mockups).

Desarrollo del software: Como resultado de esta etapa se espera desarrollar la aplicación móvil en su totalidad, contemplando principalmente los siguientes módulos:

- Módulo de autenticación.
- Módulo de solicitudes.
- Módulo de DUE.
- Módulo de sincronización con lectoras y lectura de microchip.

- Módulo de validación de chips.
- Módulo de sincronización de datos con la plataforma Web.

Implementación del software y capacitación a usuarios: Los resultados esperados de esta etapa son:

- Publicación de la aplicación móvil en las distintas plataformas de distribución digital.
- Capacitación a los veterinarios en el uso de la aplicación móvil.
- Descarga y utilización de la app por parte de los veterinarios en un entorno productivo, generando DUE reales.
- Corrección de errores en la aplicación móvil en caso que los hubiere.

1.5. Estructura del documento

Para finalizar este primer capítulo, se especifica en este apartado la estructura general del documento:

Capítulo 1: Se presentó la motivación de esta tesina, y se detallaron el objetivo, los desarrollos propuestos y los resultados esperados de la misma.

Se definió el enfoque de la presente tesina respecto al alcance general del Sistema de Gestión del Documento Único Equino del MAIBA.

Capítulo 2: Se resume el marco legal que encuadra e impulsa el desarrollo del Sistema Informático de Gestión del Documento Único Equino del MAIBA.

Capítulo 3: Se mencionan los documentos sobre los cuales se basó el diseño, prototipado y desarrollo de la aplicación móvil. Se detallan los módulos que la conforman.

Capítulo 4: Se describen las herramientas utilizadas para el desarrollo del Sistema Informático de Gestión del Documento Único Equino, haciendo hincapié en las usadas para la aplicación móvil y su integración con aquel.

Capítulo 5: Se detallan los prototipos realizados para la aplicación móvil y se mencionan los fundamentos teóricos aplicados para hacerlos.

Capítulo 6: Se detalla el proceso de desarrollo de la aplicación móvil, desde la creación y configuración del proyecto hasta su compilación y subida a las plataformas de distribución digital.

Capítulo 7: Para finalizar este trabajo, se presentan las conclusiones y se detallan algunas líneas de desarrollo futuras que permitirían incluir nuevas funcionalidades a la aplicación o mejorar algunas de las ya existentes.

2

MARCO CONTEXTUAL

2.1. Introducción

A modo introductorio, se resume el marco legal que encuadra e impulsa el desarrollo del Sistema Informático de Gestión del Documento Único Equino del MAIBA.

Se detalla particularmente la Ley N° 13.627, la cual implementa en todo el territorio de Buenos Aires con carácter obligatorio el DUE, para la identificación y traslado de la especie equina (reemplazando la antigua ley de marcas y señales), y el Decreto reglamentario N° 1.734/11, el cual establece que la puesta en marcha de este sistema conlleva la generación de varios registros informáticos.

2.2. Marco legal

En el país existe la Ley N° 22.939, que unifica el régimen de marcas y señales, certificados y guías, y la Ley N° 20.378, que otorga el régimen jurídico de la propiedad de los Sangre Pura de Carrera (SPC), ambas nacionales. Por otra parte, las provincias tienen sus propias leyes de marcas. En el caso particular de Buenos Aires, aplica el Código Rural (Decreto-Ley 10.081/1983), complementado con la Ley de Guía Única de Traslado N° 10.891/1990, vigentes hasta el 31 de diciembre de 2019 [6], las cuales establecen la obligatoriedad para todo propietario, de marcar el ganado mayor antes del año de edad. Para poder hacerlo, el propietario debe poseer un Boleto de Marca a su nombre expedido por la

Provincia, registrar el mismo ante la oficina de guías municipal y obtener allí mismo el permiso de marcación correspondiente. Se establece también en el Código Rural que, para tramitar un Boleto de Marca, el interesado debe ser ocupante legal de un inmueble rural (propietario, inquilino, con contrato de capitalización, etc.).

Una vez marcado el caballo, para realizar un movimiento se debe solicitar una Guía de Traslado. El proceso implica dirigirse en primer lugar a la oficina local de SENASA a efectos de obtener el DT-e (documento de tránsito electrónico). Para dar cumplimiento a este trámite, el propietario debe estar previamente registrado como productor en el Registro Nacional de Productores Agropecuarios (RENSPA), tener existencia de animales declarada, y acompañar los certificados sanitarios, expedidos por un Médico Veterinario habilitado. Posteriormente dirigirse a la oficina municipal para obtener la Guía de Traslado. Finalmente concurrir a la comisaría local y solicitar allí un certificado que debe acompañar el traslado, con los datos del propietario, del transportista y del vehículo.

Esta descripción del marco legal vigente hasta diciembre de 2019 inclusive, permite comprender cómo el mismo conspira contra las distintas actividades hípicas, contra el comercio y principalmente contra la tenencia regular de los equinos. En tal sentido se puede señalar:

a. Aquellos que cumplen con la obligación de marcar sus equinos, o los que están exceptuados de tal requisito (por ejemplo, los animales de raza pura como los SPC) y que viajan con frecuencia, por lo general los fines de semana, para participar en distintos eventos, sufren el inconveniente de efectuar un trámite sumamente burocrático: obtención de DT-e, Guía de Traslado y certificación policial.
b. Cuando el propietario de un equino sin marca quiere vender el mismo a algún concentrador para posterior envío a faena, no tiene manera de documentar correctamente dicha operación, transfiriendo ese problema al comprador, que no puede incorporar correctamente los animales de su stock, sacar DT-e para los mismos,

tampoco obtener permiso de reducción a marca propia o archivo de Guía, y mucho menos permiso de marcación (ya que se trata de animales mayores del año de edad y que no nacieron de madres de su propiedad). En la práctica esto se salva circunstancialmente por SENASA, con una Declaración Jurada que efectúa el comprador, lo cual carece de valor legal en cuanto a la identificación del equino y determinación de la propiedad, sin que tampoco solucione el problema ante las oficinas municipales a la hora de tramitar la Guía de Traslado, con la que se debe ingresar a la faena. Esto constituye uno de los principales problemas que es la falta de trazabilidad.

c. Aquellos equinos mestizos y sin marcas que quieren ser enviados a venta en remates/ferias, encuentran el mismo problema que lo expresado en el punto anterior.
d. Esta imposibilidad de cumplir la normativa, deriva también en un profundo descontrol sanitario.
e. Tampoco se establece claramente, en la legislación vigente hasta diciembre de 2019, el temperamento a adoptar en aquellos casos en que el equino se desplaza constituyéndose él mismo en un medio de transporte (por ejemplo: peones rurales que van del campo en el que viven al pueblo más cercano a efectuar compras, alumnos de escuelas rurales, cabalgatas recreativas que organizan distintos centros hípicos, etc.). En todos estos casos el propietario se mueve montando su caballo o sobre algún carruaje tirado por el animal, por lo tanto, pretender que lleve consigo la documentación requerida por el marco legal en vigencia constituye un despropósito.
f. Un alto porcentaje de los propietarios de caballos deportivos, de paseo, trabajo, desfiles, etc., no ocupan un inmueble rural. La mayoría de ellos tienen caballos como un elemento recreativo, en boxes o pequeños lotes dentro del radio urbano. También se da el caso de muchísimos habitantes del ámbito rural, que tienen caballos en el campo donde están empleados, y no son propietarios. Por lo tanto, todos ellos se ven

imposibilitados de obtener Boletos de Marca y Guías de Traslado. Sus caballos son orejanos (es decir que no están marcados), lo que motiva que a menudo sean sancionados, o incluso, en algunos casos, sufran el comiso de los animales.

Tales circunstancias derivaron en una gran desorganización en la identificación, contralor del traslado y control sanitario de los equinos. Es por esto que se sancionó la Ley Nº 13.627, vigente desde octubre de 2018 [6], la cual instaura un Registro de Identificación Equino y establece la implementación en todo el territorio de la Provincia de Buenos Aires, con carácter obligatorio, del Documento Único Equino y un microchip asociado como sistema de identificación y contralor del traslado de los equinos.

La identificación individual de los equinos consiste de un dispositivo electrónico subcutáneo (microchip), un documento identificatorio del animal y su propietario, vinculado de manera unívoca al microchip, y un registro en una base de datos informatizada.

El microchip es un identificador pasivo, con tecnología FDX–B (Full Duplex) de acuerdo a la definición en las normas ISO (International Organization for Standardization) Nº 11.784 y Nº 11.785; incorporado en material biocompatible no poroso; inyectable, capaz de comunicarse con lectores de mano a 12 cm de distancia; de configuración inviolable de único uso; frecuencia de operación 134.2 KHz; con el código del identificador electrónico programado en su proceso de fabricación como dispositivo de sólo lectura (“read-only”). Dicho código, es un código electrónico de 64 bits compuesto por los siguientes caracteres:

Tabla 1. Composición electrónica de un microchip.

Nº de bits	**Nº de cifras**	**Nº de combinaciones**	**Descripción**
1	1	2	Este bit indica si el identificador se usa o no para la identificación de los animales. En todas las aplicaciones relativas a animales ese bit será «1».
2-4	1	8	Vacío — todo ceros (espacio reservado para aplicaciones futuras).

5-9	2	32	Campo de información de usuario. Este bit será «0».
10-15	2	64	Vacío — todo ceros (espacio reservado para aplicaciones futuras).
16	1	2	Este bit indica la presencia o ausencia de un bloque de datos (para su uso en animales, ese bit será «0» = bloque sin datos).
17-26	4	1.024	Código del país. Es un código numérico de 3 dígitos que representa el nombre del país según norma ISO Nº 3.166. Para Argentina es el 032 (cero, tres, dos).
27-64	12	274.877.906.944	Código de identificación nacional de 12 dígitos, que después del código de país, identifica individualmente un animal a nivel nacional. Si el código de identificación nacional tiene menos de doce cifras, el espacio entre el código de identificación nacional y el código del país se rellenará con ceros.

Los microchips son provistos por empresas registradas en el MAIBA y pueden ser adquiridos por el propietario del equino o el veterinario habilitado que interviene en la identificación.

Para la lectura del microchip, se debe utilizar un lector de radiofrecuencia que cumpla con la norma ISO 11.785, es decir, al menos capaz de leer identificadores y de mostrar el código del país y el código nacional de identificación. Además, el lector debe poseer una tecnología que permita su conexión (inalámbrica o por cable) a un celular o tableta digital, al cual sea posible transferir el número de microchip leído.

El procedimiento de identificación debe ser llevado a cabo por un Médico Veterinario oficial o privado habilitado, lo que implica estar capacitado y debidamente registrado en la base de datos del MAIBA. El Médico Veterinario será quien efectúe la apertura del DUE, ante la solicitud del propietario del equino.

La presente Ley tiene entonces por prioridad brindar a los propietarios de equinos un nuevo estatus de identificación inviolable, registro, propiedad y control efectivo por las autoridades, utilizando para ello los nuevos elementos tecnológicos disponibles.

El correcto cumplimiento de la identificación del ganado equino, permitirá ofrecer un sistema confiable de trazabilidad individual y por rodeos, suficiente para asegurar tanto la procedencia como la propiedad de los animales. También posibilitará el desarrollo de los procesos de fiscalización eficientes, sin necesidad de trámites burocráticos, ni demoras en los transportes. Contribuirá al ordenamiento de los registros equinos, tendientes a determinar la cantidad de animales de esta especie que existe en el país y finalmente identificar claramente al equino y a su propietario, brindando así una herramienta de gran utilidad para la lucha contra el abigeato (hurto de ganado) y la determinación de la responsabilidad civil en casos de accidentes.

3

ALCANCE DE LA APLICACIÓN MÓVIL

3.1. Planificación de requerimientos

Como se ha mencionado anteriormente, la obtención de los requerimientos funcionales y no funcionales de la aplicación ha quedado a cargo del equipo profesional de analistas funcionales del proyecto.

A título informativo, se pueden mencionar los siguientes documentos sobre los cuales se basó el diseño, prototipado y desarrollo de la aplicación móvil:

- Minutas de reunión de las entrevistas realizadas al personal del MAIBA.
- Especificación de requisitos de la aplicación.
- Documentación anexa referente al DUE, como puede serlo, por ejemplo, su marco legal.
- Diagramas de flujo de funciones cruzadas, detallando la relación entre los procesos organizacionales y los responsables de los mismos.

3.2. Definición del alcance

Luego del análisis de la documentación obtenida y generada por el equipo de analistas funcionales, y considerando las necesidades del cliente y del usuario de la aplicación, se procedió a definir detalladamente su alcance.

La aplicación móvil está concebida como una herramienta de trabajo que deberá usar el veterinario habilitado, para poder registrar los datos del equino y eventualmente generar su respectivo DUE. La carga y obtención de estos datos se podrá realizar únicamente a través de dicha aplicación (no pudiendo realizarse desde la plataforma Web), esto quiere decir, que su uso será obligatorio al momento de generar un nuevo DUE.

Dado que en el entorno de trabajo del veterinario muchas veces no se cuenta con conexión a Internet, es primordial que la aplicación permita almacenar los datos del equino de manera offline, pudiendo posteriormente sincronizarlos, cuando se cuente con conexión a Internet.

A continuación, se detallan los módulos que conforman la aplicación móvil.

3.2.1. Ingreso al sistema

Para poder acceder al sistema, la persona deberá contar con un usuario y contraseña previamente gestionado. La validación de las credenciales se hará contra la plataforma Web, es por esto que se deberá contar con conexión a Internet para poder realizar el acceso. Una vez iniciada la sesión, se la deberá mantener activa hasta que el usuario decida finalizarla manualmente, independientemente de si se cierra la aplicación.

La administración de los usuarios se hará a través de la aplicación Web.

3.2.2. Listado de solicitudes

Una vez autenticado en la aplicación móvil, si el usuario corresponde a un veterinario habilitado, entonces deberá poder ver todas aquellas solicitudes que tiene asignadas y que ya se encuentren pagas. La generación de las solicitudes y el pago de las tasas correspondientes serán realizadas desde la aplicación Web.

Por cada solicitud se deberá mostrar el o los propietarios correspondientes, el estado en el que se encuentra y la cantidad de DUE a realizar.

El listado de solicitudes se deberá poder sincronizar y actualizar con la información que posee la aplicación Web, de forma manual por el usuario, siempre y cuando se cuente con conexión a Internet.

3.2.3. Administración de DUE

La aplicación deberá permitir la administración de DUE, esto es, generar nuevos DUE, editar su información, eliminarlos, ver su detalle y sincronizarlos con la aplicación Web. Un DUE corresponde a una solicitud en particular, por lo que, para poder administrarlo, se deberá antes indicar a qué solicitud pertenece. La cantidad máxima de DUE que se puedan generar será la especificada en la correspondiente solicitud.

Los DUE generados en la aplicación móvil estarán almacenados en la memoria local del dispositivo hasta tanto el usuario los sincronice, de forma manual, con la plataforma Web. Es por esto que la edición y eliminación de un DUE sólo surtirá efecto en la memoria del dispositivo. Una vez sincronizado un DUE, éste no podrá ser editado en la aplicación, y solo podrá ser eliminado localmente.

La aplicación deberá permitir en todo momento guardar un DUE como "borrador", posibilitando luego editar su información, hasta tanto no se haya sincronizado con la plataforma Web. Por cada DUE, la aplicación solicitará la siguiente información:

- **Estado legal (*)**: Campo seleccionable, entre "Mestizo" o "Puro".
- **Raza (*)**: Campo seleccionable, con los distintos tipos de raza de un equino.
- **Sexo (*)**: Campo seleccionable, entre "Macho", "Macho castrado" o "Hembra".
- **Edad aproximada (*)**: Campo numérico con la edad aproximada del equino en años.

- **Fecha exacta de nacimiento**: Campo de fecha, con la fecha exacta de nacimiento del equino. Al indicar este campo, el campo de edad aproximada será recalculado.
- **Número de DUE madre**: Campo de texto, con el número de DUE de la madre.
- **Número de DUE padre**: Campo de texto, con el número de DUE del padre.
- **RENSPA**: Campo de texto, con el número de RENSPA.
- **Pelaje (*)**: Campo de selección múltiple, con los distintos tipos de pelaje de un equino.
- **Detalle de la capa**: Campo de selección múltiple, con los distintos tipos de capa de un equino.
- **Particularidades de la cabeza**: Campo de selección múltiple, con los tipos de particularidad de la cabeza de un equino.
- **Particularidades del tronco**: Campo de selección múltiple, con los tipos de particularidad del tronco de un equino.
- **Particularidades de las manos y patas**: Campo de selección múltiple, por cada mano y cada pata, con los distintos tipos de particularidades posibles de las manos y patas de un equino.
- **Diagrama**: Diagramas de cada parte del cuerpo de un equino (abajo de la cabeza, frente de la cabeza, perfil derecho, perfil izquierdo, patas perfil derecho, patas perfil izquierdo y patas de frente y de atrás), sobre los cuales se podrá indicar, con un punto, la ubicación exacta o aproximada de la presencia de determinada característica, seleccionable desde un campo de selección simple.
- **Observaciones generales**: Campo de texto libre, donde se podrán indicar observaciones generales del equino.

- **Foto del lateral izquierdo y derecho (*)**: Imagen del lateral izquierdo y otra del lateral derecho del equino, capturada directamente desde la cámara u obtenida desde la galería del dispositivo. Estas fotos serán las utilizadas en el documento impreso del equino.
- **Fotos adicionales**: Cuatro imágenes opcionales, capturadas directamente desde la cámara u obtenidas desde la galería del dispositivo.
- **Número de microchip**: Número del microchip aplicado, el cual corresponderá con el número de DUE del equino. Dicho número no podrá ser indicado manualmente, es decir que la única manera de obtenerlo será a través del uso de una lectora de radiofrecuencia, vinculada de antemano con el dispositivo vía Bluetooth.
- **Número de cada microchip existente**: Número de cada microchip existente, que el equino tiene aplicado con anterioridad a la generación del DUE. Dichos números no podrán ser indicados manualmente, es decir que la única manera de obtenerlos será a través del uso de una lectora de radiofrecuencia, vinculada de antemano con el dispositivo vía Bluetooth.

Los campos que fueron marcados con (*) deben estar indicados para poder sincronizar el DUE.

Un DUE podrá ser guardado sin ningún dato especificado, como “borrador”. La aplicación validará que los datos mínimos se encuentren registrados sólo al momento de intentar sincronizarlo con la plataforma Web. En caso de no cumplirse con este requisito, la sincronización no será realizada.

3.2.4. Validación de microchip

La aplicación deberá contar con un módulo que permita verificar si un microchip es válido o no, esto es, si la empresa que lo fabricó se encuentra registrada en el MAIBA y, a su vez, si su número se encuentra dentro de una partida habilitada por el organismo.

3.2.5. Actualización de base de datos

La aplicación móvil deberá contar con una base de datos propia, que le permita al usuario trabajar de manera correcta, sin la necesidad de contar con conexión a Internet, es decir, de forma offline.

Dicha base de datos almacenará la siguiente información:

- Listado completo de pelajes de un equino.
- Listado completo de razas de un equino.
- Listado completo de capas de un equino.
- Listado completo de particularidades de la cabeza de un equino.
- Listado completo de particularidades del cuerpo de un equino.
- Listado completo de particularidades de los miembros de un equino.
- Listado de partidas de microchips habilitadas por el MAIBA, con información de la empresa que los fabricó.

La carga, modificación y eliminación de los datos que conforman cada uno de los listados antes mencionados se realizará desde la aplicación Web.

La aplicación móvil le deberá permitir al usuario sincronizar y actualizar la información de dicha base de datos, con la que se encuentra registrada en la plataforma Web. Para realizar esto, se deberá contar con conexión a Internet.

3.2.6. Permisos de la aplicación

Por lo antes mencionado, la aplicación deberá tener acceso a los siguientes elementos del dispositivo:

- **Hacer fotografías y videos**: Permite que la aplicación haga fotos o grabe videos con la cámara. Este permiso autoriza a la aplicación a utilizar la cámara en cualquier momento sin la confirmación del usuario.
- **Consultar, editar o borrar el contenido del almacenamiento USB**: Permite que la aplicación lea, modifique o elimine archivos en la memoria externa.
- **Acceso completo a red, recibir datos de Internet y ver conexiones de red**: Permite que la aplicación vea información sobre conexiones de red (por ejemplo, saber si se tiene o no conexión a Internet).
- **Acceder a los ajustes de Bluetooth y vincular con dispositivos Bluetooth**: Permite que la aplicación acceda a la configuración Bluetooth del teléfono, que establezca y que acepte conexiones con los dispositivos vinculados.

4

METODOLOGÍA Y HERRAMIENTAS DE DESARROLLO

A continuación, se mencionan la metodología y las herramientas utilizadas para el desarrollo del Sistema Informático de Gestión del Documento Único Equino.

4.1. Metodología de desarrollo

Si bien se utilizaron metodologías ágiles para el desarrollo de la aplicación Web, para la aplicación móvil se llevó a cabo un desarrollo más tradicional, siguiendo un orden específico de cada etapa del proyecto. Esto fue así ya que los requerimientos se encontraban previamente definidos por el equipo de analistas funcionales y había un único desarrollador, autor de esta tesina, para programar la aplicación móvil y realizar las pruebas funcionales básicas de la misma.

Se utilizó la herramienta Mantis Bug Tracker [12] para el seguimiento de tareas e incidencias, dado que era el software conocido por el equipo de trabajo.

4.2. Herramientas de desarrollo - Aplicación Web

En lo que respecta a la tecnología utilizada para el desarrollo de la aplicación Web, se utilizó PHP como lenguaje de programación, haciendo uso del framework Symfony v2.8.* LTS

(Long Term Support), y MariaDB como sistema de gestión de bases de datos. Dicha aplicación corre sobre un sistema operativo Ubuntu Server y un servidor Web Apache.

Por otra parte, se utilizó como herramienta de versionado Apache Subversion, la cual venía utilizando el equipo de desarrollo para el control de versiones de los demás subsistemas de la Plataforma Web del MAIBA previamente desarrollados.

Se aplicó un diseño Web responsive o adaptativo, con el objetivo de obtener una correcta visualización en distintos tipos de dispositivos, mejorando así la experiencia del usuario.

4.3. Servidores y ambientes de trabajo

La configuración y puesta a punto tanto de los servidores como de los ambientes de trabajo quedó a cargo del equipo de infraestructura.

Dado que la aplicación móvil no se implementó de forma aislada, sino que forma parte del Sistema Informático de Gestión del Documento Único Equino, es importante mencionar que, en base al análisis de la situación del centro de cómputos del MAIBA, la Dirección de Informática del organismo decidió instalar la solución informática en servidores externos que posee la Dirección Provincial de Informática de la Provincia de Buenos Aires. Para este caso en particular, se gestionó el alojamiento en el Cloud de Provincia NET, el cual posee clasificación TIER II y monitoreo 7x24. En este contexto, se asignaron dos servidores virtuales idénticos, de uso exclusivo, con los siguientes recursos:

- vCPUs 6.
- Memoria 24 GB.
- Disco 1.500 GB.

Uno tendrá como finalidad ser servidor de desarrollo y el otro de producción. Ambos servidores son idénticos, instalándose en ellos:

- Sistema operativo Ubuntu Server 16.04.2 LTS.
- Symfony 2.8.* LTS.
- PHP 7.1.4.
- Servidor Web Apache 2.4.25 LTS.
- SVN 1.9.3 Stable.
- María DB 10.1.22 Stable.

4.4. Aplicación móvil - Marco teórico

4.4.1. Introducción

A nivel de programación, existen varias formas de desarrollar una aplicación móvil. Cada una de ellas tiene diferentes características y limitaciones, especialmente desde el punto de vista técnico. Se pueden clasificar cinco tipos principales de aplicaciones móviles en base al entorno de desarrollo utilizado y a sus plataformas destino: aplicaciones nativas, aplicaciones Web móviles, aplicaciones híbridas, aplicaciones interpretadas y aplicaciones generadas por compilación cruzada [14].

4.4.2. Aplicaciones nativas

Una aplicación nativa es una aplicación que ha sido desarrollada para ejecutarse en un sistema operativo. Este tipo de aplicaciones se desarrollan en diferentes lenguajes de programación en función del sistema operativo en el que van a ser utilizadas. Por ejemplo, aquellas aplicaciones que son desarrolladas para iOS, son diseñadas con el lenguaje Objective-C o Swift, y las que son desarrolladas para el sistema operativo de Android utilizan el lenguaje Java o Kotlin.

Algunas de las ventajas de las aplicaciones nativas son:

- La aplicación está instalada en el dispositivo móvil y se dispone de un acceso directo.
- Es posible utilizar los canales de compra de aplicaciones móviles o plataformas de distribución digital, como el App Store de iOS o el Google Play de Android, y así encontrar las aplicaciones más fácilmente que buscando en la Web.
- Debido a que son programadas en un lenguaje específico, las utilidades del dispositivo son accesibles, como la cámara o el dispositivo GPS.
- Pueden ser ejecutadas sin una conexión a Internet, aunque en ocasiones algunas partes de la aplicación pueden requerir conexión.
- Permiten notificaciones push, las cuales consisten en el envío de mensajes al usuario acerca de alguna novedad sobre la aplicación.

Algunas de las desventajas de las aplicaciones nativas son:

- Se debe desarrollar una aplicación para cada plataforma, sin la posibilidad de reutilizar código.
- El usuario debe actualizar la aplicación manualmente desde el canal de compra de su dispositivo.
- La complejidad de los lenguajes de programación conlleva un mayor tiempo y coste a la hora de desarrollar dichas aplicaciones.
- El desarrollador se enfrentará a procesos de validación, en ocasiones complejos, a la hora de publicar su aplicación en las distintas plataformas de distribución digital.

4.4.3. Aplicaciones Web móviles

Se denominan aplicaciones Web móviles o WebApp, a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor Web a través de Internet o de una intranet

mediante un navegador. En otras palabras, es una aplicación de software que se codifica en un lenguaje soportado por los navegadores Web (generalmente HTML, conjuntamente con JavaScript y CSS), permitiendo su ejecución en éstos.

Algunas de las ventajas de las aplicaciones Web móviles son:

- Son traducidas y visualizadas por el propio navegador Web utilizado en el dispositivo, por lo que no necesitan instalarse y pueden ser utilizadas en diferentes plataformas sin mayores inconvenientes y sin necesidad de desarrollar un código diferente para cada caso particular.
- Fácil diseño, no es necesario pensar en el diseño de una aplicación móvil, simplemente diseñarla para dispositivos con una pantalla más pequeña.
- El proceso de desarrollo es más sencillo, puesto que el código es reutilizable de una plataforma a otra.
- Son multidispositivo y multiplataforma, es decir, funcionan en cualquier dispositivo o sistema operativo, siempre que se disponga de conexión a Internet.
- No ocupa espacio en disco.
- Actualizaciones inmediatas, al conectarse se está usando siempre la última versión.
- Los virus no dañan los datos ya que éstos están guardados en el servidor de la aplicación, de modo que tampoco se dañan otras aplicaciones.
- Pueden ser publicadas sin la aprobación de ningún fabricante.

Algunas de las desventajas de las aplicaciones Web son:

- Es necesaria una conexión a Internet para su ejecución, de modo que si no se dispone o se interrumpe dicha conexión no se pueden utilizar.
- No es posible publicarlas en las plataformas de distribución digital.

- Dadas las restricciones de seguridad impuestas a la ejecución de código por medio de un navegador, el acceso a los recursos del móvil es muy limitado.
- El usuario debe recordar la dirección e introducirla en el navegador, haciendo más difícil acceder a ellas.
- El tiempo de respuesta decae debido a la interacción cliente-servidor.
- Estas aplicaciones son más lentas, ya que ejecutar los HTML e interpretar los JavaScripts es más costoso.

4.4.4. Aplicaciones híbridas

Las aplicaciones móviles híbridas se desarrollan de manera similar a las páginas Web, con una combinación de HTML5, CSS y JavaScript. Sin embargo, en lugar de acceder a ellas a través de un navegador móvil, estas aplicaciones se utilizan a través de un WebView, es decir, un navegador integrado dentro de una aplicación nativa. Esto permite reescribir con un mismo código, diferentes aplicaciones para cada plataforma, y distribuirlas en cada una de sus tiendas. Entre los frameworks más populares se encuentran PhoneGap, CocoonJS, Ionic y Sencha Touch.

Algunas de las ventajas de las aplicaciones híbridas son:

- Son multiplataforma, permiten que el código fuente se pueda ejecutar en diferentes plataformas.
- Estas aplicaciones permiten acceder, a través de diversas API, a todos los recursos del móvil, como pueden ser la cámara o el sistema GPS.
- No es necesario tener conexión a Internet para ejecutar la aplicación, a excepción de partes concretas de la aplicación que requieran dicha conexión.
- No es necesario utilizar un lenguaje específico para crear la aplicación íntegra, solo es necesario para la plataforma contenedora.

- Es posible subir estas aplicaciones a las plataformas de distribución digital, lo cual permite al usuario encontrarlas de una forma más rápida y por ello facilita su distribución.
- El modelo híbrido permite que las aplicaciones sean instaladas en el dispositivo y que el usuario disponga de un acceso directo, facilitando el acceso a ellas.

Algunas de las desventajas de las aplicaciones híbridas son:

- La experiencia de usuario se ve perjudicada al no utilizar componentes nativos en la interfaz.
- Las actualizaciones son manuales, el usuario debe acceder al canal de compra de aplicaciones de las distintas plataformas para conseguir las actualizaciones de la aplicación.
- Al igual que en el caso de las aplicaciones nativas, para poder distribuirlas en las plataformas de distribución digital, es necesario pasar varios procesos de validación que en ocasiones pueden resultar complicados.
- La parte nativa que envuelve el código fuente basado en HTML, debe ser programada con un lenguaje específico para las distintas plataformas.

4.4.5. Aplicaciones interpretadas

Las aplicaciones interpretadas son desarrolladas utilizando un lenguaje base, el cual se traduce en su mayor parte a código nativo, mientras el resto es interpretado en tiempo de ejecución. Estas aplicaciones son implementadas de manera independiente de las plataformas, utilizando diversas tecnologías y lenguajes, tales como JavaScript, Java, Ruby y XML, entre otros. Entre los frameworks más conocidos, se pueden mencionar React Native, NativeScript y Titanium.

Algunas de las ventajas de estas aplicaciones son:

- El código se puede reutilizar entre diferentes implementaciones de la aplicación en distintos sistemas operativos.
- La interfaz de usuario es representada mediante controles "nativos", por lo que el rendimiento de la interfaz de usuario puede ser tan rápido como una aplicación nativa.

Algunas de las desventajas de las aplicaciones interpretadas son:

- No se puede reutilizar todo el código, por lo que es posible que se deba escribir código nativo.
- La velocidad puede verse afectada, ya que es posible que algunos de los idiomas intermedios deban interpretarse en tiempo de ejecución.
- El acceso a las funciones del dispositivo y del sistema operativo depende del framework o plugins utilizados.
- La personalización de la interfaz de usuario depende del framework utilizado.

4.4.6. Aplicaciones generadas por compilación cruzada

Estas aplicaciones son compiladas de forma nativa, creando una versión específica para cada plataforma destino. Algunos ejemplos de entornos de desarrollo para generar aplicaciones por compilación cruzada son Applause, Xamarin, Embarcadero Delphi 10 Seattle y RubyMotion.

Algunas de las ventajas de las aplicaciones generadas por compilación cruzada son:

- Alcanzan un alto rendimiento general debido al código nativo generado.
- Es posible realizar desarrollos para múltiples plataformas empleando un mismo lenguaje de programación, compartiendo gran porcentaje del código. Esto implica un ahorro considerable en tiempo y recursos.
- Permite hacer uso de las capacidades específicas de la plataforma.

Algunas de las desventajas de estas aplicaciones son:

- La mayoría de los entornos de desarrollo para generar aplicaciones por compilación cruzada generalmente implican altos costos, tanto en licencias como en requerimientos de hardware y software para las distintas plataformas.
- En general, los entornos de desarrollo cuentan con curvas de aprendizaje bastante acentuadas.
- Las aplicaciones generadas, si bien utilizan controles nativos, su interfaz general no siempre es óptima, ya que el diseño se realiza con las reglas de la herramienta, que luego deben ser interpretadas y convertidas a nativo.

4.5. Aplicación móvil - Framework utilizado

4.5.1. Criterios de elección de la herramienta a utilizar

Al momento de elegir la tecnología con la cual desarrollar una aplicación, entran en juego múltiples factores: tiempos del proyecto, costo, capacidad técnica del equipo y las ventajas y desventajas intrínsecas de cada tecnología.

En el caso particular de la aplicación móvil para la gestión del DUE, se debió considerar la solicitud realizada por el MAIBA de generar un producto final para el sistema operativo Android, que pudiese ser descargado por los usuarios desde el Google Play Store. De forma ideal, pero no prioritaria ni obligatoria, se solicitó también realizar la misma aplicación para el sistema operativo iOS, siempre y cuando no se presenten limitaciones o complicaciones para hacerlo.

Por otro lado, se contempló la necesidad del usuario de poder utilizar la aplicación de forma offline y acceder desde la aplicación a elementos del dispositivo, como son la cámara y el Bluetooth. En cuanto al equipo y al tiempo para su desarrollo, se disponía de cuatro meses

para hacerlo y un único desarrollador, autor de esta tesina, quien poseía poca experiencia en el desarrollo de aplicaciones móviles, pero sí bastante experiencia en el desarrollo de aplicaciones Web.

Por lo antes mencionado, y considerando el alcance relativamente acotado de la aplicación, se decidió utilizar una herramienta multiplataforma de software libre, que utilice JavaScript o similar como principal lenguaje de programación, con el objetivo de desarrollar el código una única vez y compilarlo de forma nativa tanto para Android como para iOS.

Entre las herramientas disponibles se optó por utilizar React Native como framework de desarrollo multiplataforma, aplicando las ideas principales de la arquitectura Flux para gestionar el estado y el flujo de datos de la aplicación, haciendo uso para esto de la librería MobX. Dichas herramientas se detallan a continuación.

4.5.2. React Native

React Native es una herramienta de desarrollo de aplicaciones multiplataforma basada en JavaScript, creada por Facebook en 2013 como parte de un framework de desarrollo interno, y liberada en 2015 como código abierto.

Hace uso de los componentes declarativos de React para construir las interfaces de usuario, permitiendo que el código JavaScript se ejecute e interactúe con los sistemas iOS y Android de la misma manera que lo haría un código nativo. En términos inteligibles, esta tecnología ofrece un *bridge* o "traductor", que ejecuta el código de React Native en Objective-C para iOS y Java para Android. Por lo tanto, la experiencia de usuario es prácticamente idéntica a una aplicación nativa. Por ejemplo, en el caso en el que la vista expresa que hay que renderizar un botón con el texto "Hola mundo", cuando la aplicación carga dicha pantalla, React Native creará un botón nativo con dicho texto y lo colocará en su posición correspondiente. Es decir, todas las interacciones con el usuario serán nativas.

Facebook mantiene oficialmente el soporte para las plataformas Android y iOS, pero la comunidad ha creado herramientas para ejecutar aplicaciones React Native en otras plataformas, como Windows, macOS y distintos navegadores. React Native es de código abierto y al momento utiliza la licencia MIT.

Entre las ventajas de utilizar React Native, se pueden mencionar:

- **Es multiplataforma**: Utilizar React Native permite usar el mismo código tanto para la implementación en iOS como en Android. La mayor parte del código puede ser reutilizado entre Android y iOS, permitiendo un gran ahorro en recursos.
- **Único equipo de desarrollo**: Al poder reutilizar el código entre iOS y Android, sólo es necesario tener un equipo para desarrollar y mantener una aplicación, mientras que una desarrollada con la tecnología nativa de cada plataforma requeriría tener dos equipos independientes, duplicando trabajo y especificaciones.
- **Recarga en vivo**: Otra de las características de React Native, es que permite visualizar la aplicación mientras se desarrolla. No es necesario compilar el código con los nuevos cambios que se quieran aplicar, ya que simplemente actualizando el simulador con el que se esté trabajando, se pueden ver las novedades. Esto permite agilizar las tareas de prueba sobre la app. Se puede ejecutar nuevo código sin necesidad de perder el estado en el que se encuentra la aplicación.
- **Uso de código nativo**: Una de las opciones que aporta más potencia al desarrollo con React Native es la posibilidad de combinar el trabajo de componentes desarrollados en JavaScript junto con los desarrollados en Objective-C, Swift, Java o Kotlin. Se puede incluir diferentes librerías nativas

o trabajar directamente con código nativo sin que el rendimiento o el desarrollo de los componentes JavaScript se vean afectados.

- **Curva de aprendizaje sencilla:** React Native es fácil de leer y sencillo de aprender ya que se basa en los conceptos fundamentales del lenguaje JavaScript, siendo especialmente intuitivo tanto para los expertos en dicho lenguaje como para las personas sin experiencia en él.
- **Respaldada por Facebook**: React Native es una tecnología creada y mantenida por Facebook, que además la utiliza en sus proyectos, lo cual es una garantía de que el soporte va a continuar en el largo plazo.
- **Posee una comunidad activa**: React Native es una tecnología de código abierto y su comunidad es una de las de mayor crecimiento en el mundo del software libre.

4.5.3. React

React (también llamada React.js o ReactJS) es una librería JavaScript de código abierto, desarrollada y mantenida por Facebook, enfocada en crear interfaces de usuario interactivas de forma sencilla.

Está fuertemente basado en componentes, los cuales constituyen la interfaz de usuario (un botón, un buscador, etc.). Estos componentes reaccionan tanto a los eventos producidos por el usuario como a los del servidor. Finalmente, se repintan a sí mismos cuando ocurre un cambio de estado. La definición de dichos componentes se realiza usando una sintaxis especial llamada JSX, que permite escribir etiquetas HTML dentro de JavaScript para mejorar la expresividad del código. Usar JSX no es obligatorio, pero sí recomendable. Como se verá más adelante, se ha utilizado dicha sintaxis para el desarrollo de la aplicación móvil, debido a la

comodidad que conlleva. React utiliza una herramienta llamada Babel para realizar posteriormente la transpilación a JavaScript.

En lugar de generar directamente elementos HTML y modificar el DOM, que es una operación muy lenta, los componentes de React generan (renderizan) un modelo del DOM en memoria. Una vez generado el Virtual DOM completo, React se encarga de buscar las diferencias entre el DOM real y el virtual y realizar únicamente las modificaciones necesarias sobre el DOM real. Esto se traduce en una mejora del consumo de memoria y en el rendimiento con respecto a otros frameworks.

React mantiene una distinción muy clara entre dos tipos de información, las propiedades (*props*) del componente y el estado (*state*) del componente.

Las **propiedades del componente** están formadas por información que es inmutable para el componente. Esta información se recibe a través del objeto *props* en el momento de construir el componente y el componente puede asumir que no será modificada en ningún momento durante su ciclo de vida.

Se trata de información que el componente no "posee", por lo que será otro componente el responsable de modificarla. Cuando esa modificación se produzca, el dueño de la información reconstruirá nuevamente todo su Virtual DOM asociado, incluyendo sus componentes hijos, que serán creados desde cero con la nueva información. Es por ello que se puede asumir que, durante el ciclo de vida de un componente, es decir, desde que se crea hasta que se destruye, esta información es inmutable.

El **estado del componente** se almacena en la propiedad *state* y está formado por aquella información que sí es gestionada y modificada directamente por el componente. Cada vez que se realiza un cambio en el estado del componente como respuesta a una acción del usuario (o a algún otro factor externo), React se encarga de renderizar nuevamente el componente por completo. Puesto que un componente puede contener otros componentes, en el proceso de

renderizado se volverán a crear estos componentes hijos con la nueva información. Es posible (y habitual) que el estado mutable de un componente sean propiedades inmutables de sus componentes hijos.

En el siguiente esquema se puede observar cómo se genera la jerarquía de componentes y cómo se propaga la información de unos a otros:

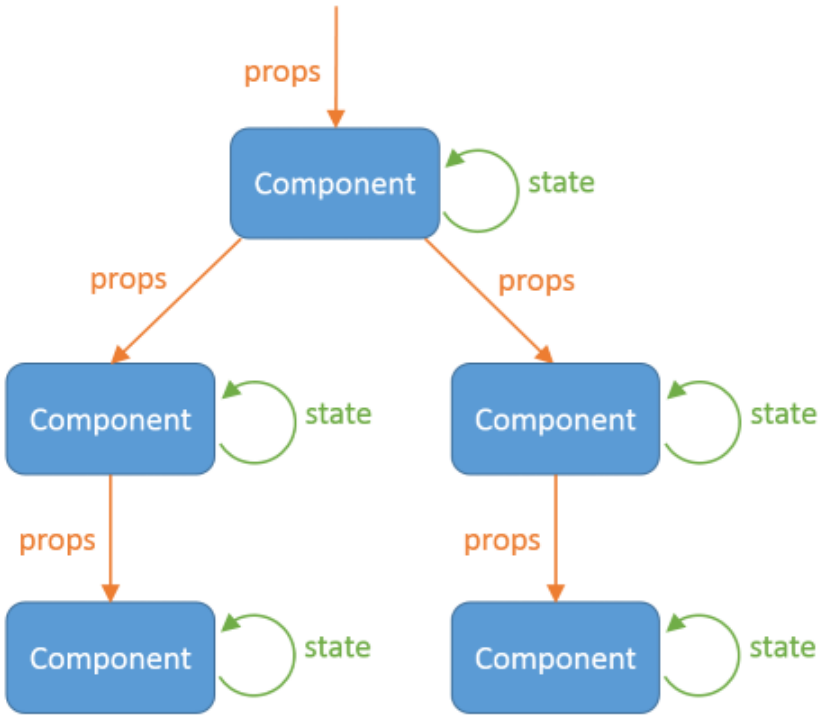


Figura 4.1. Esquema de flujo de datos en React [34].

La información siempre fluye de componentes padres a componentes hijos. Un componente hijo no puede modificar directamente el estado de un componente padre. Esto hace que sea más fácil razonar sobre el comportamiento general de la aplicación y las implicaciones de cada cambio realizado sobre el modelo y sobre el DOM.

Existen muchas situaciones en las que se tiene que notificar cambios en un componente hacia su padre. Para esos casos la solución es incluir dentro de las propiedades inmutables que recibe el hijo al ser creado, una función callback del padre que será invocada en el momento oportuno. De esta forma se trata todo de forma homogénea y la función callback no es más que otro elemento más de las propiedades inmutables que recibe el componente al ser creado.

Como React se encarga sólo de la Vista de la aplicación dentro de un paradigma MVC o MVVM, desde Facebook recomiendan el uso de Flux, el cual se detalla a continuación.

4.5.4. Flux

Flux es una arquitectura ideada por Facebook, que facilita la gestión y flujo de datos de una aplicación Web. Propone que el camino de los datos sea unidireccional y que exista una única fuente de verdad. De este modo, por medio de acciones, todo el flujo acaba llegando a un sitio que almacena el estado y que se encarga de actualizar las vistas que están suscritas a los cambios que en éste tienen lugar.

Se puede decir que Flux es el encargado, en cierto modo, de desacoplar el estado global de la aplicación de la parte visual formada por los componentes.

De forma esquematizada el flujo de los datos es el siguiente:

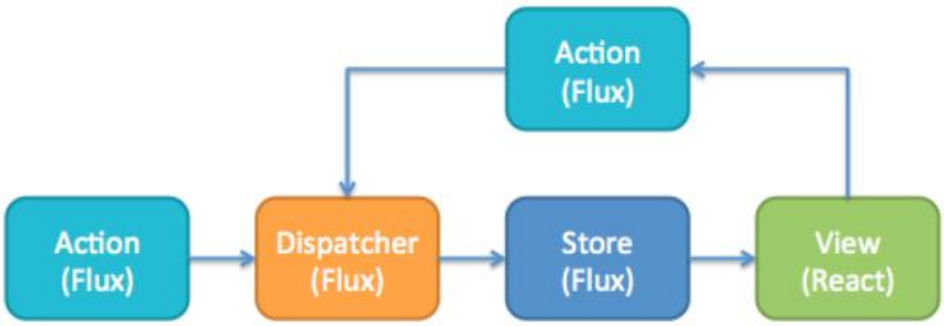


Figura 4.2. Esquema de flujo de datos en Flux [35].

La **vista** serían los componentes React.

El **store** sería lo más parecido al modelo de la aplicación ya que almacena los datos/estado de la misma. En Flux pueden haber varios store. No hay métodos en el store que permitan modificar los datos en él, eso se hace a través de dispatchers y acciones.

Una **acción** es simplemente un objeto JavaScript que indica una intención de realizar algo y que lleva datos asociados si es necesario.

Las acciones son enviadas a un **dispatcher** que se encarga de dispararla o propagarla hasta el store. La vista es la que se encarga de enviar las acciones al dispatcher. Un dispatcher no es más que un mediador entre el store o stores y las acciones. Sirve para desacoplar el store de la vista, ya que así no es necesario conocer qué store maneja una acción concreta.

En resumen, el patrón Flux sigue el siguiente recorrido:

- La vista, mediante un evento envía una acción con la intención de realizar un cambio en el estado.
- La acción contiene el tipo y los datos (si los hubiere) y es enviada al dispatcher.
- El dispatcher propaga la acción al store y se procesa en orden de llegada.
- El store recibe la acción y dependiendo del tipo recibido, actualiza el estado y notifica a las vistas de ese cambio.
- La vista recibe la notificación y se actualiza con los cambios.

Todo en un único sentido, es decir, el flujo de datos sigue una única dirección, evitando los problemas de los patrones con comunicación bidireccional como MVC o MVVM, donde el flujo no está bien definido y un cambio en la interfaz puede disparar múltiples eventos que afecten a múltiples vistas, generando efectos en cascada, lo que hace muy difícil la comprensión y depuración de aplicaciones complejas.

4.5.5. MobX

MobX es una librería independiente destinada a facilitar la gestión del estado y flujo de datos de una aplicación. En la siguiente imagen se pueden observar sus conceptos principales:

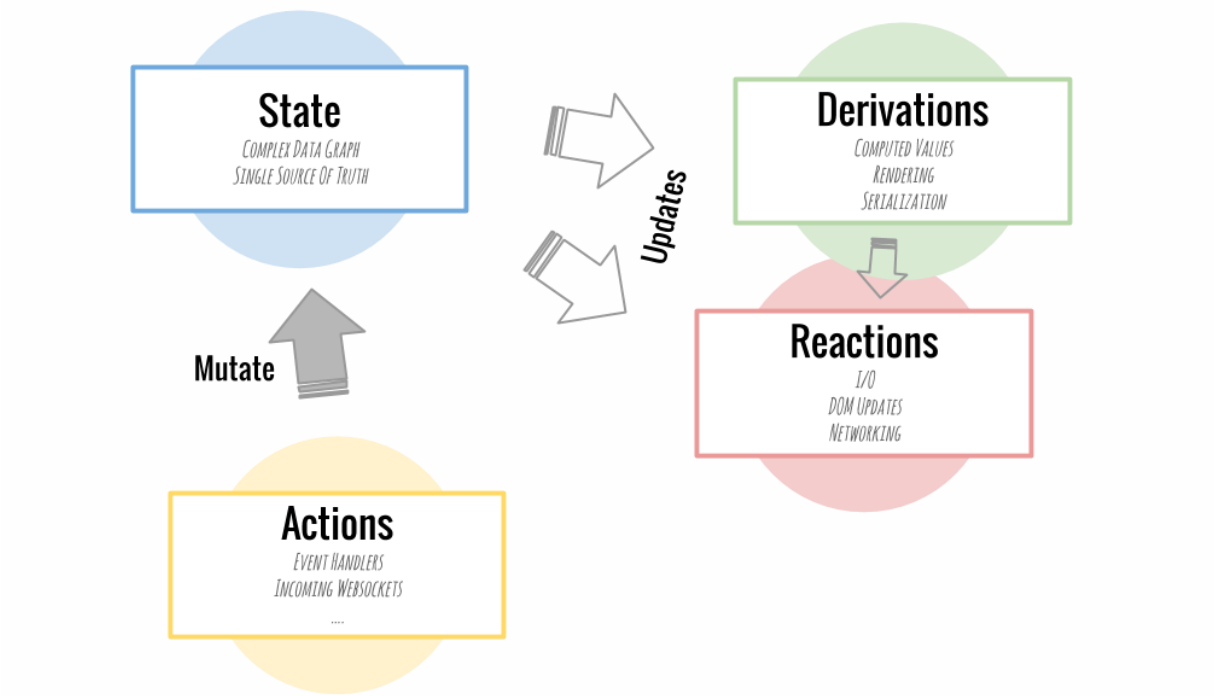


Figura 4.3. Conceptos principales de MobX [24].

1. En primer lugar, se encuentra el *estado observable* de la aplicación, integrado por clases, objetos, matrices, valores primitivos y/o referencias, conformando el modelo de la misma.
2. En segundo lugar, están las *derivaciones*. Básicamente, son cualquier valor que pueda calcularse automáticamente a partir del estado de la aplicación. Estas derivaciones, o valores calculados, pueden abarcar desde valores simples, hasta elementos complejos como por ejemplo una representación visual en HTML.
3. Las *reacciones* son muy similares a las derivaciones. La principal diferencia es que estas funciones no producen un valor. En su lugar, se ejecutan automáticamente para realizar alguna tarea, por lo general relacionada a E/S. Se aseguran de que el DOM esté actualizado o que las solicitudes de red se realicen automáticamente en el momento adecuado.
4. Finalmente se encuentran las *acciones*. Las acciones son el medio principal para modificar el estado de la aplicación. MobX se asegurará de que todos los cambios en

> dicho estado, causados por las acciones del usuario o por conexiones entrantes, sean procesados automáticamente por todas las derivaciones y reacciones, sincrónicamente y sin fallos.

En analogía con una hoja de cálculo, todas las celdas de datos que tienen valores forman el *estado observable*. Las fórmulas y los gráficos serían *valores calculados*, que pueden derivarse de las celdas de datos y otras fórmulas. Dibujar la salida de una celda de datos o una fórmula en la pantalla sería una *reacción*. Cambiar una celda o fórmula de datos sería una *acción*.

Más adelante se detallará la aplicación práctica de MobX en el desarrollo de la aplicación móvil.

4.5.6. Consideraciones

Si bien el MAIBA solicitó la utilización de ciertas herramientas para el desarrollo de la aplicación Web, para la aplicación móvil no hubo restricción alguna. Por esta razón, la selección en este caso fue responsabilidad del equipo de desarrollo. Todo el software utilizado es open source, por lo que no requiere el pago de licencias para su uso.

5

PROTOTIPOS

5.1. Introducción

Antes de comenzar con el desarrollo de la aplicación móvil, fue indispensable realizar diferentes prototipos de la misma, dando una idea general de cómo quedaría el producto final. Dichos prototipos resultan útiles para definir las funcionalidades concretas que tendrá la aplicación, para visualizar el aspecto de distintos diseños posibles e incluso para simular tests de usabilidad y experiencia de usuario.

5.2. Marco teórico

5.2.1. Experiencia de usuario

La experiencia de usuario (UX, por sus siglas en inglés) tiene su origen en el campo del marketing, estando muy vinculado con el concepto de experiencia de marca (pretensión de establecer una relación familiar y consistente entre consumidor y marca). En el marketing, un enfoque centrado en la experiencia de usuario trae consigo no solamente analizar los factores que influyen en la elección o adquisición de un determinado producto o servicio, sino también analizar cómo los consumidores hacen uso del producto o servicio y a su vez la experiencia resultante de utilizarlo.

Arhippainen y Tähti (2003) definen la experiencia de usuario como la experiencia que obtiene el usuario cuando interactúa con un producto en condiciones particulares [7].

Knapp Bjerén (2003) es más específico al definirla como "el conjunto de ideas, sensaciones y valoraciones del usuario, resultado de la interacción con un producto; es resultado de los objetivos del usuario, las variables culturales y el diseño de la interfaz" (p.4), especificando no sólo de qué fenómeno es resultante, sino también qué elementos la componen y qué factores intervienen en la interacción [8]. En el contexto de la Web, DNX (2005) define la buena experiencia del usuario como un objetivo: "lo que se persigue es generar sensaciones y valoraciones de los usuarios hacia nuestro sitio Web lo más agradables, positivas y satisfactorias posibles", además de reseñar la "fidelidad del usuario" como consecuencia de alcanzar este objetivo [9].

En conclusión, se puede definir la experiencia de usuario como la sensación, sentimiento, respuesta emocional, valoración y satisfacción del usuario respecto a un producto o servicio, resultado del fenómeno de interacción con éste y la interacción con su proveedor.

5.2.2. Factores de la experiencia de usuario

En la siguiente ilustración se puede observar qué aspectos influyen en la interacción y experiencia del usuario; diferenciando factores propios del usuario, factores sociales, culturales, del contexto de uso y propios del producto.

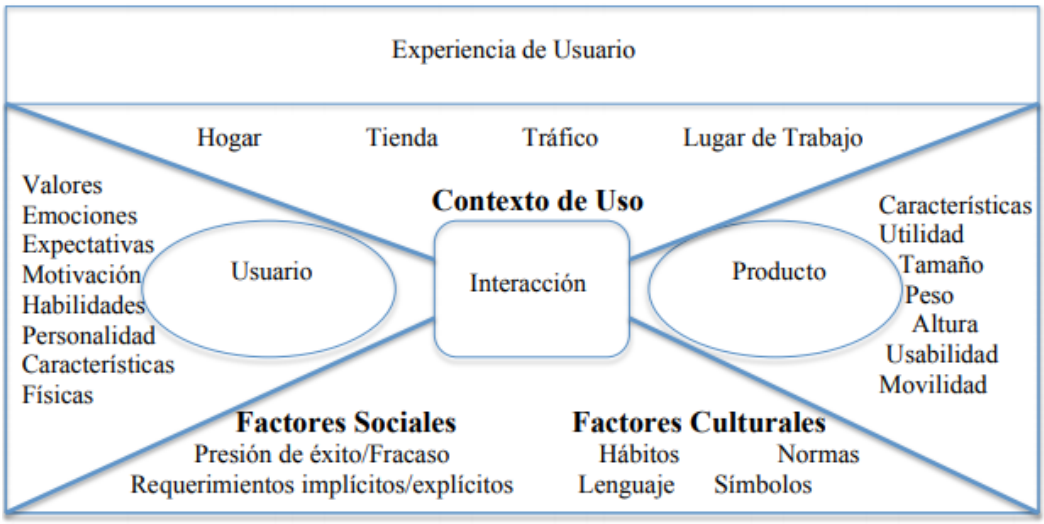


Figura 5.1. Factores que componen la experiencia de usuario.

En el contexto de la Web, Morville (2004) propone el análisis de la UX en base a siete facetas o propiedades que debe cumplir un sitio: útil, usable, deseable, encontrable, accesible, creíble y valioso [10]. En este mismo contexto, Mahlke (2002) identifica cuatro propiedades del producto Web: utilidad, facilidad de uso, cualidad de hedónico y atractivo visual; como factores percibidos por el usuario y que influyen con distinto peso en la intención de uso del producto [11].

5.2.3. Herramientas de experiencia de usuario

Entre las principales herramientas dentro de la experiencia de usuario, se pueden mencionar:

5.2.3.1. Prototipado rápido

Este método está asociado a la idea de desarrollar diferentes conceptos propuestos mediante prototipos de software o hardware, para su posterior evaluación. El desarrollo de la simulación o prototipado de la aplicación futura puede ser de gran ayuda, permitiendo a los usuarios visualizar el sistema (su concepto) e informar sobre el mismo, pudiéndose utilizar para aclarar opciones sobre los requerimientos de usuario y para especificar e incluir detalles a la interfaz de usuario.

Ventajas:

- Brinda una comunicación eficaz y rápida de las ideas que se quieren implementar en el diseño de la aplicación.
- Mayor flexibilidad de diseño.
- Disminuye los errores que se puedan dar en el desarrollo.
- Da una visión clara de lo que será el producto final.

5.2.3.2. Pruebas de usabilidad

Las pruebas de usabilidad tienen su principal enfoque en medir la capacidad de la aplicación, en cumplir con el objetivo para el cual fue diseñado y desarrollado, además de medir qué tan fácil o sencillo es para el usuario hacer uso de la interfaz gráfica de la misma. Al tratarse de una aplicación móvil, se debe medir si la aplicación desarrollada cumple con los requerimientos, satisfacción de usuario, facilidad de uso, etc.

Las pruebas de usabilidad permiten contrastar todas las ideas del equipo que diseñó y desarrolló la aplicación con la realidad del usuario que lo tiene que utilizar. Es una manera de corroborar que las decisiones que se toman tienen de verdad un impacto positivo sobre el producto o servicio mediante una demostración válida con hechos tangibles.

5.2.3.3. Evaluación heurística

Una evaluación heurística es un método de inspección de la usabilidad sin usuarios. Éste consiste en examinar la calidad de uso de una interfaz por parte de varios evaluadores expertos, a partir del cumplimiento de unos principios reconocidos de usabilidad: los heurísticos.

El principal objetivo de la evaluación heurística es medir la calidad de la interfaz de cualquier aplicativo en relación a su facilidad para ser aprendido y usado por primera vez. El evaluador se pone en la piel del usuario real del sistema, intentando predecir los errores que podrá encontrarse.

Si bien existen una variedad de listados de principios heurísticos, uno de los más reconocidos es el presentado por Jakob Nielsen, en el cual define diez principios generales, a los que denomina principios heurísticos de usabilidad [13]:

1. **Visibilidad del estado del sistema:** El sistema siempre debe mantener informados a los usuarios sobre lo que está sucediendo, a través de comentarios apropiados dentro de un tiempo razonable.
2. **Relación entre el sistema y el mundo real:** El sistema debería hablar el lenguaje de los usuarios mediante palabras, frases y conceptos que sean familiares al usuario, más que con términos relacionados con el sistema. Seguir las convenciones del mundo real, haciendo que la información aparezca en un orden natural y lógico.
3. **Libertad y control por el usuario:** En muchas ocasiones los usuarios eligen ciertas opciones y funciones por error dentro el sistema, por lo que lo ideal en estos casos es darle la libertad y opción al usuario de poder salir en cualquier momento que desee de cierta interfaz, sin tener que realizar mucho esfuerzo.
4. **Consistencia y estándares:** Los usuarios no deberían cuestionarse si acciones, situaciones o palabras diferentes significan en realidad la misma cosa, simplemente seguir la interfaz del sistema de manera sencilla.
5. **Prevención de errores:** Si bien los mensajes de error en una aplicación son muy importantes y relevantes, lo mejor sería tener un cuidado dentro del diseño para prevenir dichos inconvenientes con el usuario.
6. **Reconocer antes que recordar:** Se deben hacer visibles los objetos, acciones y opciones. El usuario no tendría que recordar la información que se le da en una parte del proceso, para poder seguir adelante. Las instrucciones para el uso del sistema deben estar a la vista o ser fácilmente recuperables cuando sea necesario.
7. **Flexibilidad y eficiencia de uso:** Se debe tener una aplicación preparada para todo tipo de usuario, desde los más novatos hasta los más experimentados. Si se consigue que cualquiera pueda utilizar la aplicación se logra flexibilidad, y si se tienen opciones para los más experimentados se obtiene eficiencia.

8. **Estética y diseño minimalista:** La información presentada al usuario no debe ser irrelevante o innecesaria. Cada información adicional dentro de un diálogo compite con la información mostrada al usuario y disminuye su visibilidad relativa.
9. **Ayudar a los usuarios a reconocer, diagnosticar y solucionar los errores:** Los mensajes de error deben ser expresados en un lenguaje sencillo y comprensible para el usuario, indicando de forma precisa el problema y adicionalmente sugerir una solución apropiada.
10. **Ayuda y documentación:** Es mucho mejor si el sistema puede ser utilizado sin documentación, sin embargo, una buena práctica y como opción de ayuda para el usuario se puede presentar una documentación detallada de la aplicación. Dicha información debe ser fácil de buscar, enfocada en la necesidad del usuario y no debe ser extensa.

Cabe aclarar que las evaluaciones heurísticas en ningún caso sustituyen las evaluaciones de usabilidad con usuarios reales. Solamente las complementan. Y es que, aunque en el global del proceso de definición, diseño e implementación de un sistema interactivo se deben tener en cuenta distintas técnicas para asegurar la usabilidad, siempre se debe considerar al usuario como centro de todo el proceso.

5.3. Prototipos generados

La aplicación a desarrollar presenta características multiplataforma, por lo tanto, para el diseño de sus interfaces se han seguido las guías de diseño de Apple Inc. para iOS y de Google Inc. para Android respecto a plataformas móviles, dando énfasis en el diseño de la aplicación, en sus colores, en la navegabilidad y en el control por parte del usuario, presentando de esta manera una interfaz sencilla e interactiva.

Los colores principales y los isologos utilizados en la aplicación han sido seleccionados en base al manual de estilo de Buenos Aires Provincia.

A continuación, se muestran los diferentes prototipos realizados, buscando la mayor satisfacción y mejor experiencia de quienes utilizarán la aplicación.

5.3.1. Pantalla de ingreso

Al iniciar la aplicación, aparecerá una splash screen y luego la pantalla de inicio de sesión (en caso de que el usuario no se encuentre ya logueado), en la cual se deberá ingresar el nombre de usuario y la contraseña.

La splash screen es una pantalla que se muestra al usuario justo al abrir la aplicación, normalmente utilizada para mostrar alguna información, mientras la aplicación carga en segundo plano los recursos y hace los preparativos necesarios para funcionar.

En el caso de la presente aplicación, se mostrará el logo institucional de la provincia de Buenos Aires junto al texto “Ministerio de Agroindustria” y “DUE” o “Documento Único Equino”.

Figura 5.2. Prototipo de splash screen e inicio de sesión.

5.3.2. Listado de solicitudes

Una vez autenticado en la aplicación móvil, si el usuario se corresponde a un veterinario habilitado, entonces el sistema mostrará una lista simple con aquellas solicitudes que posee asignadas y que ya se encuentran pagas. Por cada solicitud se deberá mostrar el o los propietarios correspondientes, el estado en el que se encuentra y la cantidad de DUE pendientes de realizar.

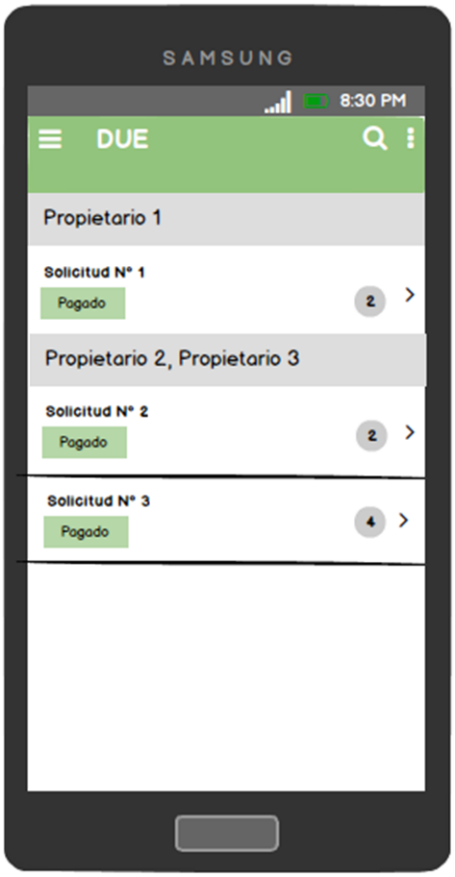


Figura 5.3. Prototipo del listado de solicitudes de un usuario.

Por otra parte, el listado de solicitudes deberá poder actualizarse y sincronizarse, de forma manual por el usuario, siempre y cuando se cuente con conexión a Internet. Para poder realizar esto, el usuario deberá "deslizar hacia abajo" con su dedo sobre el listado, mientras mantiene presionada la pantalla.

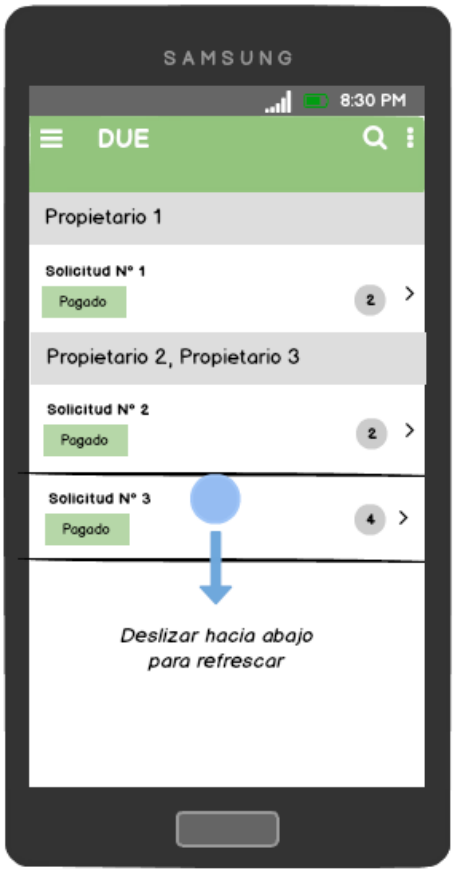


Figura 5.4. Prototipo para actualizar listado de solicitudes de un usuario.

Para administrar los DUE de una solicitud, se la debe seleccionar presionando sobre la misma.

5.3.3. Detalle de una solicitud

Una vez que se selecciona e ingresa al detalle de una solicitud en particular, la aplicación permitirá la administración de sus DUE, esto es, generar nuevos, editar su información, eliminarlos, ver su detalle y sincronizarlos con la aplicación Web.

En caso de que la solicitud no presente ningún DUE, se le indicará al usuario que dicha solicitud no posee DUE cargados. Para ingresar un nuevo DUE, se deberá presionar el botón con el signo más (+), ubicado en la parte inferior derecha de la pantalla. En caso de que se haya alcanzada la cantidad máxima permitida de DUE generados en la solicitud seleccionada, dicho botón se deshabilitará.

Figura 5.5. Prototipo de detalle de una solicitud y de la administración de sus DUE.

5.3.4. Carga DUE - Solapa de Datos Básicos

Al cargar un nuevo DUE o editar uno previamente registrado, se abre la solapa de "Datos Básicos" para comenzar o continuar completando los datos que identifican al equino. La marca (*) que tienen algunos campos, significa que deben estar completados para poder sincronizar el DUE con la plataforma Web.

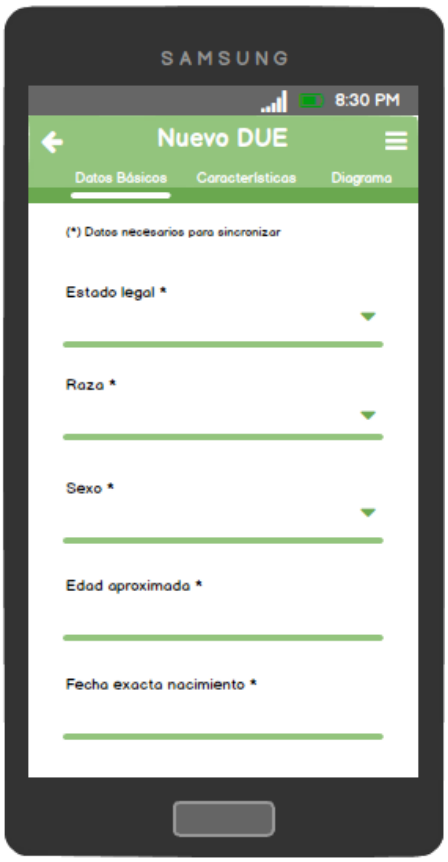


Figura 5.6. Prototipo de carga de "Datos Básicos" de un DUE.

5.3.5. Carga DUE - Solapa de Características

Desde la solapa de "Datos Básicos", el usuario puede desplazarse hacia la derecha con el dedo o presionar directamente sobre el menú superior para acceder a la solapa de "Características", donde se podrán indicar las distintas características del equino.

Para agregar una nueva característica, se deberá presionar el botón con el signo más (+) ubicado en la parte derecha de cada uno de los rótulos correspondientes. Para eliminar una característica en particular, se deberá presionar el botón de eliminar ubicado en la parte derecha de cada una de éstas. Al presionar el botón de eliminar, el usuario deberá confirmar dicha acción a través de un cuadro de diálogo.

Se decidió mostrar en todo momento los botones de acciones posibles de realizar sobre una característica equina, facilitando y simplificando la interacción del usuario con la aplicación.

Figura 5.7. Prototipo de carga de "Características" de un DUE.

Al presionar el botón de agregar una nueva característica, se desplegará una lista de selección múltiple donde el usuario deberá seleccionar las opciones correspondientes. El usuario podrá buscar una característica en particular, filtrándola desde la opción de búsqueda ubicada en la parte superior derecha de la pantalla. Luego de seleccionar las opciones deseadas, el usuario deberá volver a la pantalla anterior presionando la flecha (⇦) ubicada en la parte superior izquierda de la pantalla, o utilizando el botón físico para volver del dispositivo.

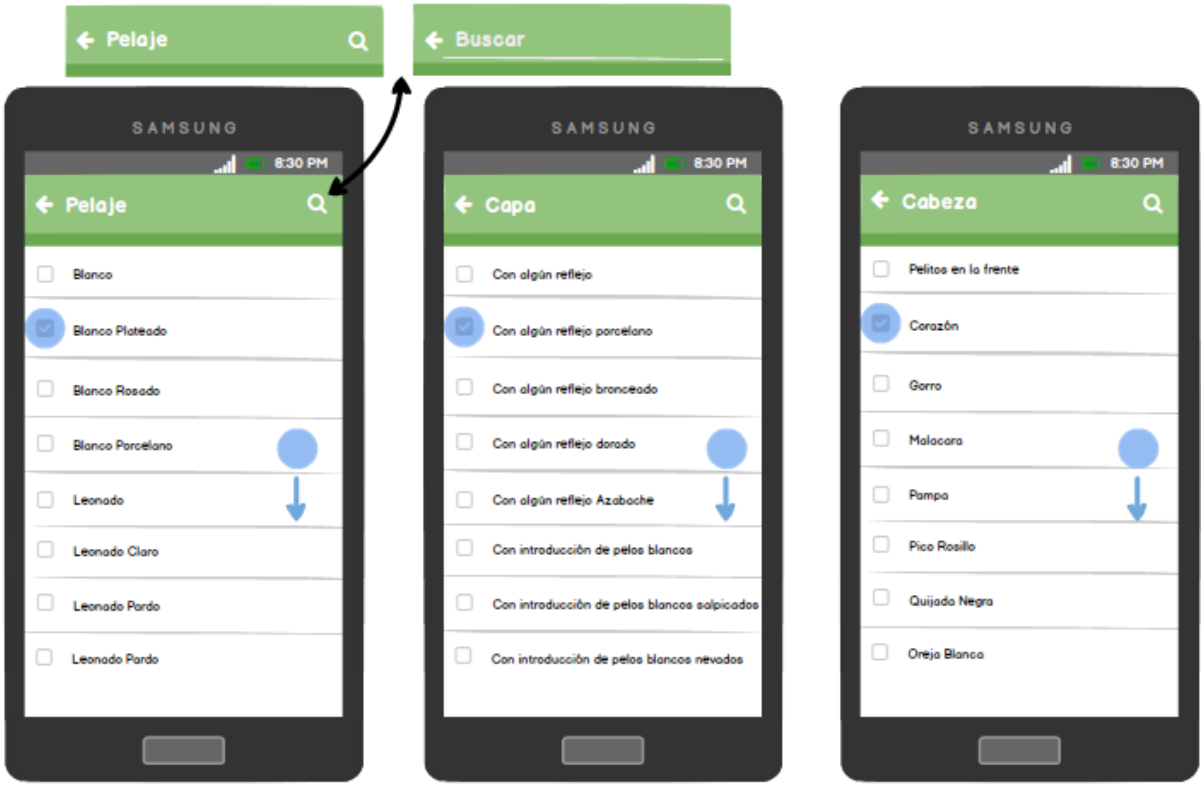


Figura 5.8. Prototipo de carga de una "Característica" particular de un DUE.

5.3.6. Carga DUE - Solapa de Diagrama

Desde la solapa de "Características", el usuario puede desplazarse hacia la derecha con el dedo o presionar directamente sobre el menú superior para acceder a la solapa de "Diagrama". Aquí se verán varias imágenes de los distintos perfiles de un equino, sobre las cuales se podrá indicar distintas características particulares que posea.

El usuario podrá agregar una nueva característica sobre un diagrama en particular. Para esto, deberá presionar el botón con el signo más (+) ubicado en la parte derecha de cada uno de los rótulos.

Al presionar el botón para agregar una nueva característica, la aplicación mostrará en pantalla el diagrama correspondiente, en formato SVG, sobre el cual el usuario podrá indicar distintas características particulares. Para realizar esto, se deberá presionar sobre el lugar específico del diagrama donde se quiere posicionar la característica. Hecho esto, se desplegará un pequeño formulario donde el usuario deberá seleccionar el tipo de característica y

opcionalmente una observación de la misma. Luego de confirmar la carga, se mostrará en el diagrama la nueva característica ingresada, utilizando un icono distintivo. Cada característica tendrá su propio icono relacionado que la distinga por sobre las demás.

Para eliminar una característica de un diagrama en particular, deberá presionar el botón de eliminar ubicado en la parte derecha de cada una de éstas, o desde el propio diagrama, presionar sobre el icono de una característica previamente cargada. En ambos casos, el usuario deberá confirmar la acción a través de un cuadro de diálogo.

Figura 5.9. Prototipo de carga en un "Diagrama" particular de un DUE.

5.3.7. Carga DUE - Solapa de Fotos

Desde la solapa de "Diagrama", el usuario puede desplazarse hacia la derecha con el dedo o presionar directamente sobre el menú superior para acceder a la solapa de "Fotos". Aquí se ingresarán las fotos del equino, particularmente una del lateral izquierdo y otra del derecho, las cuales deberán estar cargadas para poder sincronizar el DUE con la aplicación Web, ya que serán las utilizadas luego en el documento impreso del equino. También se podrán cargar cuatro fotos adicionales extra de forma opcional. En todos los casos, podrán ser obtenidas con la cámara en el momento, o ser seleccionadas desde la memoria del dispositivo.

Las fotos seleccionadas podrán ser editadas, es decir, el usuario podrá rotarlas o recortarlas a su gusto. También podrán ser eliminadas del DUE, presionando el botón de eliminación y confirmando luego en el cuadro de diálogo.

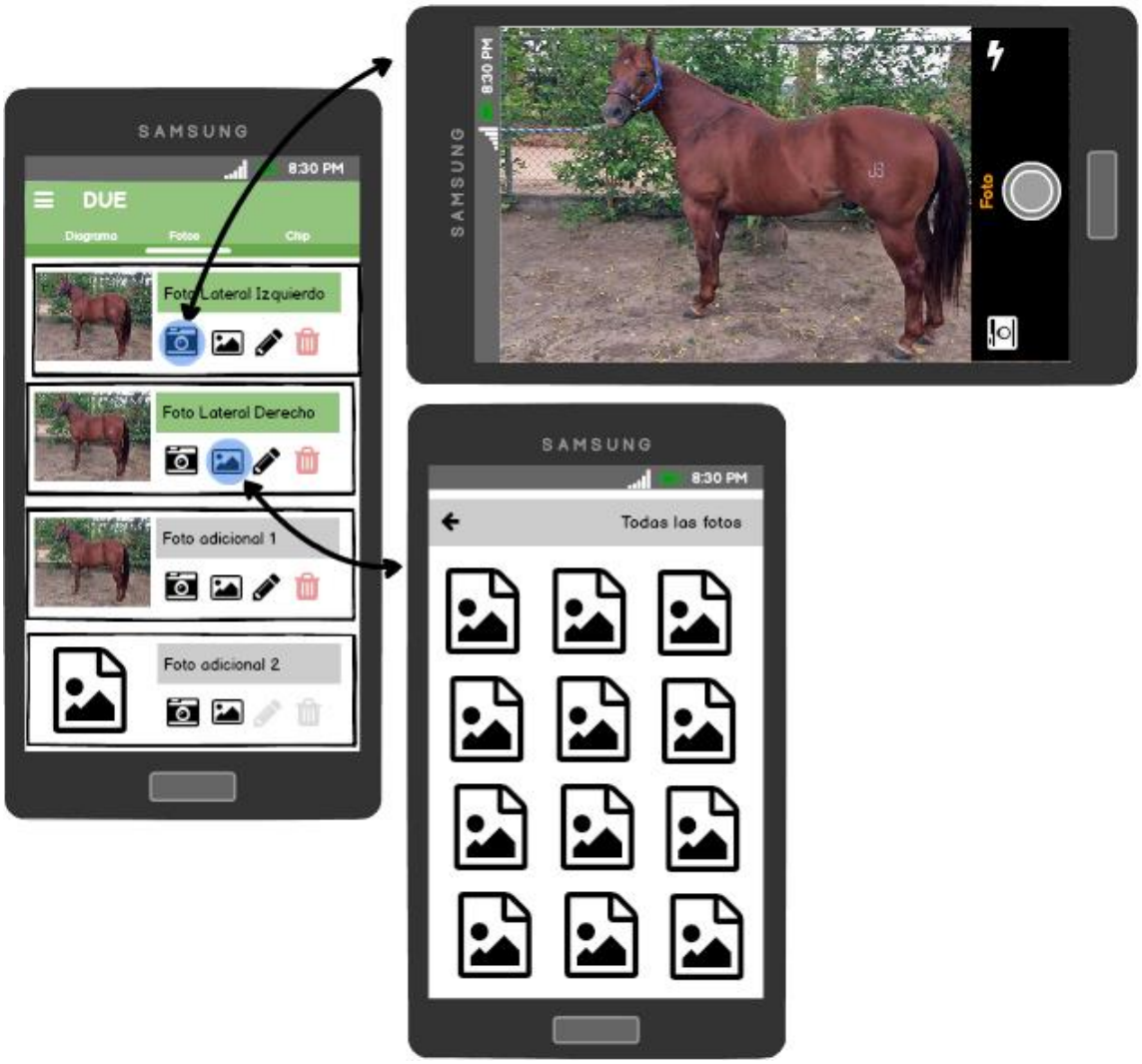


Figura 5.10. Prototipo de carga de las "Fotos" de un DUE.

5.3.8. Carga DUE - Solapa de Chip

Desde la solapa de "Fotos", el usuario puede desplazarse hacia la derecha con el dedo o presionar directamente sobre el menú superior para acceder a la solapa de "Chip", donde podrá indicar el número del microchip aplicado al equino, el cual corresponderá finalmente con su número de DUE. La única manera de obtener dicho número será a través del uso de una lectora de radiofrecuencia, vinculada de antemano con el dispositivo vía Bluetooth.

Para vincular la aplicación a la lectora, el usuario deberá presionar el botón de "Buscar" ubicado en la parte derecha de la pantalla. En caso de que el Bluetooth no se encuentre activo, se mostrará un cuadro de diálogo solicitándole al usuario activarlo. Una vez activado el Bluetooth, la aplicación buscará los dispositivos cercanos con Bluetooth activo y los mostrará en una lista simple. De dicho listado, el usuario deberá seleccionar la lectora correspondiente. También podrá reiniciar la búsqueda de dispositivos o cancelar la misma.

Figura 5.11. Prototipo de búsqueda y vinculación de una lectora.

Una vez vinculada la lectora con la aplicación, el usuario podrá leer el número de microchip desde la lectora, y el mismo será agregado al DUE correspondiente. El usuario podrá indicar si el microchip a leer es uno ya existente, es decir, aplicado en el equino con anterioridad.

Al leer un microchip, la aplicación corroborará si el mismo es válido o no, esto es, si la empresa que lo fabricó se encuentra registrada en el MAIBA y, a su vez, si se encuentra dentro de una partida habilitada por el organismo.

Por cada microchip leído se indicará si es nuevo (N) o existente (E), su número y si fue seleccionado o no como número de DUE del equino. También podrán eliminarse presionando el botón de borrar ubicado a la derecha de cada uno.

La aplicación no permitirá indicar un microchip inválido o existente como número de DUE del equino.

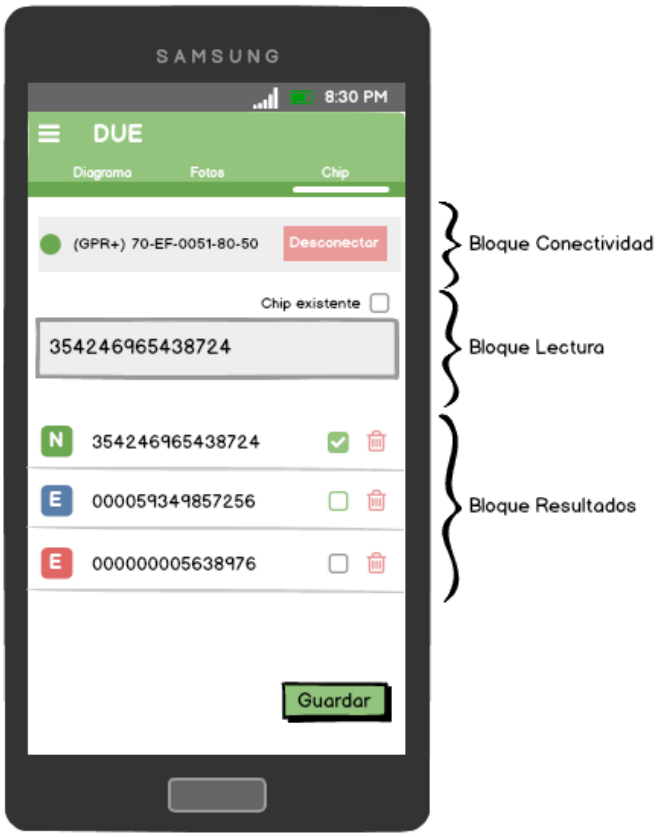


Figura 5.12. Prototipo de lectura de un microchip.

5.3.9. Guardado de un DUE

En todo momento se podrá cancelar o guardar el formulario de carga de un DUE. Esto se podrá hacer, para ambos casos, presionando el botón físico de volver del dispositivo o presionando el botón de volver (⇦) de la aplicación ubicado en la parte superior izquierda de

la pantalla, y para el caso del guardado, presionando el botón de “Guardar” ubicado en la solapa de “Chip”, en la parte inferior derecha de la pantalla.

En todos los casos el usuario deberá confirmar la acción a través de un cuadro de diálogo.

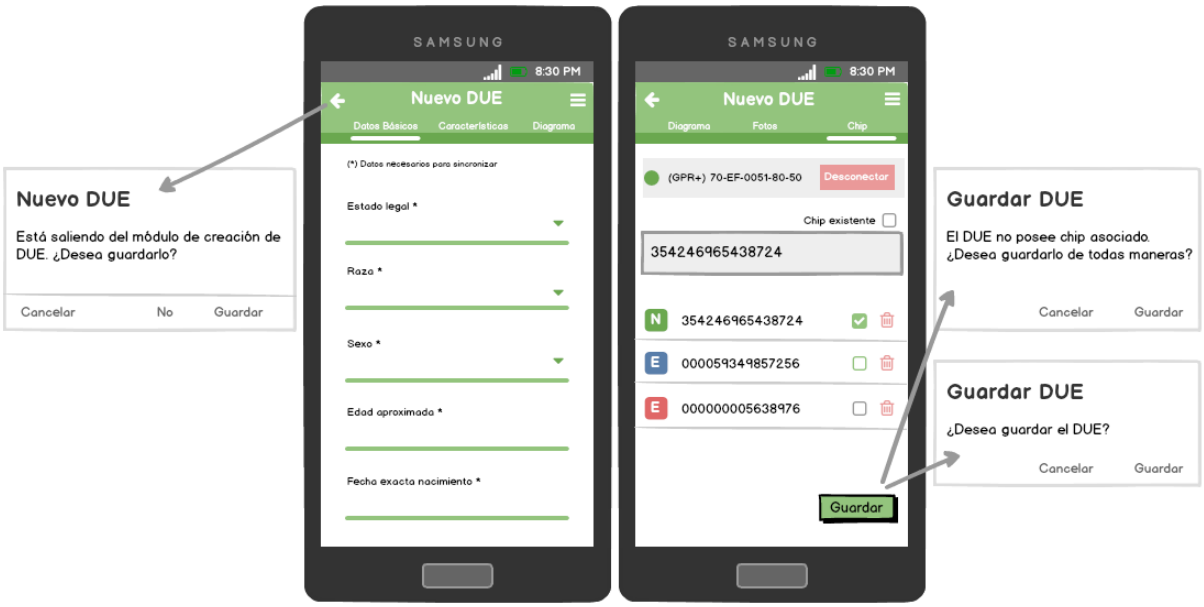


Figura 5.13. Prototipo de guardado o cancelación de un DUE.

5.3.10. Aspectos generales

Como ícono de inicio de la aplicación se utilizará el nuevo logo de la provincia de Buenos Aires.

Figura 5.14. Ícono de lanzamiento de la aplicación.

Se optó por utilizar cuadros de diálogo para confirmar toda acción realizada por el usuario que pueda implicar pérdida de información. En particular, al eliminar características de un DUE y para el guardado o cancelación del mismo.

Por otra parte, se decidió utilizar *toasts* para comunicarle al usuario los resultados de las acciones realizadas que tuvieron como consecuencia la modificación del estado de la aplicación. Un *toast* es un mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o, dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento). En el caso de la aplicación, aparecerán en la parte inferior de la pantalla, sobre un rectángulo gris ligeramente translúcido.

Por sus propias características, este tipo de notificaciones serán utilizadas para mostrar mensajes rápidos y sencillos al usuario, pero, por el contrario, al no requerir confirmación por su parte, no se utilizarán para hacer notificaciones demasiado importantes.

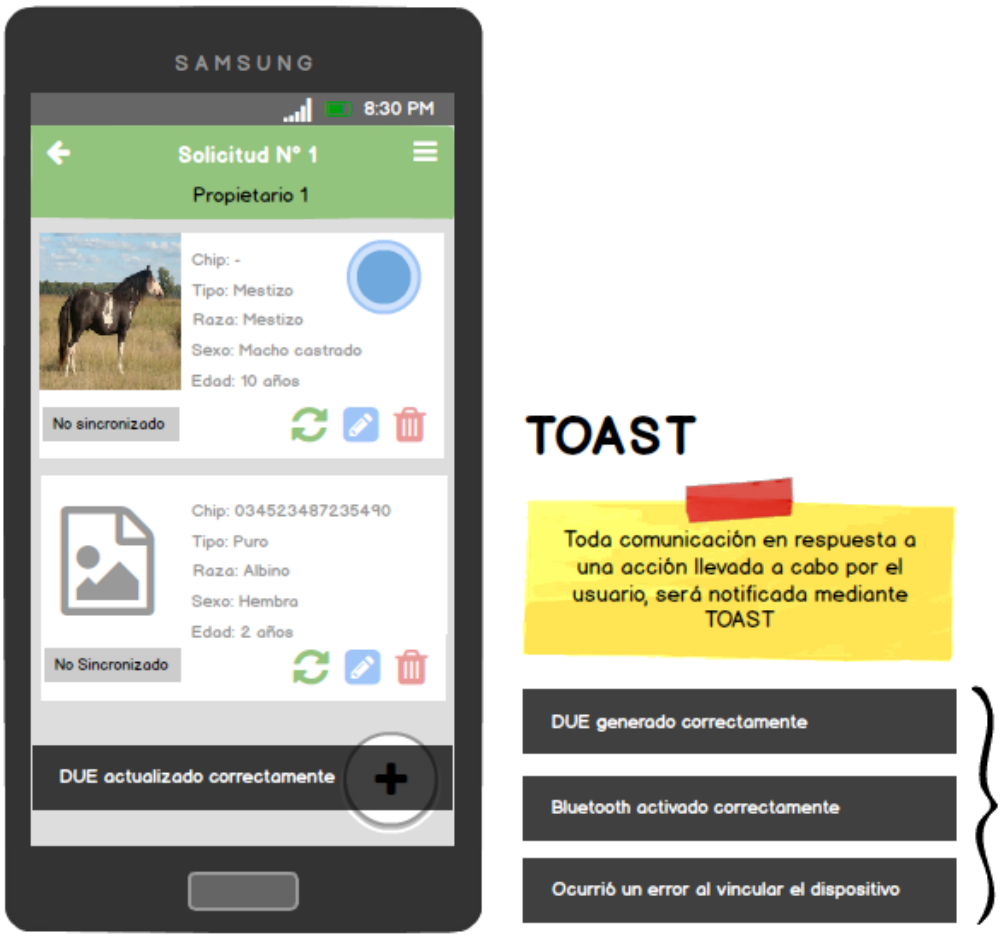


Figura 5.15. Prototipo de notificación simple en la aplicación.

6

DESARROLLO DE LA APLICACIÓN MÓVIL

6.1. Introducción

Una vez realizados los prototipos de diseño y seleccionadas las herramientas a utilizar, se prosiguió con la configuración de las mismas y con el desarrollo de la aplicación móvil, la cual será referida a partir de aquí como AppDUE.

Es importante recordar que la plataforma destino principal es Android, por lo que la configuración y el desarrollo estará enfocado en dicho sistema operativo.

6.2. Selección de API de Android

Antes de comenzar con la configuración de las herramientas, es necesario definir con cuál versión de la Application Programming Interface (API) de Android se va a trabajar. Se debe considerar que utilizar una versión específica limitará los dispositivos que podrán ejecutar la aplicación. Si se escoge una versión muy antigua, se perderían las nuevas funcionalidades de la plataforma; por el contrario, si se elige una versión demasiado reciente, serían pocos los usuarios que podrían utilizarla.

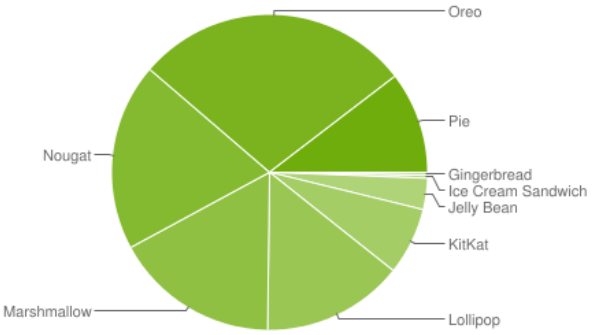
Se deben considerar tres configuraciones de nivel de API dentro de la aplicación:

- **minSdkVersion**: Versión mínima de la plataforma Android en la cual la app podrá ejecutarse, especificada por el identificador de nivel de API de la plataforma.

- **targetSdkVersion**: Especifica el nivel de API con el cual la app está diseñada para funcionar. En algunos casos, esto permite que la app use elementos del manifiesto o comportamientos definidos en el nivel de API de destino, en vez de limitarse a usar únicamente aquellos definidos por el nivel mínimo de API.
- **compileSdkVersion**: Especifica el nivel de API que Gradle debe usar para compilar la aplicación. Esto significa que la aplicación puede usar las funciones de API incluidas en este nivel de API e inferiores.

En la siguiente figura se puede observar la cantidad relativa de dispositivos que usan una versión determinada de la plataforma Android, a la fecha de mayo 2019 [28].

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%



Datos recopilados durante un período de 7 días hasta 7/5/2019.
No se muestran versiones con una distribución inferior al 0,1%.

Figura 6.1. Distribución de versiones de Android - mayo 2019.

De esta información, se desprende que el 99.4% de los celulares activos poseen una versión de Android 4.1 o superior. Considerando que React Native requiere como mínimo dicha versión para funcionar [33], se determinó utilizar la API 16 como "minSdkVersion" de la

AppDUE. Respecto a la "targetSdkVersion" y a la "compileSdkVersion", se utilizó la versión 22 y 23 respectivamente.

Cabe destacar que, a partir del 1 de agosto de 2019, Google Play exige que las apps nuevas se orienten como mínimo a Android 9.0 (nivel de API 28), y el mismo requisito será obligatorio para las actualizaciones a partir del 1 noviembre de 2019. Hasta esas fechas, las apps nuevas y las actualizaciones debieran orientarse, como mínimo, a Android 8.0 (nivel de API 26) [36].

6.3. Configuración de React Native

Antes de comenzar con el desarrollo de la AppDUE, primero fue necesario instalar los siguientes componentes:

- Node.js (versión 8.3 o superior).
- Kit de desarrollo de Java SE (JDK versión 8 o superior).
- Python (versión 2.x).
- Interfaz de línea de comandos de React Native (React Native CLI).
- Android Studio.

A continuación, se detallan los pasos que se siguieron para la instalación y configuración de dichos componentes:

Paso 1: Instalación de Node.js

Descargar e instalar la última versión LTS de Node.js desde su página oficial [29]. La versión a instalar debe ser la 8.3 o superior. La utilizada durante el desarrollo de la AppDUE fue la versión 8.3.0.

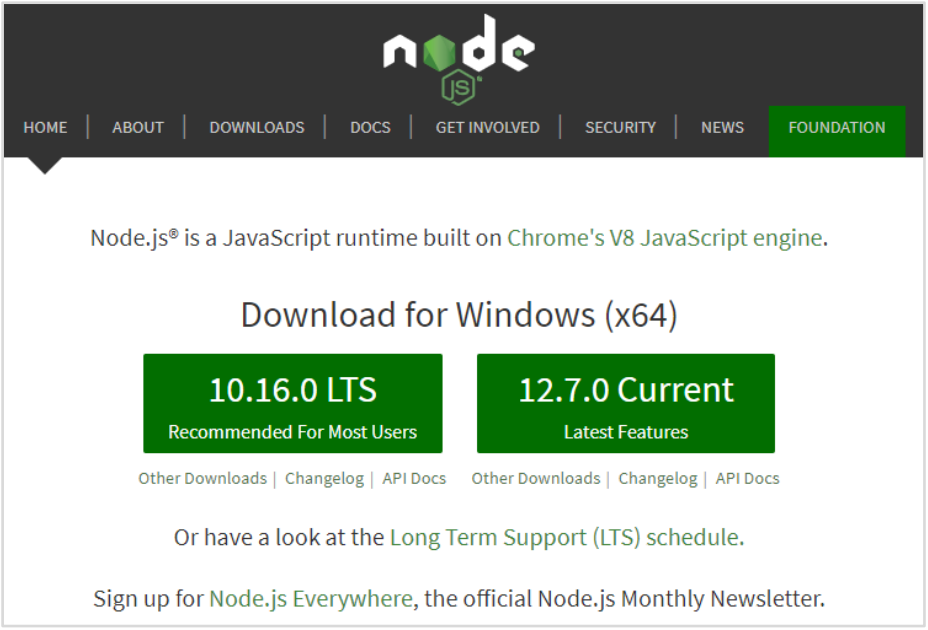


Figura 6.2. Página de descarga de Node.js.

Paso 2: Instalación de Java SE Development Kit

Descargar e instalar la última versión del kit de desarrollo de Java SE (Standard Edition) desde su página oficial de descargas [30]. La utilizada para el desarrollo de la AppDUE fue la versión 8u152.

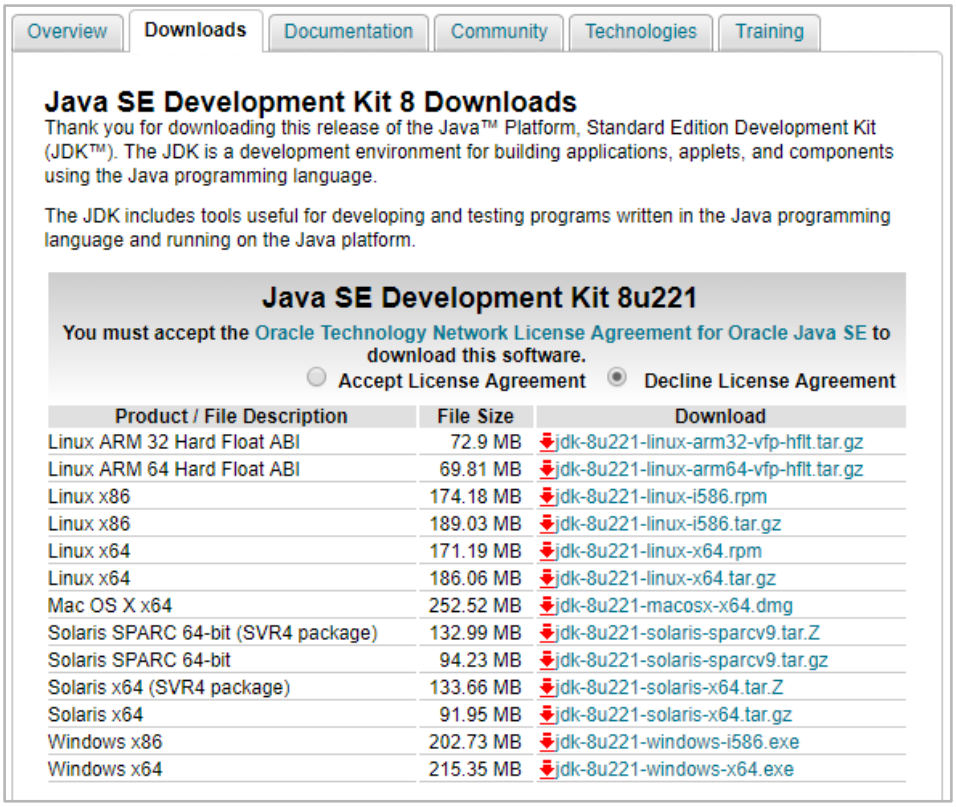


Figura 6.3. Página de descarga de Java SE Development Kit.

Paso 3: Instalación de Python 2

Descargar e instalar la última versión 2 de Python desde su página oficial de descargas [31]. La utilizada durante el desarrollo de la AppDUE fue la versión 2.7.13.

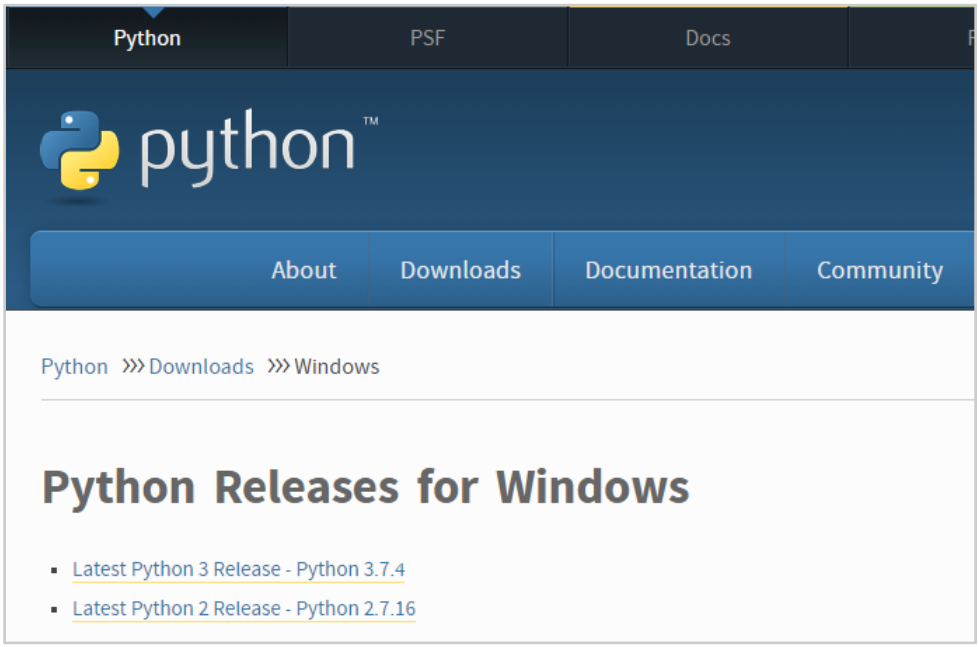


Figura 6.4. Página de descarga de Python.

Paso 4: Instalar React Native CLI

Una vez instalado Node.js junto a npm (Node Package Manager), instalar la interfaz de línea de comandos de React Native, ejecutando el siguiente código desde un intérprete de comandos, por ejemplo, el Símbolo del sistema de Windows:

```
$ npm install -g react-native-cli
```

Paso 5: Instalación de Android Studio

Descargar e instalar la última versión de Android Studio para Windows desde su página oficial [32].

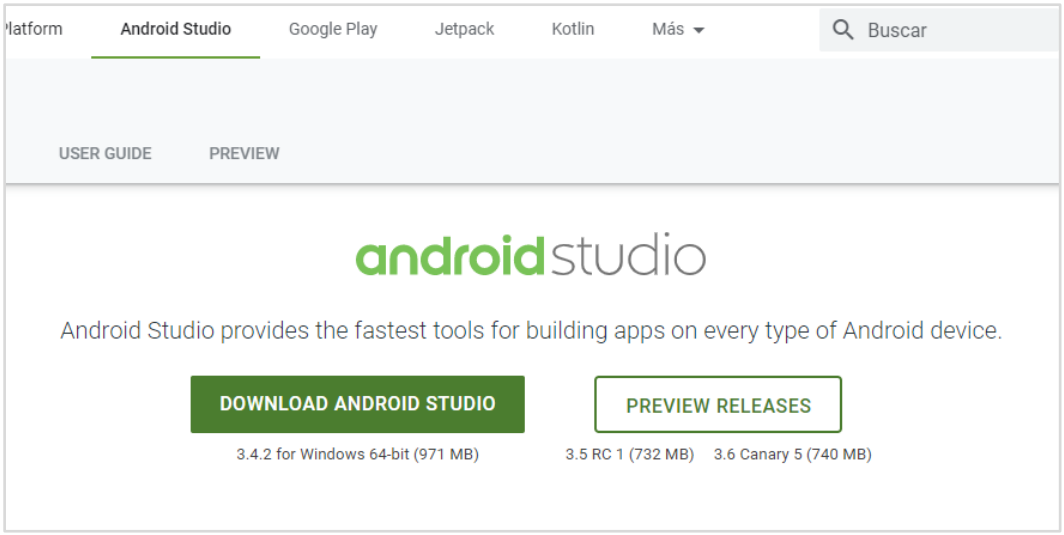


Figura 6.5. Página de descarga de Android Studio.

Cuando se solicite el tipo de instalación, seleccionar la opción “Custom” y asegurarse de tener marcados los siguientes componentes:

- *Android SDK*
- *Android SDK Platform*
- *Performance (Intel ® HAXM)*
- *Android Virtual Device*

Paso 6: Instalación de SDK de Android

Android Studio instala el último SDK de Android de forma predeterminada, sin embargo, se pueden instalar SDK adicionales a través del Administrador de SDK. Se puede acceder allí desde la pantalla de bienvenida de Android Studio, haciendo clic en "Configure" y luego seleccionando "SDK Manager".

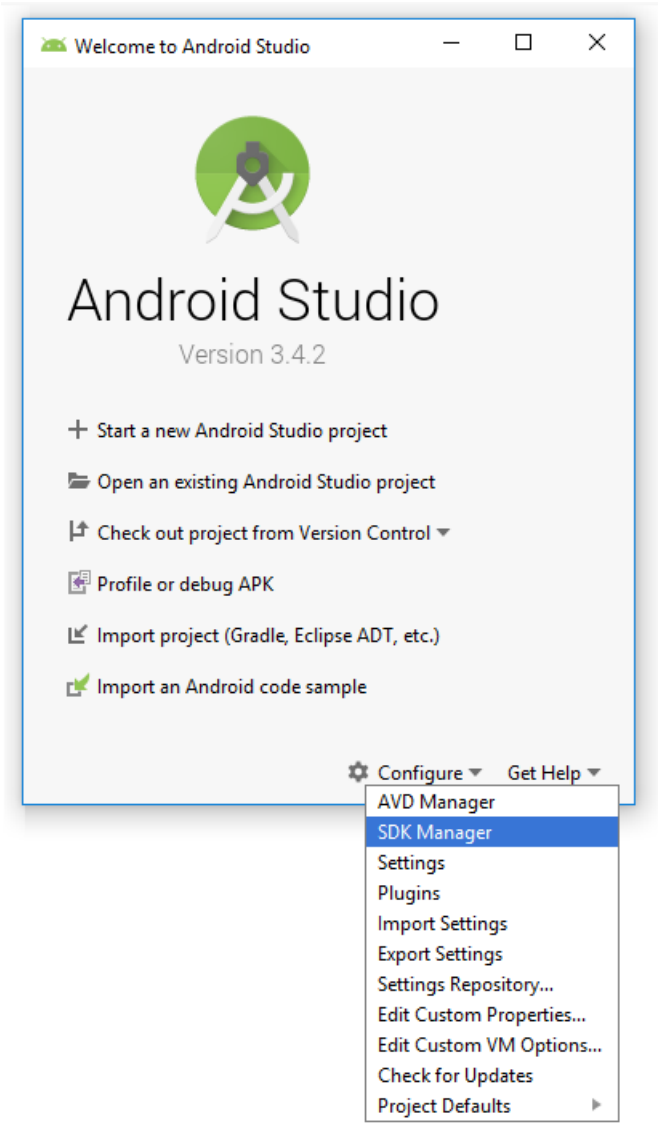


Figura 6.6. Pantalla de bienvenida de Android Studio.

Desde el Administrador de SDK, seleccionar la pestaña "SDK Platforms" y marcar la casilla "Show Package Details", ubicada en la esquina inferior derecha. Luego buscar y expandir la opción de Android a utilizar (en el caso de la AppDUE se utilizó Android 6.0), y asegurarse de que los siguientes elementos se encuentran marcados:

- *Android SDK Platform 23*
- *Intel x86 Atom_64 System Image* o *Google APIs Intel x86 Atom System Image*

A continuación, seleccionar la pestaña "SDK Tools" y marcar aquí también la casilla "Show Package Details". Luego buscar y expandir la entrada "Android SDK Build-Tools" y

asegurarse de que la versión deseada se encuentra seleccionada. En el caso de la AppDUE, se utilizó la versión 23.0.1.

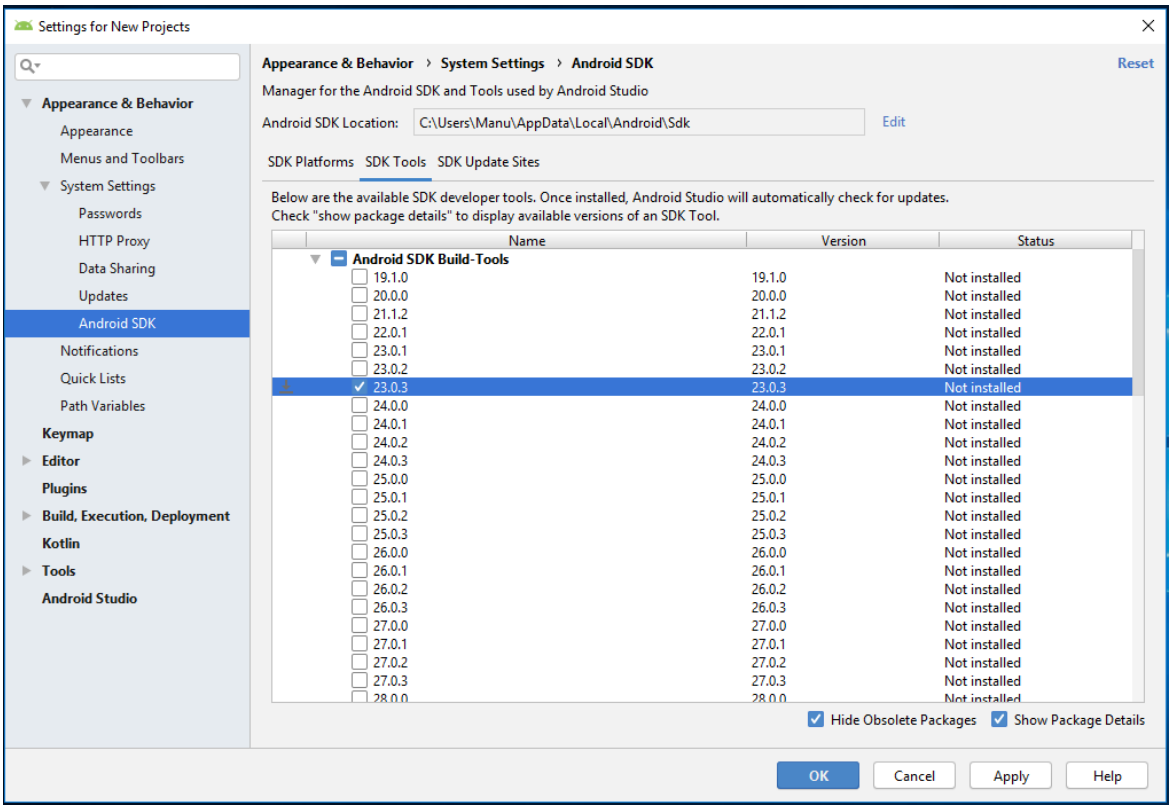


Figura 6.7. Pantalla de configuración de las herramientas del SDK.

Finalmente, hacer clic en "OK" para descargar e instalar el SDK de Android y las herramientas de compilación relacionadas.

Paso 7: Configuración de variables de entorno

React Native requiere que se configuren algunas variables de entorno para desarrollar aplicaciones con código nativo:

- Variable ANDROID_HOME: Desde el administrador de variables de entorno de Windows, agregar la variable de usuario ANDROID_HOME, cuyo valor debe ser la ruta de la carpeta de instalación del SDK de Android. Por defecto dicha ruta es "*C:\Users\NOMBRE_USUARIO\AppData\Local\Android\Sdk*".

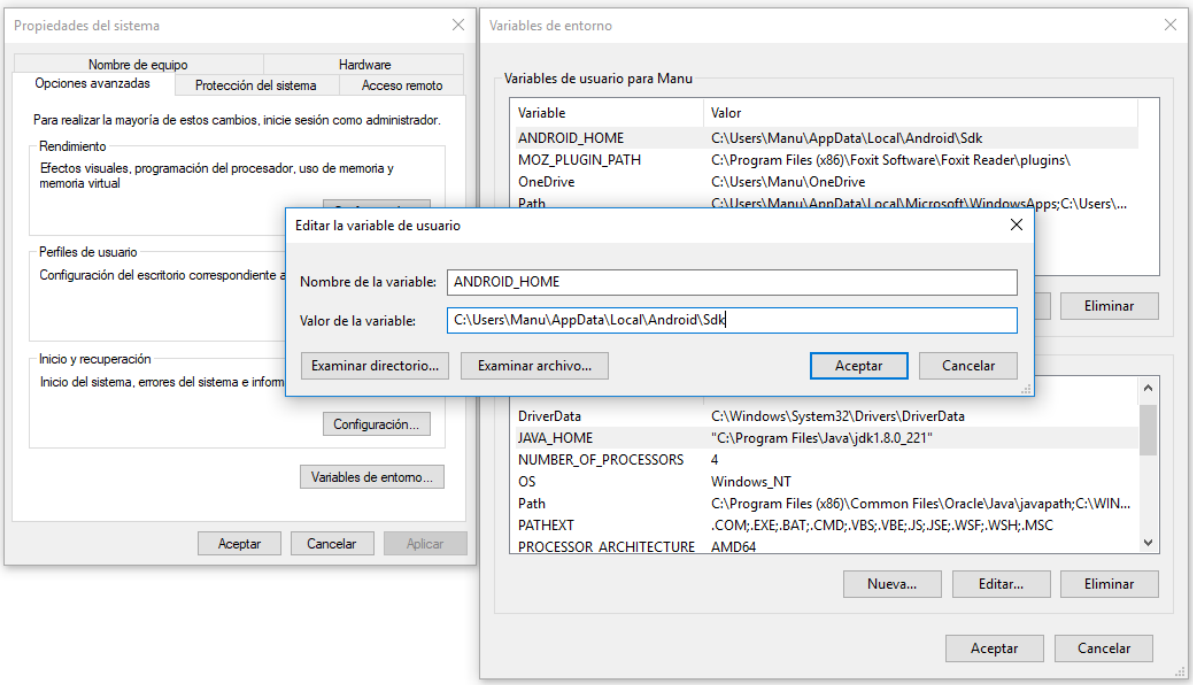


Figura 6.8. Ejemplo de configuración de la variable ANDROID_HOME.

- Variable JAVA_HOME: Desde el administrador de variables de entorno de Windows, agregar la variable de sistema JAVA_HOME, cuyo valor debe ser la carpeta de instalación del JDK. Dicha variable deberá ser agregada al final de la variable del sistema "Path".

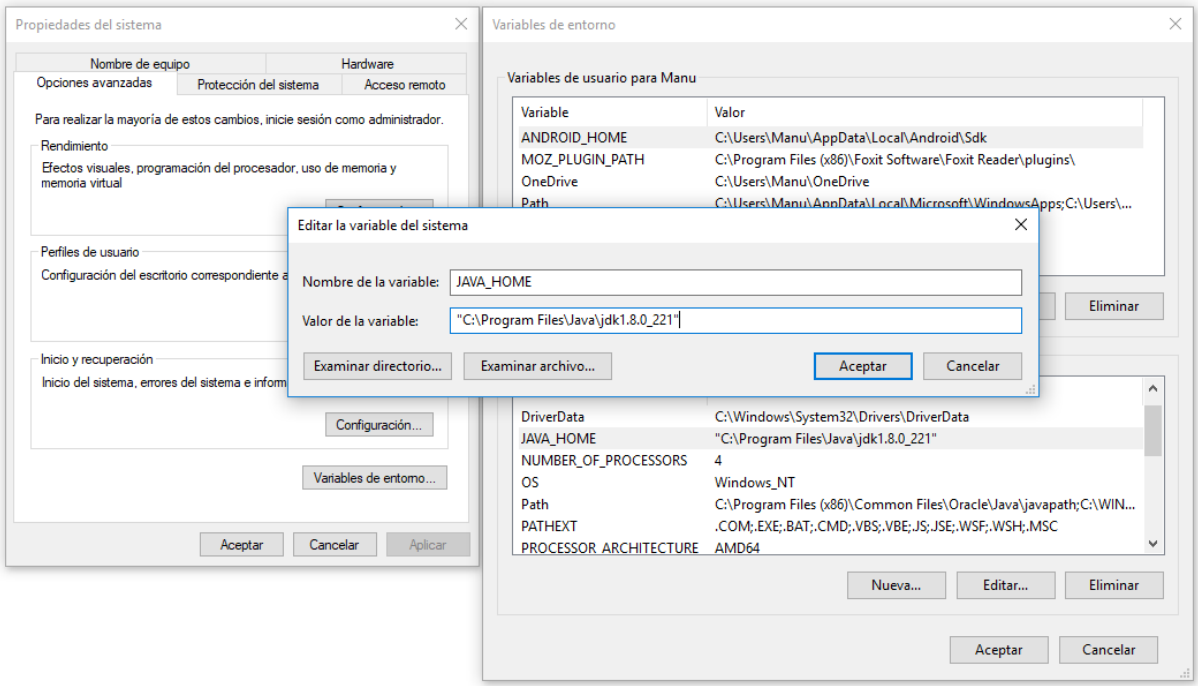


Figura 6.9. Ejemplo de configuración de la variable JAVA_HOME.

- Android platform-tools: Desde el administrador de variables de entorno de Windows, agregar a la variable de sistema "Path", la ruta de la carpeta platform-tools de Android. Por defecto se encuentra ubicada en "*C:\Users\NOMBRE_USUARIO\AppData\Local\Android\Sdk\platform-tools*".

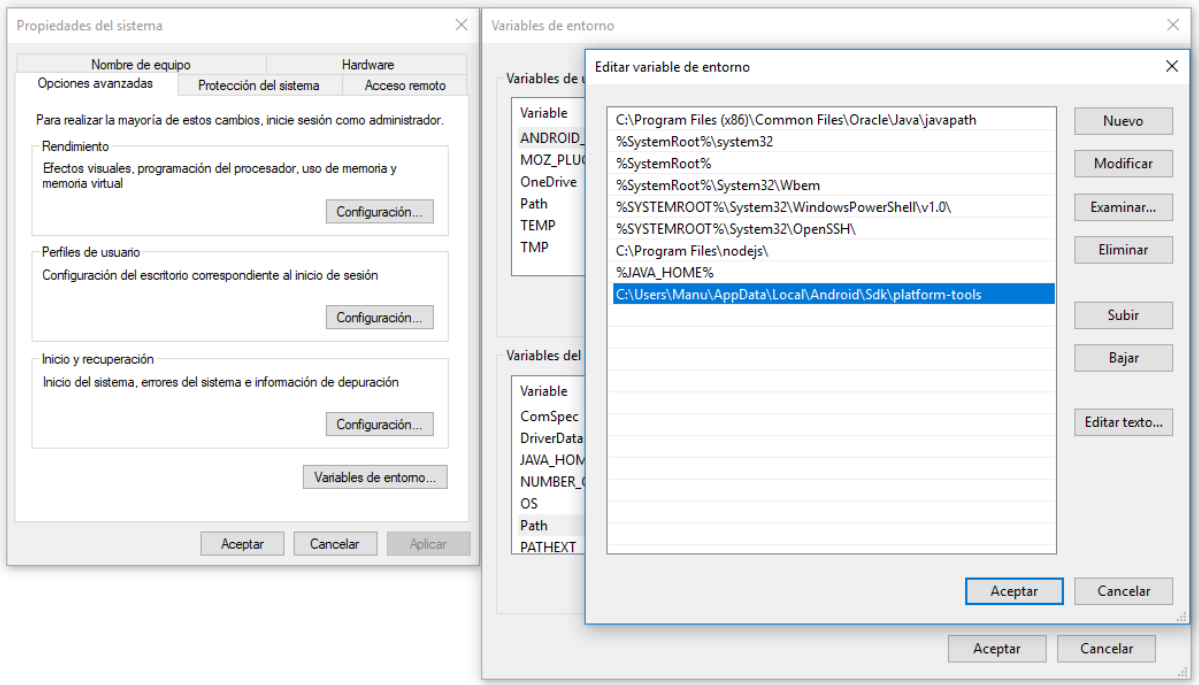


Figura 6.10. Ejemplo de configuración de "platform-tools" en la variable Path.

6.4. Configuración del entorno de desarrollo

El sistema operativo sobre el cual se desarrolló la AppDUE para Android fue Windows 10 de 64 bits (x64).

Como entorno de desarrollo se utilizó JetBrains WebStorm [47], un IDE de desarrollo para JavaScript, creado por la empresa JetBrains. WebStorm proporciona soporte completo para JavaScript, TypeScript, HTML y CSS, así como para frameworks tales como React, Angular y Vue.js, sin la necesidad de complementos adicionales. También ofrece soporte para desarrollar aplicaciones del lado servidor con Node.js, para aplicaciones móviles con React Native o Cordova, y para aplicaciones de escritorio con Electron.

6.5. Creación del proyecto

El primer paso para desarrollar la AppDUE fue crear un nuevo proyecto. Para hacer esto, desde el Símbolo del sistema de Windows, se debe navegar a la carpeta donde se desea almacenar el mismo y ejecutar el siguiente comando indicando el nombre que tendrá el proyecto, en este caso “DUE”:

```
$ react-native init DUE
```

Para el desarrollo de la AppDUE se utilizó la versión 0.47.1 de React Native. Si se quisiera usar una versión específica de la herramienta, se puede utilizar el argumento “*--version*” para indicarla. Por ejemplo:

```
$ react-native init DUE --version 0.47.1
```

Finalizada la ejecución de dicho comando, se habrá creado el nuevo proyecto con la siguiente estructura interna de carpetas:

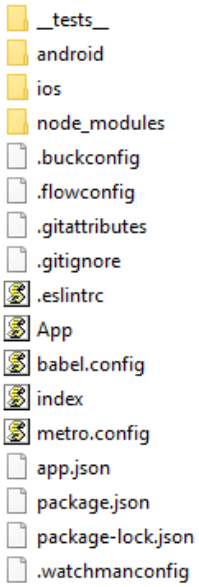


Figura 6.11. Estructura de carpetas de un proyecto React Native.

Antes de ejecutar el proyecto, primero es necesario configurar el emulador de Android. Para esto, abrir el Android Studio, seleccionar la opción "Open an existing Android Studio project" e indicar la carpeta “android” del proyecto recién generado. Luego hacer clic en “OK”.

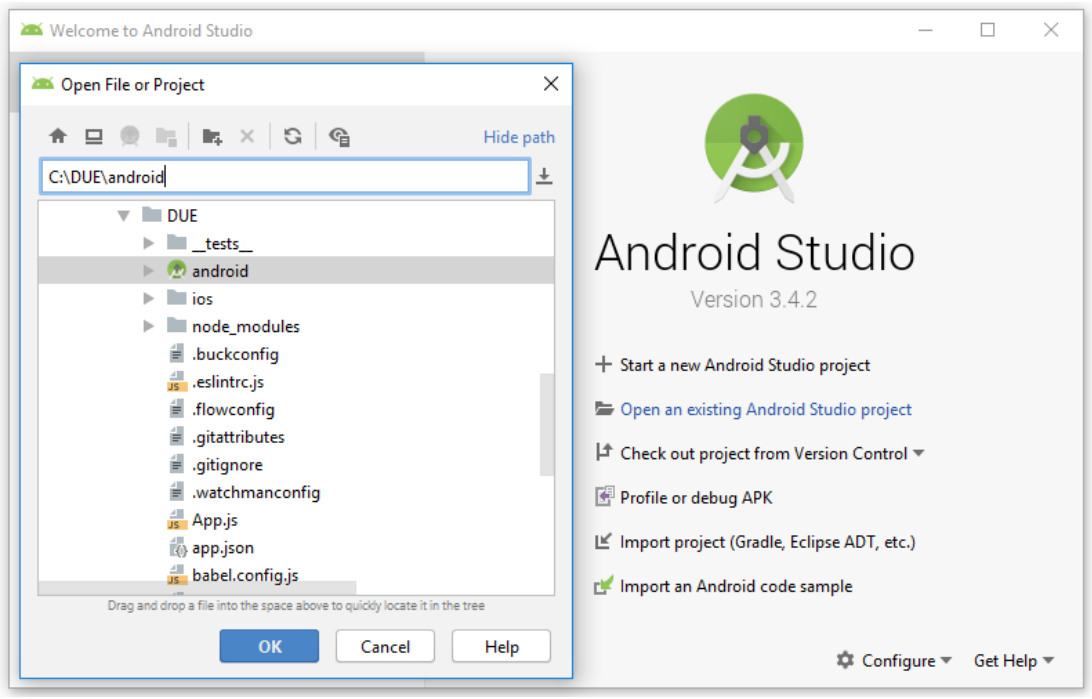


Figura 6.12. Ejemplo para configurar el proyecto en Android Studio.

Una vez cargado el proyecto, desde el menú superior seleccionar la opción "Tools" → "AVD Manager" para abrir el administrador de dispositivos virtuales de Android, donde aparecerán los dispositivos virtuales existentes. En caso de necesitar crear uno nuevo, hacer clic en "Create Virtual Device..." y seguir los pasos para su configuración.

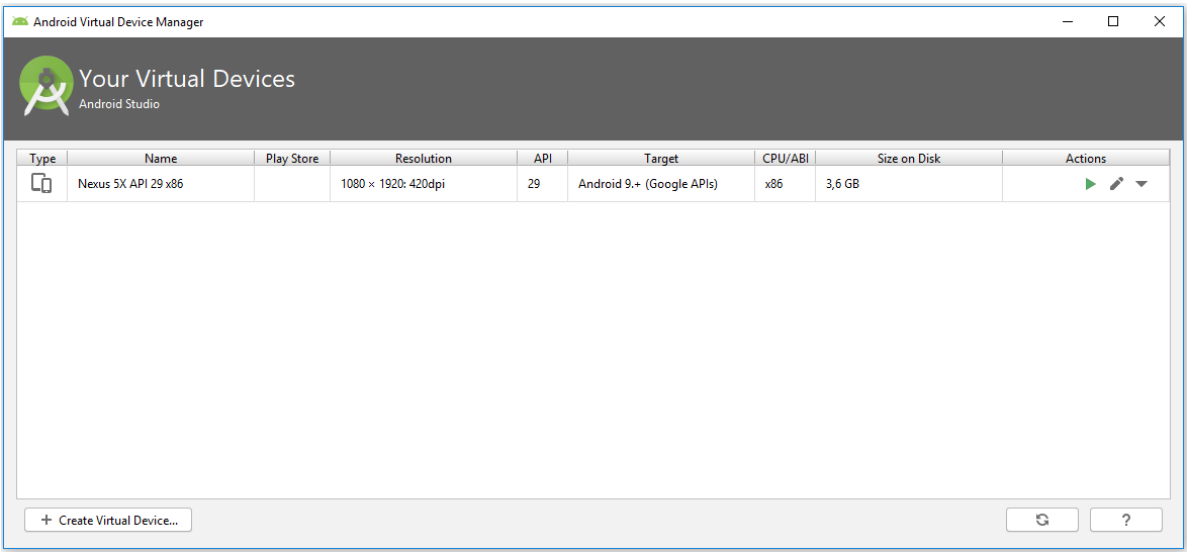


Figura 6.13. Ejemplo del administrador de dispositivos virtuales de Android.

De la lista de dispositivos virtuales, se deberá iniciar aquel sobre el cual se quiera ejecutar la aplicación, presionando sobre el botón “Play” ubicado en el menú de acciones a la derecha del mismo. Una vez hecho esto, se podrá iniciar el nuevo proyecto, ejecutando desde el Símbolo del sistema de Windows el siguiente código dentro de la carpeta raíz:

```
$ react-native run-android
```

Si todo fue configurado correctamente, se debería ejecutar la nueva aplicación sobre el emulador de Android, de la siguiente manera:

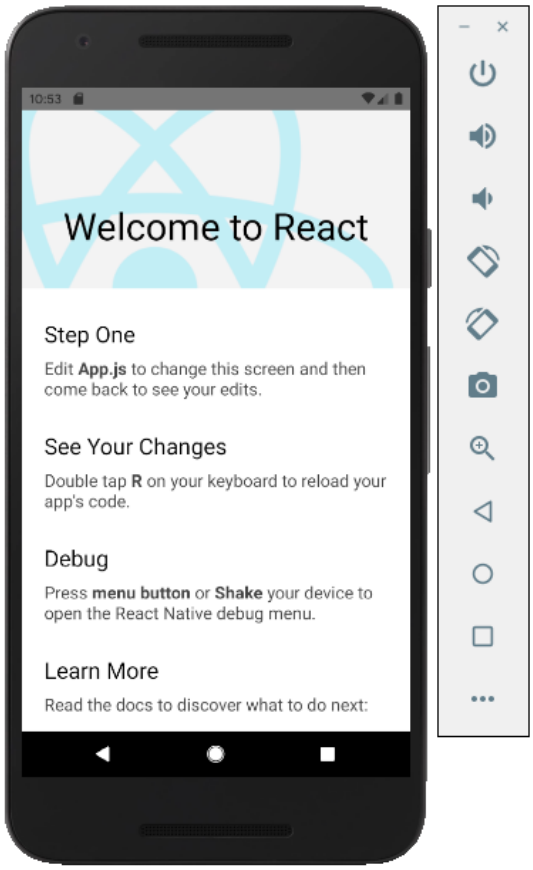


Figura 6.14. Ejemplo de proyecto React Native ejecutándose en emulador de Android.

Una vez configurado React Native, es necesario instalar las dependencias de MobX que se utilizarán para el desarrollo. Para esto, se debe ejecutar desde el Símbolo del sistema de Windows, el siguiente código dentro de la carpeta raíz del proyecto:

```
$ npm i mobx mobx-react –save
```

En el caso de la AppDUE, también se necesitará instalar un complemento para Babel 6, para poder utilizar los decoradores de ECMAScript 6 (ES6):

```
$ npm i babel-plugin-transform-decorators-legacy --save-dev
```

Por último, se debe actualizar el archivo `.babelrc` para configurar los complementos de Babel instalados:

```
{
  "presets": ["react-native"],
  "plugins": ["transform-decorators-legacy"]
}
```

En este punto, ya es posible comenzar con el desarrollo de la aplicación.

6.6. Estructura del proyecto

Una vez configuradas las herramientas a utilizar y creado el proyecto, se prosiguió con la definición de la estructura interna del mismo. Dentro de la carpeta raíz del proyecto, se creó la carpeta "**app**", en la cual se incluyeron todos los archivos correspondientes al desarrollo de la AppDUE. El contenido de ésta ha sido estructurado de la siguiente manera:

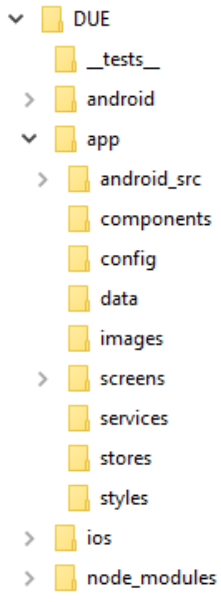


Figura 6.15. Estructura del proyecto.

A modo de resumen, se puede decir que cada carpeta agrupa la siguiente información:

- **app/android_src:** Contiene las copias de todos los archivos que fueron modificados en la carpeta "android", con el fin de facilitar el versionado.
- **app/components**: Contiene todos los componentes reutilizables dentro de la aplicación.
- **app/config:** Contiene los archivos de configuración de la aplicación.
- **app/data**: Contiene los archivos de datos utilizados localmente por la aplicación.
- **app/images**: Contiene las imágenes locales utilizadas en la aplicación.
- **app/screens**: Contiene los componentes de presentación para cada una de las pantallas de la aplicación.
- **app/services**: Contiene los archivos encargados de la comunicación externa y sincronización de la aplicación.
- **app/stores**: Contiene cada uno de los stores definidos en la aplicación.
- **app/styles**: Contiene los archivos de estilos utilizados en la aplicación.
- **app/screens/Root.js**: Será el punto de entrada multiplataforma de la AppDUE.

En cuanto a las demás carpetas, los directorios "**android"** e "**ios"** contienen los respectivos proyectos nativos de la aplicación. El directorio "**node_modules"** almacena los módulos npm utilizados por la aplicación, y la gestión y configuración general de éstos está declarada en el fichero `package.json`. El directorio "**__tests__"** contiene los test de la aplicación.

Para configurar correctamente el punto de entrada de la aplicación, es necesario modificar los archivos `index.android.js` e `index.ios.js`, para que, en el caso de la AppDUE, registren al componente raíz `Root.js`. Por ejemplo, en el caso de Android:

```
#index.android.js

import React from 'react';
import {AppRegistry} from 'react-native';

import Root from './app/screens/Root';

AppRegistry.registerComponent('due', () => Root);
```

`AppRegistry` es el punto de entrada para ejecutar todas las aplicaciones React Native. El componente raíz de la aplicación debe registrarse a través del comando `AppRegistry.registerComponent`, el cual lo conectará al `main.m` en iOS y al `MainActivity.java` y `MainApplication.java` en Android. En el caso de la AppDUE, el archivo `main.m` se lo puede encontrar en la carpeta "./ios/due" y los archivos Java en la carpeta "android/app/src/main/java/com/due".

NOTA: A partir de la versión 0.49 de React Native, las rutas de entrada separadas `index.android.js` e `index.ios.js` ya no son compatibles. En cambio, se debe utilizar la entrada `index.js`, la cual por defecto hace referencia al componente raíz "`./App.js`". [37]

6.7. Definición de stores

Un store se encarga de administrar el estado de un dominio particular de la aplicación, a través de un conjunto de estructuras de datos y de métodos para la recuperación de dichos datos y para responder a las llamadas del dispatcher. La responsabilidad principal de los stores es mover la lógica y el estado fuera de los componentes hacia una unidad de prueba independiente.

Para la AppDUE se han definido múltiples stores, cada uno según el contexto de dominio al cual se enfocan. Estos son:

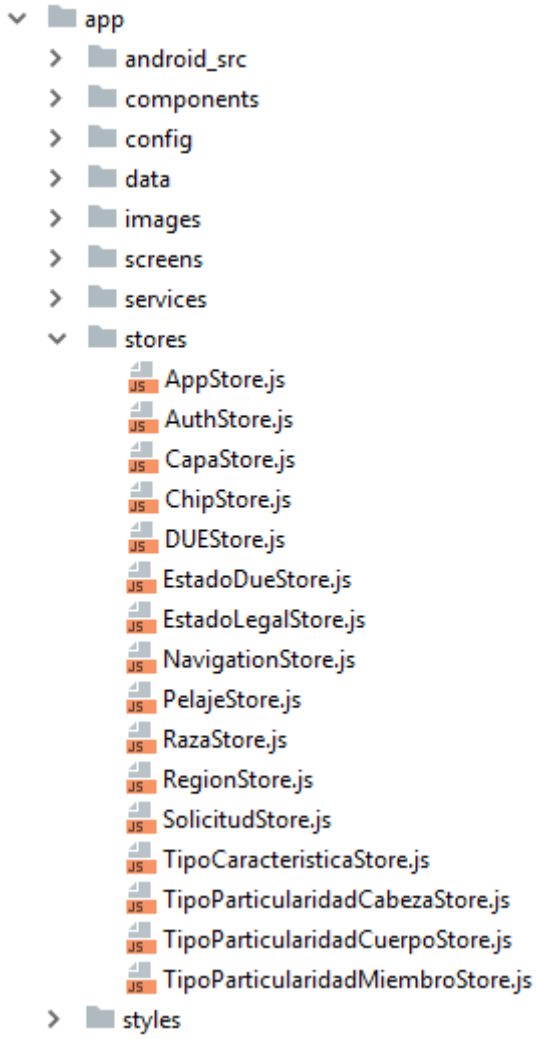


Figura 6.16. Contenido de la carpeta "store" del proyecto.

AppStore: Es el store raíz ya que centraliza el acceso a los demás stores y administra el estado general de la aplicación, por ejemplo, si se encuentra o no iniciada en su totalidad, si posee o no conexión a Internet o si se encuentra cargando información. También se encarga de validar si la versión instalada por el usuario es la última publicada; en caso de que no lo sea, la aplicación indicará si es necesario o no realizar una actualización para continuar. Si la versión instalada permite utilizar la aplicación, el AppStore se ocupa de la inicialización de la misma, obteniendo la información necesaria para operar, tanto la almacenada localmente a través del `AsyncStorage`, como la de la aplicación Web a través de sus Web Services. Por último, se encarga de administrar las notificaciones generales del sistema.

AuthStore: Administra la información básica del usuario, esto es, su id, nombre de usuario y token de seguridad. A su vez, contiene los métodos para validar el inicio de sesión a la AppDUE utilizando los WS de la aplicación Web, y para cerrar la sesión del usuario, eliminando localmente todos los datos almacenados. También posee los métodos para validar la versión instalada de la aplicación móvil, invocados por el AppStore al momento de la inicialización.

CapaStore: Se encarga de administrar los tipos de capa de un equino, esto es, obtenerlos a través de los WS de la aplicación Web y almacenarlos localmente a través del `AsyncStorage`.

ChipStore: Se encarga de administrar las partidas de microchips habilitadas por el MAIBA, esto es, obtenerlas a través de los WS de la aplicación Web y almacenarlas localmente a través del `AsyncStorage`. A su vez, define el modelo de datos de un microchip, compuesto por un id, número, fecha de lectura, estado y si existía previamente o no en el equino.

DUEStore: Define el modelo de datos de un DUE y se encarga de administrar aquellos generados por el usuario desde la app. A su vez, indica si un DUE es sincronizable o no, y centraliza las validaciones realizadas al vincular un microchip a un DUE, verificando por ejemplo si el número de microchip leído no se encuentra ya utilizado localmente por otros DUE, o si el DUE ya posee un microchip previamente vinculado o si el microchip leído no se encuentra dentro de un rango válido.

EstadoDueStore: Contiene los estados posibles de un DUE, los cuales obtiene de forma local a través del archivo `estado_due.json`, ubicado dentro de la carpeta “data”.

EstadoLegalStore: Contiene los estados legales de un equino, los cuales obtiene de forma local a través del archivo `estado_legal.json`, ubicado dentro de la carpeta “data”.

NavigationStore: Centraliza la navegabilidad de la aplicación, definida principalmente en el componente `AppNavigator`, dentro de la carpeta “components”.

PelajeStore: Se encarga de administrar los tipos de pelaje de un equino, esto es, obtenerlos a través de los WS de la aplicación Web y almacenarlos localmente a través del `AsyncStorage`.

RazaStore: Se encarga de administrar las razas de un equino, esto es, obtenerlas a través de los WS de la aplicación Web y almacenarlas localmente a través del `AsyncStorage`.

RegionStore: Contiene las regiones del cuerpo de un equino, las cuales obtiene de forma local a través del archivo `region.json`, ubicado dentro de la carpeta "data".

SolicitudStore: Define el modelo de datos de una solicitud, compuesto por un id, sus propietarios, su estado, los DUE que posee y la cantidad restante de DUE a generar. Se encarga de administrar las solicitudes, esto es, obtenerlas a través de los WS de la aplicación Web y almacenarlas localmente a través del `AsyncStorage`.

TipoCaracteristicaStore: Contiene los tipos de características de un equino, los cuales obtiene de forma local a través del archivo `tipo_caracteristica_equino.json`, ubicado dentro de la carpeta "data".

TipoParticularidadCabezaStore: Se encarga de administrar los tipos de particularidad de la cabeza de un equino, esto es, obtenerlos a través de los WS de la aplicación Web y almacenarlos localmente a través del `AsyncStorage`.

TipoParticularidadCuerpoStore: Se encarga de administrar los tipos de particularidad del tronco de un equino, esto es, obtenerlos a través de los WS de la aplicación Web y almacenarlos localmente a través del `AsyncStorage`.

TipoParticularidadMiembroStore: Se encarga de administrar los distintos tipos de particularidades posibles de las manos y patas de un equino, esto es, obtenerlos a través de los WS de la aplicación Web y almacenarlos localmente a través del `AsyncStorage`.

6.8. Almacenamiento de datos

Si bien todos los datos utilizados o generados dentro de la AppDUE se almacenan de forma local, se pueden diferenciar dos tipos distintos, los estáticos y los dinámicos.

Los datos estáticos son aquellos que ya se encuentran incluidos en la aplicación y que no pueden ser modificados ni por el sistema ni por el usuario, es decir, son utilizados en modo de "sólo lectura". Estos datos se encuentran almacenados en distintos archivos JSON, dentro de la carpeta "data" del proyecto. Es siempre la misma y no debería variar a lo largo del tiempo, pero en caso de que lo haga, será necesario actualizarla y compilar nuevamente la aplicación para reflejar las modificaciones en una nueva versión. En la siguiente imagen se puede ver cuáles son estos archivos:

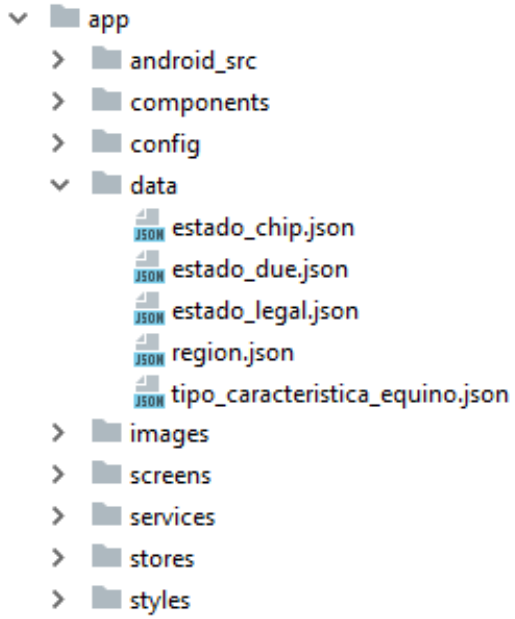


Figura 6.17. Contenido de la carpeta "data" del proyecto.

Cada archivo es una colección de pares de la forma <nombre>:<valor>, especificando el `id` y la `denominación` del objeto. Los valores de dichos campos se corresponden con los utilizados en la base de datos de la aplicación Web. Por ejemplo, el contenido del archivo `estado_legal.json` es el siguiente:

```
{
 "RECORDS": [
   {
     "id": 1,
     "denominacion": "Puro"
   },
   {
     "id": 2,
     "denominacion": "Mestizo"
   }
 ]
}
```

Por otro lado, se encuentra la información dinámica, la cual surge como consecuencia del uso de la aplicación o es obtenida dinámicamente desde la aplicación Web a través de sus WS. Esta información es administrada y almacenada por el `AsyncStorage`, a excepción de las imágenes cargadas por el usuario, las cuales se almacenan directamente en el sistema de archivos, a través del uso de la librería `RNFS` de React Native.

`AsyncStorage` es un sistema de almacenamiento simple, no cifrado y asíncrono del tipo clave-valor, global a toda la aplicación. Cada uno de sus métodos están basados en promesas, por lo tanto, se debe usar la sintaxis `async/yield` para ejecutarlos de forma correcta. Para utilizar `AsyncStorage`, es necesario importarla. En el caso de la AppDUE se utilizó la siguiente librería:

```
import {AsyncStorage} from 'react-native';
```

Una de las formas de persistir datos es a través del método "`setItem`". Por ejemplo, al almacenar las solicitudes:

```
async afterAttach() {
  reaction(() => getSnapshot(self.solicitudes), solicitudes => {
      try {
         AsyncStorage.setItem("solicitudes", JSON.stringify(solicitudes));
      } catch (error) {
         console.log('Ha ocurrido un error: ' + error);
      }
  })
}
```

Para recuperar datos previamente almacenados, se debe hacer uso del método "`getItem`". Por ejemplo, para recuperar las solicitudes almacenadas:

```
const solicitudes = yield AsyncStorage.getItem('solicitudes');

if (solicitudes !== null) {
    self.store.solicitudStore.setSolicitudes(JSON.parse(solicitudes));
}
```

Una de las limitaciones del `AsyncStorage` en Android es el tamaño de almacenamiento máximo permitido, el cual, al momento de escribir la presente tesina, es de 6 MB [39]. Por tal motivo, las imágenes cargadas por el usuario se almacenan directamente en el sistema de archivos del dispositivo, a través del uso de la librería `RNFS`.

En el caso de iOS, el tamaño del almacenamiento no se encuentra limitado programáticamente.

NOTA: Actualmente la librería de `AsyncStorage` utilizada en la app se encuentra deprecada. En su lugar se debe usar:

```
import AsyncStorage from '@react-native-community/async-storage';
```

6.9. Esquema de sincronización

A continuación, se detalla el esquema de sincronización de datos entre la aplicación móvil y la aplicación Web a través de sus Web Services, tanto al enviar información como al recibirla.

6.9.1. Web Services de la aplicación Web

La aplicación Web consta de dos Web Services utilizados por la aplicación móvil: el de Autenticación y el de Sincronización de datos.

El WS de Autenticación se encarga principalmente de validar las credenciales ingresadas por el usuario al iniciar sesión en la AppDUE. Dicha validación se realiza a través del método "`getToken`" del Web Service, el cual recibe como parámetros el nombre de usuario y la contraseña, y verifica que éstos sean correctos para acceder a la aplicación móvil. En caso de que las credenciales no sean válidas o no tengan permisos de acceso a la app, el WS devolverá un código y un mensaje de error, el cual le será notificado al usuario.

En la siguiente figura se puede ver parte del WSDL del Web Service de Autenticación.

```
▼<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="urn:WS-Authentication"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:WS-Authentication">
  ▼<types>
    ▼<xsd:schema targetNamespace="urn:WS-Authentication">
       <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
       <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
      ▼<xsd:complexType name="Resultado">
        ▼<xsd:all>
           <xsd:element name="idusuario" type="xsd:string"/>
           <xsd:element name="nombre" type="xsd:string"/>
           <xsd:element name="token" type="xsd:string"/>
           <xsd:element name="status" type="xsd:string"/>
         </xsd:all>
       </xsd:complexType>
      ▶<xsd:complexType name="ResultadoArray">...</xsd:complexType>
     </xsd:schema>
  </types>
 ▼<message name="getTokenRequest">
    <part name="username" type="xsd:string"/>
    <part name="password" type="xsd:string"/>
    <part name="app" type="xsd:string"/>
  </message>
 ▼<message name="getTokenResponse">
    <part name="return" type="tns:ResultadoArray"/>
  </message>
 ▶<message name="getAppVersionRequest">...</message>
 ▶<message name="getAppVersionResponse">...</message>
 ▶<portType name="WS-AuthenticationPortType">...</portType>
 ▶<binding name="WS-AuthenticationBinding" type="tns:WS-AuthenticationPortType">...</binding>
 ▶<service name="WS-Authentication">...</service>
</definitions>
```

Figura 6.18. WSDL del Web Service de Autenticación de la aplicación Web.

Por otra parte, el WS de Sincronización se encarga principalmente de la sincronización de los datos entre la aplicación móvil y la aplicación Web. Dentro de la información que recibe la app desde dicho WS se encuentran las solicitudes del usuario, las partidas de microchips habilitadas por el MAIBA y los tipos de pelajes, razas, capas y particularidades de un equino. Respecto a los datos que envía la app hacia el WS se encuentran los DUE sincronizados por el usuario.

En la siguiente figura se puede ver parte del WSDL del Web Service de Sincronización.

```
▼<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="urn:WS-Synchronization_2"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:WS-Synchronization_2">
  ▶<types>...</types>
  ▶<message name="getEstadosLegalesRequest">...</message>
  ▶<message name="getEstadosLegalesResponse">...</message>
  ▶<message name="getPelajesRequest">...</message>
  ▶<message name="getPelajesResponse">...</message>
  ▶<message name="getRazasRequest">...</message>
  ▶<message name="getRazasResponse">...</message>
  ▶<message name="getCapasRequest">...</message>
  ▶<message name="getCapasResponse">...</message>
  ▶<message name="getParticularidadesCabezaRequest">...</message>
  ▶<message name="getParticularidadesCabezaResponse">...</message>
  ▶<message name="getParticularidadesCuerpoRequest">...</message>
  ▶<message name="getParticularidadesCuerpoResponse">...</message>
  ▶<message name="getParticularidadesMiembroRequest">...</message>
  ▶<message name="getParticularidadesMiembroResponse">...</message>
  ▶<message name="getPartidasMicrochipRequest">...</message>
  ▶<message name="getPartidasMicrochipResponse">...</message>
  ▶<message name="getSolicitudesRequest">...</message>
  ▶<message name="getSolicitudesResponse">...</message>
  ▶<message name="saveDUEsRequest">...</message>
  ▶<message name="saveDUEsResponse">...</message>
  ▶<message name="checkAppVersionRequest">...</message>
  ▶<message name="checkAppVersionResponse">...</message>
  ▶<portType name="WS-Synchronization_2PortType">...</portType>
  ▶<binding name="WS-Synchronization_2Binding" type="tns:WS-Synchronization_2PortType">...</binding>
  ▶<service name="WS-Synchronization_2">...</service>
 </definitions>
```

Figura 6.19. WSDL del Web Service de Sincronización de la aplicación Web.

6.9.2. Casos de sincronización de datos

Los dos casos principales de sincronización de datos son: la sincronización de las solicitudes y la sincronización de los DUE generados, ambas peticiones originadas desde la aplicación móvil hacia la aplicación Web a través de sus Web Services.

Para ambos casos, la sincronización será solicitada de forma manual por el usuario y la aplicación móvil deberá contar con conexión a Internet para poder realizarlas.

Sincronización de solicitudes:

Origen: Aplicación móvil.

Destino: Aplicación Web.

Precondiciones:

- El usuario se encuentra logueado en la aplicación móvil.
- La aplicación móvil se encuentra con conexión a Internet.
- El usuario se encuentra en la pantalla de listado de solicitudes y realiza el gesto de actualizar contenido (mantiene presionado un punto en la parte superior de la pantalla, desliza hacia abajo y suelta).

Estrategia:

1. La aplicación móvil solicita el listado de solicitudes del usuario (veterinario habilitado) a la aplicación Web, a través del método "`getSolicitudes`" del WS, indicando como parámetros el "`idUsuario`" del usuario logueado y el "`token`" de validación, ambos almacenados al momento de iniciar sesión en la aplicación móvil.

   La aplicación móvil pausa su ejecución y queda esperando la respuesta del WS. Si en dos minutos no se recibe dicha respuesta, la operación se cancela automáticamente y se le notifica al usuario que se ha superado el tiempo de espera.
2. El WS recibe y procesa la petición y genera la respuesta con el resultado de la consulta.

3. La aplicación móvil recibe la respuesta con el nuevo listado de solicitudes y actualiza en su base de datos local y en la pantalla, la lista de solicitudes con dicha respuesta. Por cada solicitud recibida, la aplicación realiza las siguientes acciones (ver código en Figura 6.20):
    - Si la solicitud posee el "flag" de "borrado" en verdadero (1), entonces la aplicación móvil borra dicha solicitud de su lista local de solicitudes.
    - Por el contrario, si la solicitud posee el "flag" de "borrado" en falso (0), entonces:
        i. Si la solicitud recibida no existe en la lista local de solicitudes, entonces la agrega como una solicitud nueva a dicho listado.
        ii. En cambio, si la solicitud recibida ya existe en la lista local de solicitudes, se actualiza la cantidad restante de DUE de dicha solicitud, esto es, la cantidad restante total recibida menos la cantidad sin sincronizar de la solicitud. En caso de que dicha resta sea negativa, la cantidad restante se indicará en cero (0).
4. La aplicación móvil actualiza el listado de solicitudes que se ve en la pantalla y muestra un mensaje de sincronización exitosa.

Estructura de respuesta:

Ejemplo de estructura de respuesta del WS:

```
respuesta = {
  "borradas": [1, 2],
  "canceladas": [3],
  "inconsistentes": [4],
  "finalizadas": [5],
  "solicitudes": [
        {"idSolicitud": 6, "cantidadRestante": 3},
        {"idSolicitud": 7, "cantidadRestante": 2}
   ]
}
```

Postcondiciones:

El listado de solicitudes de la aplicación móvil se actualizó correctamente.

```
/**
 *
 * @param solicitudNueva
 */
sincronizarSolicitud(solicitudNueva) {

    // Si la solicitud fue borrada en la plataforma Web,
    // se borra dicha solicitud del listado local de solicitudes
    if (solicitudNueva.borrado == '1') {
        self.removeSolicitud(solicitudNueva.id)
    }
    // Sino, si la solicitud NO fue borrada
    else {

        // Trato de obtener la solicitud del listado local de solicitudes
        let solicitud = self.findSolicitud(solicitudNueva.id);

        // Si la solicitud existe localmente
        if (solicitud) {

            let cantRestanteTotal = parseInt(solicitudNueva.cantidadRestante);
            let cantDuesSinSincronizar = parseInt(solicitud.cantidadDuesSinSincronizar);

            // Calculo la nueva cantidad restante de DUE
            let cantidadRestante = cantRestanteTotal - cantDuesSinSincronizar;

            // Actualizo la cantidad restante de la solicitud
            solicitud.setCantidadRestante(cantidadRestante > 0 ? cantidadRestante : 0);
        }
        // Sino, si la solicitud NO existe localmente,
        // se agrega como una nueva solicitud
        else {
            self.addSolicitud(
                parseInt(solicitudNueva.id),
                solicitudNueva.propietarios,
                parseInt(solicitudNueva.cantidadRestante)
            );
        }
    }
},
```

Figura 6.20. Código de sincronización local por solicitud.

Sincronización de DUE:

Origen: Aplicación móvil.

Destino: Aplicación Web.

Precondiciones:

- El usuario se encuentra logueado en la aplicación móvil.
- La aplicación móvil se encuentra con conexión a Internet.

- El usuario posee DUE generados en la aplicación móvil en estado “Sin sincronizar”.

Estrategia:

1. El usuario indica de forma manual que quiere sincronizar un DUE específico.
2. La aplicación móvil valida que dicho DUE es sincronizable, es decir, que sus datos mínimos se encuentran completados. Si el DUE no es sincronizable, el sistema le notificará al usuario que no es posible realizar la sincronización.
3. En caso de que el DUE sea sincronizable, la aplicación móvil envía el DUE hacia la aplicación Web, a través del método “`saveDUEs`” del WS, indicando como parámetros el “`idUsuario`” del usuario logueado, el “`token`” de validación y el DUE a sincronizar.

   La aplicación móvil pausa su ejecución y queda esperando la respuesta del WS. Si en dos minutos no se recibe dicha respuesta, la operación se cancela automáticamente y se le notifica al usuario que se ha superado el tiempo máximo de espera.
4. El WS recibe la petición y valida el estado de la solicitud correspondiente al DUE recibido:
5. Si la solicitud fue borrada en la base de datos de la aplicación Web, entonces el sistema desestimará la información recibida y retornará un mensaje de error, indicando que la solicitud ya no es válida. El flag “`solicitudInexistente`” será indicado como verdadero (1) en la respuesta del WS.
6. Si el estado de la solicitud es “Pendiente de pago”, el sistema desestimará la información recibida y retornará un mensaje de error, indicando que la solicitud no se encuentra paga. El flag “`solicitudImpaga`” será indicado como verdadero (1) en la respuesta del WS.

a. En cambio, si el estado de la solicitud es “Pagada”, entonces el sistema validará la información recibida del DUE:
    i. Si el DUE enviado no posee un número de microchip especificado, la aplicación Web lo almacenará en la base de datos en estado “Borrador”. Dicho DUE será marcado como “generado” en la respuesta del WS.
    ii. En cambio, si el DUE enviado sí posee un número de microchip especificado, el sistema validará que dicho número sea correcto:
        1. Si el número de microchip se encuentra utilizado por otro DUE en estado “Generado”, el DUE recibido no será almacenado en la base de datos y será marcado como “existente” en la respuesta del WS.
        2. Si el número de microchip se encuentra utilizado por otro DUE en estado “Borrador”, este último es actualizado con la información especificada en el DUE recibido, el cual será marcado como “actualizado” en la respuesta del WS.
        3. Si el número de microchip no está siendo utilizado por otros DUE, el sistema almacenará el DUE recibido en su base de datos con el estado “Generado”, y lo marcará como “generado” en la respuesta del WS.
    iii. Tanto al actualizar como al almacenar un DUE en la base de datos de la aplicación Web, el sistema validará que la cantidad resultante de DUE pendientes de generación de la solicitud sea consistente:

1. Si la cantidad de DUE generados resultante es mayor a la máxima permitida por la solicitud, ésta será indicada con el estado "inconsistente" en la base de datos de la aplicación Web, al igual que todos sus DUE asociados. En dicho caso, el usuario deberá corregir manualmente la situación a través de la aplicación Web.
2. En cambio, si la cantidad de DUE generados resultante no es mayor a la máxima permitida por la solicitud, el sistema asumirá que dicha cantidad es consistente y correcta.

7. El WS genera la respuesta con el resultado de la sincronización.
8. La aplicación móvil recibe dicha respuesta y la procesa:
    a. La app verifica que el resultado de la sincronización haya sido correcto. Esto lo hace evaluando el valor de la variable "`status`", recibido en la respuesta del WS. Si dicho valor indica que no hubo error, la app continuará con su ejecución. En cambio, si el código indica que ha ocurrido un error, el mismo le será notificado al usuario.
    b. En el caso de que no haya habido error, si el flag "`solicitudInexistente`" de la respuesta del WS se encuentra en verdadero (1), entonces la aplicación móvil le notifica al usuario que la solicitud asociada al DUE a sincronizar ya no es válida, por lo que la sincronización no se ha podido realizar.
    c. Sino, si el flag "`solicitudImpaga`" de la respuesta del WS se encuentra en verdadero (1), entonces la aplicación móvil le notifica al usuario que

la solicitud asociada al DUE a sincronizar no se encuentra paga, por lo que la sincronización no se ha podido realizar.

d. Por el contrario, si el valor de los flags "`solicitudInexistente`" y "`solicitudImpaga`" es falso (0), entonces la aplicación móvil verifica si el DUE a sincronizar se encuentra dentro de los arreglos "`generados`" o "`actualizados`" de la respuesta del WS. En dichos casos:
    i. Se actualiza la "`cantidadRestante`" de la solicitud asociada al DUE, con la cantidad recibida en la respuesta del WS.
    ii. El DUE es marcado con el estado "Sincronizado".
    iii. El sistema le notifica al usuario que el DUE se ha sincronizado correctamente.
e. Si en la respuesta del WS, el arreglo "`existentes`" contiene información (no se encuentra vacío), entonces por cada uno de los DUE incluidos en dicho arreglo, la aplicación le notifica al usuario el motivo por el cual no fue sincronizado. Por ejemplo, porque el número de microchip del DUE ya se encontraba registrado previamente por otro distinto.

Estructura de respuesta:

Ejemplo de estructura de respuesta del WS:

```
respuesta = {
  "id": 54, // Id de la solicitud asociada.
  "status": 20, // Cód. del estado resultante de la sincronización.
  "solicitudInexistente": 0, // Indica si la solicitud fue o no borrada.
  "solicitudImpaga": 0, // Indica si la solicitud se encuentra o no paga.
  "solicitudQuedoInconsistente": 0, // Flag de solicitud inconsistente.
  "cantidadTotal": 10, // Cantidad total de la solicitud.
  "cantidadRestante": 3, // Cantidad restante de la solicitud.
  "generados": [1], // Ids de los DUEs marcados como "generado".
  "actualizados": [], // Ids de los DUEs marcados como "actualizado".
  "existentes": [], // Ids de los DUEs marcados como "existente".
}
```

Postcondiciones:

Se sincronizaron correctamente los DUE a través del WS de la aplicación Web, y se actualizaron las solicitudes y el DUE dentro de la aplicación móvil.

NOTA: Como se puede observar, tanto el código de la aplicación móvil como el del WS de la aplicación Web están preparados para enviar y recibir varios DUE a la vez, siempre y cuando correspondan a una misma solicitud. Esto es así ya que en su comienzo el cliente había solicitado poder sincronizar una solicitud completa, es decir, enviar todos sus DUE en una única vez, pero luego dicho requerimiento fue modificado por el cliente mismo, solicitando realizar las sincronizaciones solamente a nivel de DUE.

6.10. Librerías utilizadas

A continuación, se listan las librerías utilizadas en el proyecto junto a sus versiones:

```
"@patwoz/react-navigation-is-focused-hoc": "^1.2.1",
"easy-bluetooth-classic": "^1.0.4",
"formik": "^0.9.4",
"mobx": "^3.3.1",
"mobx-react": "^4.3.3",
"mobx-state-tree": "^1.0.1",
"native-base": "^2.3.2",
"prop-types": "^15.6.0",
"react": "^16.0.0",
"react-native": "^0.47.1",
"react-native-action-button": "^2.8.0",
"react-native-animatable": "^1.2.4",
"react-native-background-timer": "^2.0.0",
"react-native-bluetooth-serial": "^1.0.0-rc1",
"react-native-datepicker": "^1.6.0",
"react-native-device-info": "^0.12.1",
"react-native-elements": "^0.17.0",
"react-native-fs": "^2.8.5",
"react-native-image-crop-picker": "^0.20.1",
"react-native-image-picker": "^0.26.7",
"react-native-image-resizer": "^1.0.0",
"react-native-image-zoom-viewer": "^2.1.2",
"react-native-modalbox": "^1.4.2",
```

```
  "react-native-orientation": "^3.1.0",
  "react-native-smart-splash-screen": "^2.3.5",
  "react-native-svg": "^5.4.1",
  "react-native-vector-icons": "^4.4.2",
  "react-navigation": "^1.0.0-beta.13",
  "soap-everywhere": "^1.0.7",
  "yup": "^0.22.0"
```

6.11. Permisos de la aplicación

Tratándose de una aplicación desarrollada para Android, los permisos específicos requeridos se detallan a continuación [40]:

ACCESS_NETWORK_STATE

- Permite que las aplicaciones accedan a información sobre redes.
- Nivel de protección: normal.
- Valor constante: "android.permission.ACCESS_NETWORK_STATE".

BLUETOOTH

- Permite que las aplicaciones se conecten a dispositivos Bluetooth emparejados.
- Nivel de protección: normal.
- Valor constante: "android.permission.BLUETOOTH".

BLUETOOTH_ADMIN

- Permite que las aplicaciones descubran y emparejen dispositivos Bluetooth.
- Nivel de protección: normal.
- Valor constante: "android.permission.BLUETOOTH_ADMIN".

CAMERA

- Necesario para poder acceder al dispositivo de la cámara.
- Nivel de protección: peligroso.
- Valor constante: "android.permission.CAMERA".

INTERNET

- Permite que las aplicaciones abran conexiones de red.
- Nivel de protección: normal.
- Valor constante: "android.permission.INTERNET".

WRITE_EXTERNAL_STORAGE

- Permite que una aplicación escriba en almacenamiento externo.
- Nivel de protección: peligroso.
- Valor constante: "android.permission.WRITE_EXTERNAL_STORAGE".

6.12. Publicación en Google Play

Una vez finalizado el desarrollo de la aplicación, el siguiente paso fue publicarla en la tienda de Google Play.

6.12.1. Generación de clave de carga

Dado que Android requiere que todos los APK estén firmados de manera digital con un certificado para poder instalarse o actualizarse en un dispositivo, primero fue necesario firmar la app con una clave de carga, generada manualmente con la herramienta `keytool` de Java. Para hacer esto, desde el Símbolo del sistema de Windows, se debió navegar a la carpeta "C:\Program Files\Java\jdkx.x.x_x\bin." y ejecutar el siguiente comando:

```
 $ keytool -genkeypair -v -keystore due.keystore -alias due -keyalg RSA -keysize
2048 -validity 10000
```

Como resultado, se generó el archivo `due.keystore`, válido por 10.000 días.

6.12.2. Configuración de variables Gradle

Una vez generado el archivo `due.keystore`, se colocó el mismo dentro del directorio "android/app" del proyecto.

Luego, se editó el archivo "android/gradle.properties" y se agregaron las siguientes variables globales, que luego serán utilizadas en la configuración de Gradle para la firma de la aplicación (se reemplazaron las contraseñas reales por asteriscos).

```
MYAPP_RELEASE_STORE_FILE=due.keystore
MYAPP_RELEASE_KEY_ALIAS=due
MYAPP_RELEASE_STORE_PASSWORD=*****
MYAPP_RELEASE_KEY_PASSWORD=*****
```

Por último, se debió configurar las versiones de lanzamiento para que se firmen con la clave de carga. Para esto, se editó el archivo "android/app/build.gradle" del proyecto y se agregó la configuración de firma:

```
...
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) {
                storeFile file(MYAPP_UPLOAD_STORE_FILE)
                storePassword MYAPP_UPLOAD_STORE_PASSWORD
                keyAlias MYAPP_UPLOAD_KEY_ALIAS
                keyPassword MYAPP_UPLOAD_KEY_PASSWORD
            }
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
...
```

6.12.3. Generación de APK

Finalizada la configuración, se prosiguió con la generación del APK. Para esto, se ejecutó desde el Símbolo del sistema de Windows el siguiente código, dentro de la carpeta raíz del proyecto:

```
$ cd android
$ gradlew bundleRelease
```

Como resultado, se generó el archivo APK dentro del directorio "/android/app/build/outputs/apk/app-release.apk", el cual será el subido a la tienda de Google Play.

6.12.4. Publicación en Play Console

Luego de compilar y firmar la versión de lanzamiento de la app, el siguiente paso fue subirla a la tienda de Google Play para inspeccionarla, probarla y publicarla. Paso esto, se siguió la guía oficial para la preparación y lanzamiento de versiones en Google Play [46].

Una vez creada y publicada la versión productiva de la app, la misma puede ser instalada por todos los usuarios del país. Es posible acceder a ella a través del siguiente link: https://play.google.com/store/apps/details?id=com.due

Figura 6.21. Ficha de la app en Play Store.

6.13. Limitaciones en iOS

Durante el proceso de análisis de requerimientos surgió que gran parte de las lectoras de radiofrecuencia del mercado utilizadas por los veterinarios para leer los microchips, no hacen uso de la tecnología BLE (Bluetooth Low Energy) para emparejarse y comunicarse con otros dispositivos, esto es porque utilizan una versión de Bluetooth anterior a la 4.0.

Por otra parte, Apple presenta serias restricciones para vincular sus productos vía Bluetooth con dispositivos externos que, entre otras condiciones, no formen parte de los perfiles de Bluetooth estándar soportados por iOS o que no utilicen únicamente la tecnología BLE para comunicarse, a menos que los mismos se encuentren certificados por la compañía, es decir, que formen parte del Programa MFi (Made For iPhone) de Apple [41] [42] [43].

Por tales motivos y dado que la implementación de la aplicación móvil en iOS no era un requisito, se decidió postergar dicha compilación hasta que la mayoría de las lectoras de radiofrecuencia del mercado hagan uso de la tecnología BLE o se encuentren certificadas por Apple a través del Programa MFi.

7

CONCLUSIONES Y FUTURAS MEJORAS

En este capítulo se presentan las conclusiones obtenidas a partir del trabajo realizado y se mencionan algunas de las posibles mejoras que se pueden realizar a futuro sobre la aplicación móvil.

7.1. Conclusiones

Haber formado parte del equipo de desarrollo del Sistema de Gestión del Documento Único Equino del MAIBA me ha resultado enriquecedor en múltiples aspectos. Desde el punto de vista técnico, me ha permitido conocer y utilizar por primera vez una herramienta para el desarrollo de aplicaciones móviles multiplataforma, la cual desconocía.

Como se ha mencionado a lo largo de la presente tesina, dicha herramienta utilizada fue React Native, cuya experiencia de uso fue realmente positiva. Contando con conocimientos en JavaScript, la curva de aprendizaje fue rápida, aunque fue necesario familiarizarse con la sintaxis JSX, con el concepto de promesas y con la arquitectura de flujo de datos unidireccional propuesta por Flux. Además, se destacan los tiempos de compilación casi inmediatos y la recarga en vivo, que permite ejecutar nuevo código sin la necesidad de perder el estado en el que se encuentra la aplicación, agilizando considerablemente las tareas de prueba sobre la misma. En cuanto al rendimiento y a la experiencia de usuario al utilizar la app, no se notan

diferencias con una aplicación nativa. Tanto el tiempo de respuesta como la fluidez al navegar entre las distintas pantallas es realmente buena.

A nivel personal, he podido capitalizar el desafío de formar parte de un equipo multidisciplinario, trabajando en el desarrollo de una aplicación móvil con una herramienta que no conocía y en un tiempo acotado. La pericia que esto ha sumado a mis diez años de experiencia laboral, ha motivado la elección de este proyecto para el desarrollo de la presente tesina.

El hecho de haber formado parte de la implementación de la aplicación móvil, tanto de las capacitaciones brindadas al usuario final sobre su uso, como durante sus primeras interacciones con la misma, me permitió no sólo conocer cómo funciona la aplicación en una práctica real, sino también tener un feedback directo por parte de los usuarios. Esto es un valor agregado que quiero destacar, el cual me permitió recordar que gran parte de nuestro trabajo tiene por objeto la facilitación de tareas a personas reales, mejorando la calidad de su trabajo o su vida.

7.2. Trabajo futuro

La principal mejora pendiente es el desarrollo, compilación y publicación de la aplicación móvil para iOS. Para realizar esto, antes será necesario modificar y ajustar parte del código, ya que algunos de los componentes utilizados solo son soportados por Android. Como se ha mencionado en el capítulo anterior, dicho desarrollo estará justificado cuando la mayoría de las lectoras de radiofrecuencia del mercado hagan uso de la tecnología BLE o se encuentren certificadas por Apple a través del Programa MFi.

Por otro lado, existen algunas funcionalidades que, si bien no fueron solicitadas por el cliente, su desarrollo mejoraría la experiencia de aquellos que utilicen la aplicación. Por ejemplo, se podría incluir una nueva funcionalidad que permita, a partir de la lectura de un

microchip, consultar todos los datos del DUE asociado, tanto las características completas del equino como la información de sus propietarios.

También se podría incluir un módulo de denuncias, que le permita a los usuarios realizar distintos tipos de denuncias a través de la aplicación. Por ejemplo, podrían denunciar el robo de un equino.

Otro de los aspectos a mejorar es la interacción del usuario con la app, particularmente en la solapa de “Diagrama”, donde el usuario puede indicar las características del equino dentro de las distintas imágenes SVG que lo representan. Actualmente, dicha interacción está muy limitada, ya que el usuario no puede manipular el zoom de la imagen ni mover dinámicamente la característica una vez ingresada. Si se quisiera corregir la posición de una característica, el usuario deberá eliminarla y agregarla nuevamente.

REFERENCIAS

[1] Ley Nº 13.627 - Documento Único Equino (DUE) http://www.maa.gba.gov.ar/sites/default/files/DUE_ley_13627.pdf

[2] Fundamentos de la Ley Nº 13.627 http://www.gob.gba.gov.ar/legislacion/legislacion/f-13627.html

[3] Ley Nº 10.891- Guía Única de Traslado de Ganado http://www.maa.gba.gov.ar/sites/default/files/GUT_LEY_10891.pdf

[4] Decreto - Ley Nº 10.081/1983 https://www.mseg.gba.gov.ar/directorios/marco_normativo/codigos%20provinciales/dec_ley10.081-83_codigo_rural.pdf

[5] Decreto 1.734/11 http://www.gob.gba.gov.ar/legislacion/legislacion/11-1734.html

[6] Resolución N° 248- MAGP-18 https://www.gba.gob.ar/sites/default/files/agroindustria/docs/resolucion_due.pdf

[7] Arhippainen, L., Tähti, M. (2003). Empirical Evaluation of User Experience in Two Adaptative Mobile Application Prototypes. Proceedings of the 2nd International Conference on Mobile and Ubiquitous Multimedia, 10–12 diciembre, 2003, Norrköping, Sweden. Recuperado de http://www.ep.liu.se/ecp/011/007/ecp011007.pdf

[8] Knapp Bjerén, A. (2003). La Experiencia del Usuario. En: Knapp Bjerén, A. (coord.). La Experiencia del Usuario. Madrid: Anaya Multimedia, 2003, ISBN 84-415-1044-X.

[9] DNX (2005). Usabilidad y Experiencia de Usuario. Microsoft España: Guía Práctica de Usabilidad Web.

[10] Morville, P. (2004). User Experience Design. SemanticStudios, 21 de junio de 2004. Recuperado de http://semanticstudios.com/user_experience_design/

[11] Mahlke, S. (2002). Factors influencing the experience of website usage. Conference on Human Factors in Computing Systems - Proceedings. 846-847. 10.1145/506443.506628.

[12] Mantis Bug Tracker. [En línea]. https://www.mantisbt.org/

[13] Nielsen, Jakob. (1994). Ten Usability Heuristics. Recuperado de https://www.nngroup.com/articles/ten-usability-heuristics/

[14] Stelios Xinogalos Spyros Xanthopoulos, "A Comparative Analysis of Crossplatform Development Approaches for Mobile Applications," in Proceedings of the 6th Balkan Conference in Informatics, Thessaloniki, Greece, 2013, pp. 213-220. Recuperado de

https://www.academia.edu/23709239/A_comparative_analysis_of_cross-platform_development_approaches_for_mobile_applications

[15] PHP. [En línea]. https://www.php.net/

[16] Symfony. [En línea]. https://symfony.com/

[17] MariaDB. [En línea]. https://mariadb.org/

[18] Apache Subversion. [En línea]. https://subversion.apache.org/

[19] Apache. [En línea]. https://www.apache.org/

[20] React Native. [En línea]. http://www.reactnative.com/

[21] React Native. [En línea]. https://facebook.github.io/react-native/

[22] React. [En línea]. https://es.reactjs.org/

[23] Flux. [En línea]. https://facebook.github.io/flux/

[24] MobX. [En línea]. https://mobx.js.org/

[25] Android - Interfaz de usuario: https://developer.android.com/guide/topics/ui

[26] Apple - Human Interface Guidelines: https://developer.apple.com/design/human-interface-guidelines/

[27] Apple - UI Design Do's and Don'ts: https://developer.apple.com/design/tips/

[28] Android Dashboard. [En línea]. https://developer.android.com/about/dashboards/index.html

[29] Node.js. [Download]. https://nodejs.org/en/download/

[30] Java SE Development Kit 8. [Download]. https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[31] Python 2. [Download]. https://www.python.org/downloads/windows/

[32] Android Studio. [Download]. https://developer.android.com/studio/index.html

[33] Requerimientos de React Native. https://github.com/facebook/react-native#-requirements

[34] Lagares, E. (2018). ¿Qué es React?, 18 de junio de 2018. Recuperado de https://mvpcluster.com/react/

[35] Azaustre, C. (2018). ¿Qué es Flux? Entendiendo su arquitectura, 25 de mayo de 2018. Recuperado de https://carlosazaustre.es/como-funciona-flux/

[36] Cumplimiento de los requisitos de nivel objetivo de la API de Google Play. [En línea]. https://developer.android.com/distribute/best-practices/develop/target-sdk?hl=ES

[37] React Native - Release 0.49.0. [En línea]. https://github.com/facebook/react-native/releases/tag/v0.49.0

[38] Babel. [En línea]. https://babeljs.io/

[39] React Native - Increase Async Storage size. [En línea]. https://github.com/react-native-community/async-storage/blob/LEGACY/docs/advanced/IncreaseDbSize.md

[40] Android - Manifest.permission. [En línea]. ttps://developer.android.com/reference/android/Manifest.permission

[41] Apple - Perfiles de Bluetooth compatibles con iOS. [En línea]. https://support.apple.com/es-lamr/HT204387

[42] Apple - MFi Program. [En línea]. https://developer.apple.com/programs/mfi/

[43] Apple - MFi Program. Frequently Asked Questions. [En línea]. https://mfi.apple.com/MFiWeb/getFAQ.action

[44] React Native - Publicación en Google Play Store. [En línea]. https://facebook.github.io/react-native/docs/0.47/signed-apk-android

[45] Android Studio - Sube tu app a Play Console. [En línea]. https://developer.android.com/studio/publish/upload-bundle?hl=es

[46] Play Console - Preparar y lanzar versiones. [En línea]. https://support.google.com/googleplay/android-developer/answer/7159011?hl=es

[47] JetBrains WebStorm. [En línea]. https://www.jetbrains.com/webstorm/