UNMDP - Facultad de Ingeniería
Departamento de Informática

Sistema de medición y monitoreo de pasturas:

Pastech

Subsistema aplicación móvil

Alumnos

Veitch, Matías Nicolás <matiasveitch@gmail.com>
Escudé, Nicolás Fernando <nicolasescude.soft@gmail.com>

Directora

Kuzman, Melisa

Co-Director

Hinojal, Hernán

Proyecto final para optar al grado de Ingeniero/a en Informática
Mar del Plata, 15 de julio de 2024

UNMDP - Facultad de Ingeniería
Departamento de Informática

Sistema de medición y monitoreo de pasturas:

Pastech

Subsistema aplicación móvil

Alumnos

Veitch, Matías Nicolás <matiasveitch@gmail.com>
Escudé, Nicolás Fernando <nicolasescude.soft@gmail.com>

Directora

Kuzman, Melisa

Co-Director

Hinojal, Hernán

Proyecto final para optar al grado de Ingeniero/a en Informática
Mar del Plata, 15 de julio de 2024

Agradecimientos

Queremos expresar nuestra sincera gratitud a quienes han sido parte fundamental de nuestro trayecto:

A nuestros padres, por su amor, apoyo y paciencia que nos acompañaron en cada paso de esta travesía.

A nuestros compañeros a lo largo de la carrera, cuya colaboración y amistad dejaron marcas en nuestras personas.

A los docentes de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata, quienes a lo largo de nuestra formación nos dieron las herramientas para poder resolver este proyecto.

A nuestra directora Melisa Kuzman, al codirector Hernán Hinojal y a Fernando Genin por su guía a lo largo del proyecto.

A los Ingenieros agrónomos, referentes funcionales y fundadores del proyecto Pastech, Juan Insúa y Alejandra Marino, por brindarnos la oportunidad y darnos esta experiencia.

Índice

Resumen

El presente informe detalla el desarrollo de la aplicación móvil realizada por los estudiantes Matías Veitch y Nicolás Escudé. Dicha aplicación nace de un proyecto mayor llamado Pastech, cuyo objetivo es desarrollar una solución completa destinada a optimizar la eficiencia en el manejo de pasturas. Por otra parte, el objetivo particular de la aplicación en la que se centra este informe es el de resolver los desafíos de la carga y gestión de datos provenientes de dispositivos de IoT utilizados para medir pastizales en contextos rurales.

Este proyecto se ejecutó simultáneamente con el equipo de estudiantes integrado por Tobías Demoor y Leopoldo Lening Celaya, quienes se encargaron del desarrollo del denominado sistema *cloud*. Dicho sistema está integrado por una aplicación web que ofrece una interfaz de usuario para la gestión de los datos y un conjunto de servicios en la nube utilizados tanto por la aplicación web como por la móvil. Por esto último, existió una crucial colaboración entre ambos equipos, lo cual fue una experiencia sumamente enriquecedora a lo largo de todas las etapas del proceso.

También se trabajó en estrecha colaboración con el Departamento de Ingeniería en Electrónica y Computación de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata, quienes se encargaron de desarrollar el dispositivo de medición del cual la aplicación obtiene sus datos.

El informe abarca la descripción detallada del análisis, diseño e implementación llevados a cabo por los estudiantes, en constante comunicación con los referentes funcionales. Además, se presentan las estimaciones de tiempos planificadas previamente y se realiza una comparación con la duración real del proyecto.

Introducción

La producción ganadera desempeña un papel fundamental en la economía argentina, representando más del 40% del Producto Bruto Agropecuario. Uno de los factores determinantes en el rendimiento de los sistemas ganaderos de base pastoril es la eficiente gestión del pastoreo, donde la cantidad y calidad del pasto disponible juegan un papel crucial, siendo esta la materia prima a partir de la cual se genera el producto animal. A pesar de su importancia, la medición precisa de la disponibilidad forrajera en los potreros ha sido históricamente un desafío para imponer como práctica común, esto se debe a que las técnicas tradicionales son laboriosas y demandantes en tiempo.

Para abordar esta problemática, el proyecto Pastech surge como una iniciativa innovadora desarrollada por investigadores de la Facultad de Ciencias Agrarias, los ingenieros Juan Insúa y Alejandra Marino. Su objetivo principal es diseñar un sistema integral de medición y monitoreo de pasturas, basado en la utilización de tecnología para permitir estimar de manera objetiva, rápida y precisa la variación espacio-temporal del pasto en cada establecimiento ganadero.

Con esto se busca propiciar la gestión fundamentada en información confiable sobre las pasturas. Que tiene como impacto el aumento de la eficiencia de las producciones ganaderas, como así también la disminución de su impacto ambiental al favorecer el uso óptimo de los recursos.

Objetivos del proyecto

Objetivos Pastech

El propósito de Pastech es desarrollar una solución completa destinada a optimizar la eficiencia en el manejo de pasturas. Esta solución se enfoca en la medición y monitoreo de pastizales, lo que facilita la toma de decisiones informadas para una gestión eficiente y consistente en el tiempo.

Las metodologías tradicionales para la toma de datos son costosas, estas implican la medición manual y la transcripción a planillas para finalmente cargarse en un sistema que procese los datos, normalmente una planilla de cálculo. Por su dificultad y costo es normal que los productores no lleven a cabo este proceso y a la hora de decidir cuántos suplementos utilizar o cómo rotar el ganado no tengan suficientes datos para tomar una decisión óptima.

Al desarrollar una solución que simplifique la medición y monitoreo de pastizales, se aumenta el rendimiento de la ganadería. Como así también, disminuye el impacto ecológico por cabeza de ganado al necesitar menos tierras y hacer un uso más eficiente de los suplementos.

División en subproyectos

Dada la magnitud del proyecto, se tomó la decisión de dividirlo en dos subproyectos. Por un lado, el sistema *cloud* se encargará de recibir las mediciones a través de internet, procesarlas y mostrar sus estadísticas. Por otro lado, se encuentra la aplicación móvil, que es el foco principal de este informe.

Este enfoque permite abordar cada aspecto del sistema con mayor especialización y, al mismo tiempo, facilita un desarrollo en paralelo, lo cual es crucial para acelerar la introducción del producto al mercado.

Para lograr esto, fue necesario definir de manera conjunta las interfaces que utilizarían los sistemas para comunicarse, una vez que se tuvieran definidos los flujos de datos y los requerimientos completos de cada subsistema.

Objetivos de la aplicación móvil

La aplicación móvil deberá facilitar la tarea de medición de pasturas y la carga de los datos obtenidos, evitando así el traspaso manual de información entre sistemas. Para esto, debe ser capaz de recibir mediciones del dispositivo pasturómetro, que utiliza Bluetooth para transmitir datos sobre la altura del pasto. Estas mediciones deben ser asociadas de manera transparente con su ubicación geográfica y posteriormente deben ser enviadas al sistema *cloud* donde se monitorea la información. Al utilizarse en un entorno primordialmente rural, deberá poder prescindir de conexión a internet durante largos intervalos de tiempo.

Asimismo, se tendrá la funcionalidad de realizar mediciones para calibración, las cuales difieren a las previamente mencionadas, ya que están vinculadas a una calibración y tienen un código identificador. El código permite más tarde asociarlas con un peso de pasto seco para así calcular curvas de calibración que permitan la transformación de altura de pasto a kilogramo de pasto seco por hectárea.

Todo esto debe ser diseñado para que sea intuitivo y adaptable a distintos dispositivos móviles con sus diferencias de *hardware* y *software*. Haciendo frente a una amplia variedad de capacidades de procesamiento y resoluciones de pantalla para mostrar información.
A su vez, la interfaz de usuario debe cumplir con los estándares modernos de diseño gráfico para así estar a la par o por encima de las soluciones actuales y futuras cercanas.

Alcance

Se desarrollará una aplicación móvil que provea al usuario de las siguientes funcionalidades:

- Gestión de la conexión a distintos dispositivos compatibles.
- Recepción de mediciones del pasturómetro.
- Manejo de fallos en mediciones. Notificación de usuario en caso de fallo.
- Envío de mediciones al sistema *cloud*.
- Visualización de últimas mediciones.
- Creación, visualización, edición, eliminación y envío de calibraciones al sistema *cloud*.
- Visualización geográfica de mediciones y potreros en mapas capaces de funcionar sin conexión a internet.
- Manejo de sesión del sistema *cloud.*
- Sincronización de potreros y calibraciones con el sistema *cloud*.

Las pruebas se realizarán solamente sobre Android, ya que es la plataforma con la que el equipo de desarrollo cuenta disponibilidad.
El proyecto no abarca la gestión de publicación a tiendas digitales ni testeos de la aplicación en iOS.

Entregables

- Requerimientos funcionales y no funcionales a cumplir.
- Casos de uso.
- Entrega del código fuente multiplataforma de la aplicación.
- Aplicación funcional compilada para Android.

Proyecto interdisciplinario

Durante el desarrollo de la aplicación, se colaboró estrechamente con el Departamento de Electrónica de la Universidad Nacional de Mar del Plata. Una parte esencial del proyecto consistió en la definición de protocolos para la comunicación entre la aplicación y el dispositivo pasturómetro, el cual contaba con su propia electrónica diseñada por dicho departamento.

El departamento proporcionó un dispositivo que simulaba el comportamiento del pasturómetro para fines de desarrollo y pruebas. En este dispositivo se implementó la comunicación mediante Bluetooth Low Energy (BLE), una elección estándar para aplicaciones de IoT. La decisión de utilizar BLE se basó en su idoneidad como solución estándar para dispositivos IoT, ofreciendo una combinación óptima de bajo consumo de energía y alcance adecuado para las necesidades de comunicación con la aplicación.

Durante todo el proyecto, se mantuvo una constante retroalimentación con el equipo del Departamento de Electrónica. Este intercambio permitió definir qué información se enviaría, cómo se llevaría a cabo el proceso de envío y cómo deberían ser tratados los datos.

Otro aspecto fundamental de esta colaboración interdisciplinaria fue el trato con los referentes funcionales, los ingenieros agrónomos Juan Insúa y Alejandra Marino. Su experiencia en el sector agrícola fue indispensable para transmitir su entendimiento del problema. En conjunto, se planificaron las maneras de abordar las problemáticas detectadas.

Planificación original del proyecto

Antes de dar inicio al proyecto, se realizó un relevamiento y análisis preliminar con el objetivo de elaborar un protocolo que evaluará la viabilidad del proyecto como trabajo final. En esta fase inicial, era necesario proporcionar una estimación temporal del proyecto. Esta estimación se basó en los resultados obtenidos durante el relevamiento, junto con el análisis realizado por el equipo para desglosar el proyecto en diversas etapas. Este proceso permitió establecer el siguiente cronograma:

Cronograma inicial planificado

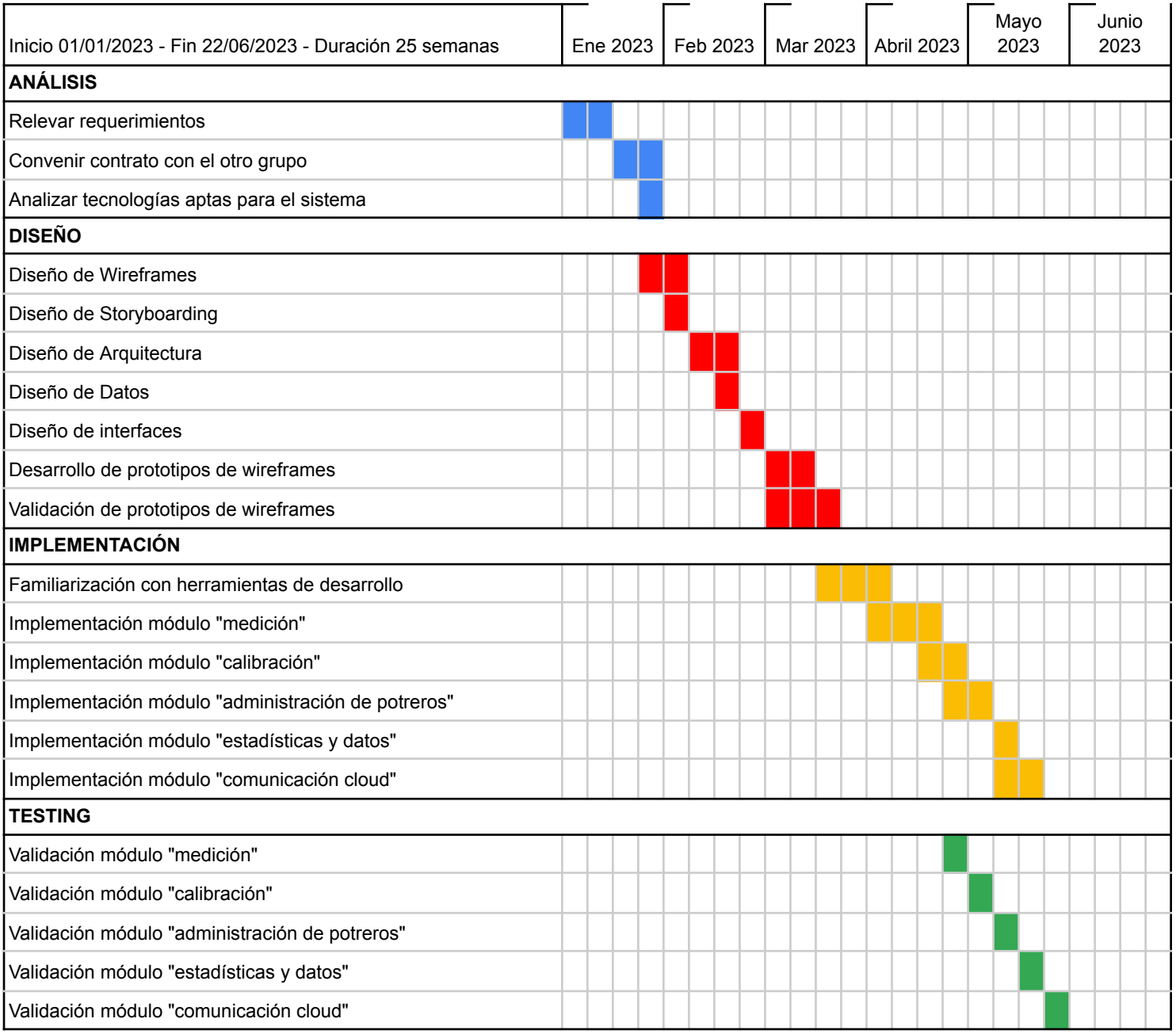


Figura 1 - Diagrama Gantt

Análisis FODA

La implementación temprana de un FODA exige identificar aquellas Fortalezas, Oportunidades, Debilidades y Amenazas que son inherentes al proyecto en su totalidad. Llevarlo a cabo permite dar una idea rápida de los factores internos y externos que pueden llevar al éxito o al fracaso. Es una herramienta simple, pero relevante durante todo el proceso.

Fortalezas

- **Visión de expertos en el proceso**: Se cuenta con la perspectiva de referentes funcionales y expertos en el proceso tradicional de pesaje de pasturas, lo que resulta esencial para validar el producto desde su concepción.
- **Formación académica**: Los integrantes del equipo cuentan con una formación en ingeniería informática. Esta base de conocimientos teóricos y prácticos permiten aprender más rápidamente nuevas herramientas y tecnologías al contar con sólidos fundamentos en los cuales apoyarse.
- **Apoyo académico**: El proyecto se beneficia de la supervisión, opiniones y consejos de profesionales del ámbito académico, disponibles dentro del marco del proyecto final.

Oportunidades

- **Desarrollo paralelo**: El trabajo en conjunto con otro grupo puede ayudar a solventar dudas comunes a ambos.

Debilidades

- **Falta de experiencia en desarrollo móvil**: No se tiene experiencia previa en la creación de aplicaciones móviles, lo que podría ralentizar el proceso de desarrollo.
- **Limitado conocimiento del sector agropecuario**: El equipo no posee experiencia directa en actividades agropecuarias, lo que puede afectar la comprensión inicial del problema.
- **Equipo nuevo**: El equipo de desarrollo no ha trabajado previamente en conjunto.
- **Separación geográfica del equipo**: La dispersión geográfica del grupo de desarrollo puede dificultar la colaboración y coordinación en tiempo real.
- **Interacción con el dispositivo pasturómetro**: La interacción entre los dos sistemas, ambos de ellos en etapas iniciales, introduce un punto adicional de posible fallo y aumenta la complejidad del sistema.

Amenazas

- **Cambios imprevistos en el desarrollo**: Mientras el proyecto está en marcha, el cliente puede cambiar, agregar o remover requerimientos que deberán ser eficazmente integrados al flujo de trabajo.
- **Intercambio entre grupos**: El desarrollo del proyecto en grupos paralelos puede conllevar ajustes repentinos en las necesidades o prioridades del producto, afectando su progreso.

Metodología de análisis

El proceso de elicitación de requerimientos se llevó a cabo inicialmente mediante reuniones presenciales con los referentes funcionales. Establecer una comunicación sólida en esta etapa fue fundamental para construir las bases del sistema correctamente. Durante las sesiones se introdujeron los desafíos y problemáticas a enfrentar, como así también la expectativa de los referentes hacia el proyecto. Se registraron los requisitos surgidos de la exposición y el diálogo, a su vez se exploraron posibles implementaciones considerando su viabilidad.

En estos intercambios fue sumamente importante la tarea de orientar al cliente, ayudándolo a definir soluciones factibles y eficientes para alcanzar los objetivos planteados. Así, este proceso se llevó a cabo de manera iterativa, ambas partes colaboraron en el modelado del sistema, que fue refinado a medida que se profundizaba en la comprensión del dominio del problema.

Sumado a los datos relevados en las entrevistas, se investigaron distintos documentos y reportes disponibles para el proyecto “Pasturómetro”. Además, se amplió con información de diversas fuentes sobre el contexto del problema, ya que era un dominio desconocido para el grupo.

Posteriormente, se analizaron los datos relevados y se definieron los objetivos. Con el fin de validar esta información de forma intuitiva y coherente, se llevó a cabo una sesión especial. Para dicha presentación se preparó un prototipo gráfico que mostraba una potencial estructura e interfaz. Esto permitió ver fácilmente los flujos y acciones de la aplicación para dar una idea aproximada de la experiencia de usuario. Durante la reunión se validaron los prototipos, también llamados *wireframes*[1,2,3], y se ajustaron según los comentarios y sugerencias. Este proceso aseguró partir de una base clara para los referentes funcionales, y así evitar la necesidad de retrabajo tanto como sea posible.

Durante el desarrollo, se adoptó un enfoque de pruebas continuas, donde distintos *stakeholders* pudieron probar fácilmente los cambios implementados. Este fue un proceso simple, ya que se generaron archivos APK (Android Application Package) regularmente, lo que permitió al cliente instalarlo en su propio dispositivo. Así, se garantizó que los cambios sean bien recibidos y que cualquier problema o mejora fuera atendida debidamente.

Desde la concepción inicial hasta la implementación final se trabajó en estrecha colaboración para garantizar que la aplicación cumpliera con las expectativas y requisitos, y para abordar cualquier desafío de manera eficaz.

Metodología de diseño

El diseño de la aplicación se fundamentó en dos metodologías principales: el diseño basado en componentes y el diseño basado en módulos.

Diseño basado en módulos

El diseño basado en módulos estructura el *software* en módulos, que son segmentos de código que encapsulan un conjunto específico de funcionalidades y datos. Los módulos interactúan entre sí a través de interfaces bien definidas.

Sus principales ventajas son:

- Cohesión: Promueve la cohesión dentro de los módulos, agrupando funcionalidades relacionadas.

- Acoplamiento bajo: Fomenta un acoplamiento bajo entre módulos, lo que facilita la comprensión, el desarrollo y la prueba del *software*.
- Organización: Mejora la organización del código y facilita la navegación a través del sistema de *software*.

El diseño basado en módulos se seleccionó por su compatibilidad con las características específicas del proyecto y su potencial para mejorar la estructura general del código.

Diseño basado en componentes

El diseño basado en componentes se centra en la construcción de *software* a partir de componentes preexistentes o fácilmente definibles, que son unidades de *software* independientes y reutilizables. Estos pueden ser ensamblados y configurados para formar aplicaciones más complejas.

Sus principales ventajas son:

- Reutilización: Facilita la reutilización de componentes existentes, lo que puede reducir el tiempo y el costo de desarrollo.
- Mantenibilidad: Los componentes pueden ser modificados o reemplazados independientemente, lo que mejora la mantenibilidad del *software*.
- Escalabilidad: Permite la adición de nuevas funcionalidades con facilidad al añadir nuevos componentes.

La aproximación basada en componentes destaca por su eficacia en el desarrollo de interfaces de usuario, siendo un paradigma ampliamente adoptado en diversas bibliotecas y *frameworks* compatibles con este proyecto, tales como React, Flutter, Swift y Objective-C.

Metodología de desarrollo

El enfoque por el cual se optó fue el del desarrollo incremental. Esta metodología ha demostrado ser altamente eficaz en contextos dinámicos de desarrollo, centrándose en la entrega de funcionalidades en incrementos sucesivos.

Características clave de la metodología:

- **Incrementos bien definidos:** El proceso de desarrollo se organiza en incrementos, donde cada uno agrega funcionalidades específicas al producto. A diferencia de un

enfoque totalmente secuencial, el desarrollo incremental permite entregar valor al cliente de manera continua, con cada incremento se construye sobre el anterior hasta que el producto esté completo.

- **Entrega continua de valor:** Se adopta un enfoque de construcción progresiva del producto, característico del desarrollo incremental. En lugar de esperar a que todas las funcionalidades estén completas para hacer el lanzamiento, se entrega el producto en fases, permitiendo que el cliente utilice y obtenga valor de las partes completadas mientras se desarrollan los siguientes incrementos.
- **Flexibilidad y adaptabilidad:** Aunque el desarrollo incremental sigue un plan general, ofrece flexibilidad para adaptar, modificar o añadir nuevas funcionalidades en incrementos futuros basándose en el *feedback* recibido, las prioridades cambiantes y los nuevos requisitos identificados. Esto ayuda a asegurar que el producto final cumpla más efectivamente con las necesidades del usuario final.
- **Retroalimentación y ajustes posteriores a la entrega:** La naturaleza incremental permite recoger *feedback* sobre cada entrega y realizar ajustes en incrementos futuros. Esto asegura una mejora continua del producto y la satisfacción del usuario, ya que las modificaciones y mejoras pueden incorporarse en cualquier etapa del desarrollo.

El desarrollo incremental es particularmente útil en proyectos donde los requisitos no están completamente definidos desde el inicio o se espera que evolucionen durante el proyecto. Permite una mayor flexibilidad y eficiencia al permitir ajustes y añadir nuevas funcionalidades a lo largo del tiempo, asegurando que el producto final se alinee estrechamente con las expectativas del cliente y las demandas del mercado.

Análisis

Requerimientos

Finalizadas las reuniones pertinentes, se analizaron los datos obtenidos y se determinaron los siguientes requerimientos:

Requerimientos funcionales:

Recepción de mediciones

- Recibir las mediciones del pasturómetro.
- Almacenar mediciones recibidas.
- Mostrar últimas mediciones.
- Notificar fallas de mediciones.

Calibraciones

- Crear calibraciones a partir de una función o a partir de mediciones.
- Visualizar calibraciones.
- Cargar mediciones para una calibración.

Sectores*

- Abrir sector que permita organizar de forma temporal las mediciones*.
- Cerrar sector*.

Estadísticas

- Mostrar mediciones históricas.
- Cambiar el periodo a mostrar.
- Filtrar por potrero.
- Filtrar por sector*.

Potreros**:

- Creación de potreros**.
- Edición de potreros**.

Mapas

- Mostrar un mapa que permita ubicar los potreros.
- Permitir mostrar mediciones en su respectiva ubicación.
- Permitir filtrar mediciones por periodo, potrero y/o sector (*).
- Permitir descargar zonas del mapa para visualización *offline*.
- Soporte visual para la creación / edición de potreros**.

Sincronización

- Permitir al usuario iniciar / cerrar sesión.
- Descargar datos de la nube (potreros, calibraciones).
- Enviar mediciones a la nube.
- Enviar datos de calibraciones por mediciones.

Múltiples lenguajes (*)

- Ofrecer traducciones al inglés para cada texto de la aplicación.

***Los requerimientos fueron agregados durante el desarrollo.**

****Los requerimientos fueron descartados durante el desarrollo.**

Requerimientos no funcionales:

- Plataforma: La aplicación deberá funcionar en dispositivos con Android y iOS
- Contexto: Deberá funcionar en un campo, donde se puede carecer de conexión a internet, por lo que esperará a restablecer la conexión para enviar los datos recopilados sin bloquear funcionalidades.
- Usuarios: Deberá ser una aplicación intuitiva y accesible, ya que será utilizada por un amplio rango de usuarios que pueden muchas veces no estar alfabetizados. Entonces, los indicadores no textuales serán de suma importancia.
- Rendimiento: Deberá ser capaz de ejecutarse en dispositivos móviles de gama baja con hasta 7 años de antigüedad de manera fluida y sin limitar su funcionalidad.

Dominio del problema

Control del forraje en el pastoreo

Se conoce como control del forraje en el pastoreo a las prácticas y estrategias utilizadas para gestionar y monitorear el suministro de forraje disponible para el ganado en un sistema de pastoreo. Esto implica controlar la cantidad, calidad y disponibilidad del forraje en los pastizales.

El control del forraje en el pastoreo puede involucrar diversas acciones, como el monitoreo regular del crecimiento del pasto, la estimación de la cantidad de forraje disponible en el pastizal, la planificación de la carga animal para evitar la sobrepastoreo o la subutilización del forraje, y a

su vez requiere de la implementación de técnicas de manejo del pastoreo, como la rotación de potreros o el pastoreo diferido.

Medición de altura de pasturas

Una opción disponible para el control del pasto es la medición de su altura. El valor permite tener una idea del crecimiento del pasto en puntos específicos. Sin embargo, este análisis es limitado, ya que no da idea del forraje particular que se tiene. Para solventar esta problemática se utiliza un método indirecto que permite calcular el peso del pasto en relación con su altura. Es necesario para llevar a cabo esta metodología conocer una fórmula que tome como valor de entrada la altura y de salida el peso del forraje. A dicha fórmula se le llamará calibración.

Pesaje de pasturas

El dato que resulta relevante al problema es la cantidad de masa de pastura por área. Para obtener esta información, se puede emplear el método directo de cortar porciones de pasto y pesarlas. Conociendo el área cortada y su masa, se calcula la densidad de pasto por área. Sin embargo, debido a la heterogeneidad de las pasturas, este proceso debe repetirse en varias ocasiones para obtener un promedio con mayor exactitud.

Dado que este proceso consume una cantidad significativa de tiempo, generalmente se utiliza como punto de partida para otros métodos. Manteniendo las mismas especies y condiciones, la densidad superficial de pasto está correlacionada con la altura. Esto implica que, a partir de la medición del área y altura del pasto, se puede desarrollar una función de calibración que convierta la altura en kilogramos de pasto seco por hectárea.

Utilizando esta función de calibración, es posible realizar mediciones sin necesidad de cortar el pasto. Simplemente midiendo la altura del pasto en un punto determinado y aplicando la función de calibración, se puede estimar la cantidad de masa de pasto seco por hectárea. Este enfoque permite obtener datos sobre la cantidad de forraje disponible de manera más rápida y eficiente.

Entidades del dominio

Pasturómetro

Dispositivo conformado por una varilla y un disco deslizante que se utiliza sobre un pastizal para medir su altura. El volumen de pasto que resulta comprimido por el disco es medido por los

componentes electrónicos del pasturómetro. Estos valores son luego mostrados mediante la pantalla integrada y, además, transmitidas hacia el dispositivo móvil vinculado.

Medición

Dato medido por el pasturómetro con cada pulsación del botón central del dispositivo, representa la altura de pasto comprimido atrapada debajo del plato deslizante. Cada medición obtendrá una serie de valores correspondientes a cada uno de los sensores disponibles. La información obtenida será primeramente almacenada en el propio pasturómetro, en forma de memoria flash, y luego será enviada mediante BLE al dispositivo móvil previamente vinculado. La aplicación se encargará de discernir si existió algún error en la toma de mediciones, ya que los datos tomados por los sensores pueden presentar diferencias insalvables.

Calibración

El forraje no es único en todas las regiones, existen diferentes especies que pueden crecer incluso en un mismo potrero. Por otro lado, la época del año afecta de forma notable las condiciones y características del pasto. Por esto, y otros factores menos significativos, se necesita una forma de calibrar las mediciones y obtener valores independientes de estas condiciones. Se crea una calibración para un contexto específico que responda a una pastura y un periodo de tiempo particular. A partir de un cierto número de muestras, y su correspondiente peso, se puede estimar una calibración, que se define como un polinomio que aproxima el comportamiento real del pasto.

Potrero

Región de terreno que se tendrá en cuenta para la medición de pasturas. Debe ser identificable geográficamente por tres o más puntos coordinados. Cada potrero podrá tener distintas especies, por lo que deberá asociarse con la calibración adecuada para ese contexto.

Sector

Un sector es una región dentro de un potrero, de un tamaño menor. Se lo puede imaginar como un subpotrero. Los potreros son zonas de terreno que pueden ser muy extensos, esto implica complejidad en su análisis. Así, surge la necesidad de un control más granular y se designan los sectores. Dividir un potrero puede ayudar, por ejemplo, en el movimiento del ganado, acción que puede suceder varias veces por día.

Sistema *cloud*

Para funcionar correctamente, la *app* necesita intercambiar datos con un servidor mediante internet, que está comprendido por distintos subsistemas para completar su funcionalidad. "Sistema *cloud*" hace referencia a todos ellos, aunque se hace énfasis en el BFF mobile, ósea, un servidor pensado específicamente para el intercambio de datos con la *app*.

Proceso de negocio actual

Previo al desarrollo de la aplicación se investigó el proceso completo y su funcionamiento. Se relevó que existe un prototipo de pasturómetro que permite realizar mediciones y las almacena en una memoria microSD. Con este prototipo un trabajador recorre un potrero y toma las mediciones que crea necesarias sin ningún tipo de indicador sobre la medición más que el valor provisto por el display del dispositivo. El número arrojado por el pasturómetro puede o no reflejar el valor real del pasto. Este valor depende del *hardware* de ese pasturómetro particular, que está sujeto a cambios. Es posible que se envíe la altura de los sensores al disco, o su altura real, que se obtendrá si se tiene la altura del propio pasturómetro.

Una vez finalizada la medición, se retira la tarjeta microSD para su análisis. Los cálculos de la altura promedio y las correcciones deben realizarse manualmente en este punto. En este momento, solo se disponen de mediciones de altura, y no se cuenta con la masa de pasto por hectárea, que es el dato de utilidad final. Para obtener este dato, es necesario evaluar cada medición de altura en una función de calibración, la cual puede variar dependiendo del lugar y el momento en que se realizó la medición.

La función de calibración puede estimarse a partir de las especies de pasturas disponibles en el área de medición, aunque esto puede resultar en una menor precisión. Alternativamente, se puede calcular como un polinomio que aproxima los kilogramos de pasto seco en función de las alturas medidas. Para llevar a cabo este método, es necesario medir sucesivamente la altura en distintos puntos significativos y luego cortar el pasto de dicho sector para pesarlo. De esta manera se consiguen los puntos que se utilizarán para dicho polinomio.

Casos de uso

Para visualizar los diferentes roles dentro de un sistema y su relación se utilizó el diagrama de casos de uso. Permite identificar fácilmente los actores implicados y sus capacidades en el sistema. Teniendo en cuenta los requerimientos funcionales recopilados, se confeccionaron los diagramas de casos de uso presentes en el anexo.

A partir de ellos también se escribieron los casos de uso completos encontrados en el anexo de este informe.

Situación del campo argentino

En el territorio argentino la actividad agrícola ganadera tiene un enorme peso, tanto en el mercado local como en el extranjero. El desarrollo tecnológico está a la orden del día para optimizar los procesos y reducir pérdidas. Sin embargo, ciertas tecnologías deben ser importadas a altos precios y muchas veces no son amortizables en el agro argentino.

La medición del pasto mediante disco deslizante es efectuada con productos importados, por ejemplo de la marca Jenquip[4]. Por otro lado, son utilizados drones con cámaras o satélites que permiten conocer el estado actual de los terrenos.

Alternativas disponibles

El pasturómetro está inspirado en *Rising Plate Meter (RPM)*, tecnología desarrollada originalmente en Nueva Zelanda y describe un bastón de metal, con un disco o placa móvil que se mueve sobre un eje central. Este contiene un contador electrónico, que registra el movimiento del eje. Permite medir el pasto y prever cuál es el mejor momento para que el ganado se alimente.

La compañía más importante es sin duda Jenquip que ofreció la primera versión del RPM. Cuenta con varias versiones en el mercado, una manual y una con medición electrónica que muestra las mediciones y pesos en pantalla. Además, para la versión electrónica existen dos variantes con USB y con Bluetooth. Esta última requiere de un dispositivo móvil para recibir estos datos, procesarlos y enviarlos a la nube, por lo que hace uso de una *app*.

En el territorio latinoamericano es común el uso de esta herramienta, mayormente en Chile, aunque se utiliza principalmente la marca Jenquip.

Figura 2. Jenquip electrónico Bluetooth.

Figura 3. Aplicación para dispositivos Jenquip.

Sin embargo, no existe una tecnología similar en el ámbito local argentino. Se buscó entonces con el pasturómetro ofrecer una opción económica que no tenga toda la funcionalidad de su contraparte extranjera. El dispositivo es más económico porque delega gran parte a la aplicación, no requiere de GPS porque utiliza el del dispositivo móvil y no hace cálculos que requieran gran capacidad de cómputo, por lo que los realiza la *app*.

Además, se consideró la aplicación de Jenquip poco intuitiva y escasamente reactiva, basándose en eso se buscó evitar la necesidad de lectura de textos tanto como sea posible y que los elementos visuales tengan una coherencia estética.

Diseño

Tecnologías

Elección de *framework*

En la búsqueda de optimizar la eficiencia del desarrollo y evitar la duplicación de código para distintas plataformas, se tomó la decisión de emplear un *framework* multiplataforma en este proyecto. Aunque en la actualidad solo se compila una versión para Android, es importante destacar que, desde la perspectiva técnica, es posible generar una versión para iOS a partir del código existente. Sin embargo, esta opción no se ha implementado debido a la necesidad de disponer de *hardware* específico y el requisito de una cuenta de desarrollo de Apple para llevar a cabo el proceso de compilación y ejecución en iOS.

Al analizar las alternativas disponibles en el panorama de los *frameworks* multiplataforma, se consideraron dos opciones principales: Flutter[5] y React Native[6]. Estas elecciones se basaron en su relevancia y popularidad en la comunidad de desarrollo móvil. Un factor crucial que influyó en esta decisión fue la presencia de comunidades activas que ofrecen una amplia gama de recursos, documentación y bibliotecas de apoyo.

Es importante señalar que tanto Flutter como React Native son productos respaldados por gigantes tecnológicos, Google y Meta (anteriormente conocida como Facebook), respectivamente. Esta asociación con empresas líderes en la industria refuerza la confiabilidad y el compromiso a largo plazo de estos *frameworks*, lo que los convierte en opciones sólidas para el desarrollo de aplicaciones multiplataforma.

Las principales diferencias es que Flutter compila aplicaciones a código nativo, estas aplicaciones utilizan un motor gráfico propio, actualmente Skia, para el renderizado. Por otra parte, React Native utiliza React para comunicarse con componentes nativos del sistema y utiliza las capacidades de renderizado ya existentes. Finalmente, en cuanto a la ejecución de código, React Native cuenta con su propio motor de JavaScript llamado Hermes, el cual está optimizado especialmente para este *framework*.

Flutter

Ventajas

1. **Rendimiento sólido**: Flutter compila su código en código nativo, lo que a menudo conduce a un rendimiento excepcionalmente bueno en comparación con otros *frameworks* multiplataforma.
2. ***Widgets* personalizables**: Flutter ofrece un amplio conjunto de *widgets* personalizables, lo que facilita la creación de interfaces de usuario altamente personalizadas sin un alto costo en el rendimiento.
3. **Documentación y comunidad activa**: Flutter cuenta con una documentación completa y una comunidad activa que proporciona soluciones a problemas comunes y actualizaciones frecuentes.

Desventajas

1. **Curva de aprendizaje**: Si los desarrolladores no tienen experiencia previa en Dart (el lenguaje de programación de Flutter), puede requerir tiempo adicional para aprenderlo.

React Native

Ventajas

1. **Conocimiento existente**: Si el equipo de desarrollo ya tiene experiencia en React, la transición a React Native será más suave, lo que aceleraría el desarrollo.
2. **Gran comunidad y madurez**: React Native ha estado en el mercado durante varios años y cuenta con una comunidad activa y una amplia gama de recursos disponibles.

Desventajas

1. **Problemas de rendimiento en casos complejos**: Puede haber problemas de rendimiento en aplicaciones extremadamente complejas o con muchas animaciones, lo que requeriría el uso de librerías de terceros para contrarrestar.

Expo

La elección de utilizar Expo[7] nace de una recomendación hallada en la propia página de React Native. Expo es un *framework* de código abierto que proporciona una serie de herramientas y un

flujo de desarrollo que simplifica significativamente la visualización de los cambios en tiempo real a medida que se modifica el código en JavaScript o TypeScript, entre otras cosas.

Al analizar las ventajas y desventajas de utilizar Expo, se descubrió que muchas de las desventajas que se mencionaron previamente ya no son relevantes debido al rápido avance en su desarrollo en los últimos años. Entre estas desventajas superadas se incluían las dificultades para utilizar módulos nativos, que eran esenciales para la comunicación a través de Bluetooth Low Energy con el pasturómetro.

Ventajas de Expo

Expo SDK

Uno de los desafíos comunes en el desarrollo de aplicaciones móviles con React Native es la limitación de las bibliotecas básicas. Expo aborda este problema con sus Expo SDK, que dan soporte a funcionalidades frecuentes en dispositivos móviles, como cámara, calendario, contactos, video, audio, códigos de barras, entre otros, que dependen de funcionalidades nativas. En este proyecto, se han utilizado expo-localization y expo-notifications de Expo SDK para satisfacer las necesidades específicas.

Expo Go

Expo Go es una aplicación que integra todas las bibliotecas nativas de Expo SDK y permite importar un *build* a través de la red local, esta se descarga fácilmente desde la App Store o Play Store. La característica sorprendente de Expo Go es su capacidad para cargar automáticamente los cambios realizados en la máquina de desarrollo al dispositivo en tiempo real. Esta capacidad transforma el proceso de desarrollo en una experiencia interactiva. En contraste, en ausencia de Expo Go, se requeriría compilar a Swift o Kotlin y luego en binarios antes de poder instalar y ejecutar la aplicación en el dispositivo, un proceso mucho más laborioso y menos ágil.

Es importante aclarar que Expo Go no es simplemente un visor web mejorado, sino que ejecuta el código JavaScript/TypeScript utilizando el motor Hermes, optimizado específicamente para React Native.

Figura 4 - Expo local connection

Clientes de desarrollo Expo

Cuando se necesita un módulo nativo no incluido dentro de Expo SDK no se lo podrá referenciar desde la aplicación Expo Go, por esta razón existen los clientes de desarrollo Expo. Se trata de extensiones de Expo Go con módulos nativos sumados por el desarrollador, lo que permite mantener el flujo antes descrito.

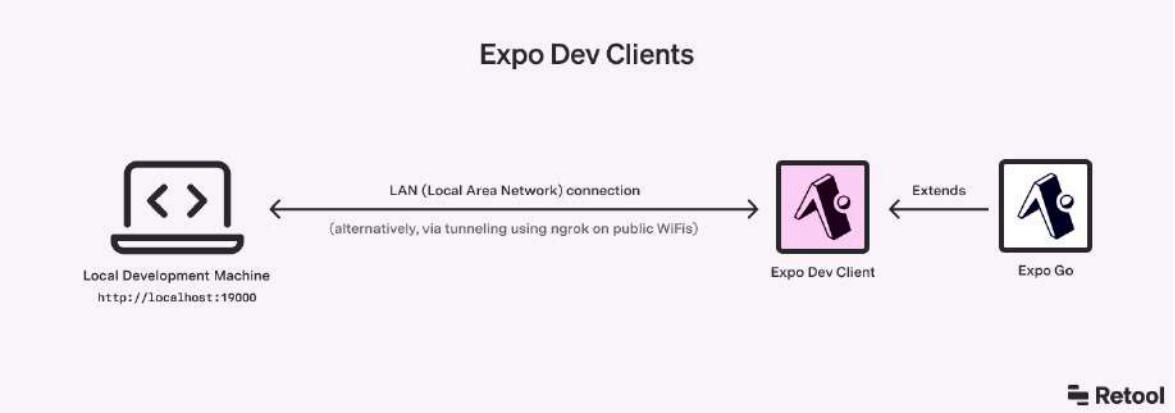


Figura 5 - Clientes de desarrollo Expo

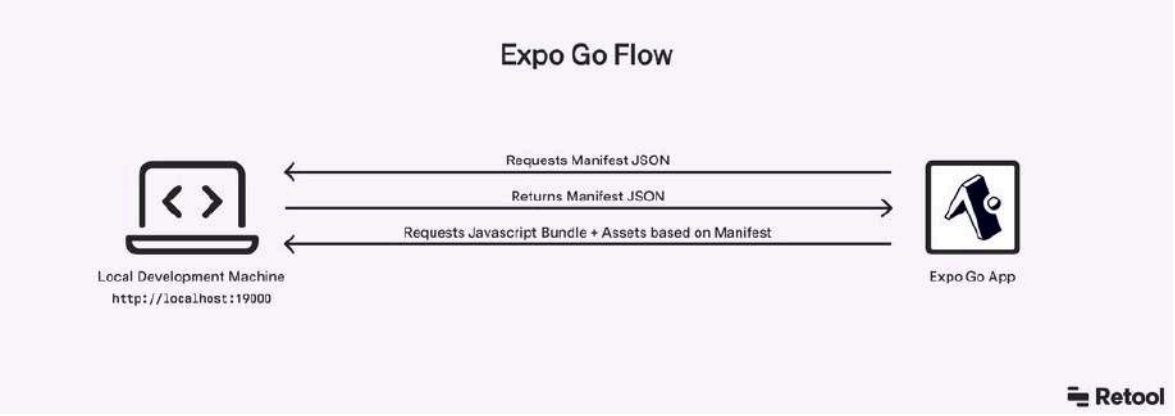


Figura 6 - Flujo de Expo Go

Expo Application Services (EAS)

EAS (Expo Application Services) es un conjunto de servicios ofrecidos por Expo Inc., diseñado para compilar y enviar aplicaciones a las tiendas de aplicaciones. Una de las ventajas más notables de EAS es su capacidad para compilar y cargar versiones de iOS en la App Store, automatizando el proceso y evitando demoras en la migración de datos entre plataformas. Es importante destacar que EAS no es de código abierto y representa el modelo de negocio principal de Expo.

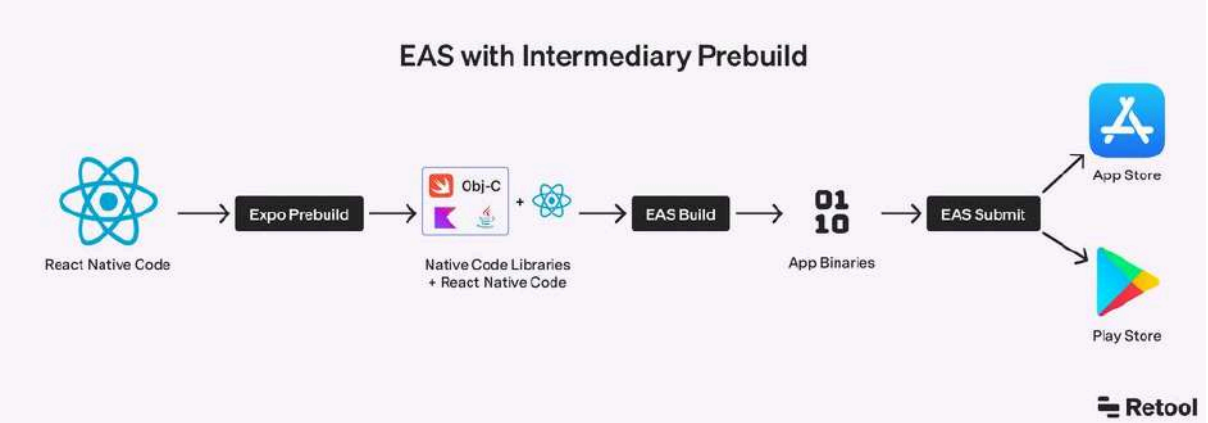


Figura 7 - EAS con preconstrucción intermedia

Valoración y elección

Dada la comparación anterior y considerando los objetivos y requisitos específicos del proyecto, la elección de utilizar React Native con Expo parece altamente ventajosa.

- **Eficiencia en el desarrollo:** React Native con Expo ofrece una eficiencia significativa en el desarrollo, permitiendo el uso compartido de código y proporcionando herramientas que simplifican la previsualización y la depuración en tiempo real.
- **Conocimiento del equipo:** Teniendo en cuenta la experiencia previa del equipo en JavaScript y React, esta elección permite aprovechar al máximo las habilidades existentes y acelerar el desarrollo.
- **Facilidad de mantenimiento:** Contar con una base de código compartida simplifica el mantenimiento a lo largo del ciclo de vida de la aplicación, lo que puede reducir los costos a largo plazo.
- **Compatibilidad multiplataforma:** React Native con Expo es compatible con iOS y Android, lo que garantiza llegar a una amplia audiencia con una sola base de código.

Si bien es cierto que Expo puede imponer algunas limitaciones en la personalización y que las aplicaciones Expo pueden ser ligeramente más grandes, en el contexto de este proyecto, estas limitaciones de pequeña escala son justificables debido a los beneficios aportados.

La elección de utilizar React Native con Expo como la tecnología principal para el desarrollo de la aplicación móvil es respaldada por su eficiencia, la experiencia previa del equipo y la capacidad de cumplir con los objetivos y requisitos específicos. Esta elección permite desarrollar una aplicación de alta calidad de manera eficiente y efectiva.

Lenguaje de programación

Pese a haber escogido el *framework* a utilizar, se presenta una nueva interrogante, ¿qué lenguaje de programación impulsará el desarrollo de la aplicación? Al elegir React Native se acota el espacio de posibilidades a solo dos, JavaScript y TypeScript.

JavaScript

JavaScript[7] es un lenguaje de programación ligero, interpretado y débilmente tipado. Es ampliamente popular por su utilización en *scripting* de páginas web desde el lado del cliente. Además, al ser un lenguaje dinámico, multiparadigma, basado en prototipos y que admite diversos estilos de programación (POO, imperativos, funcionales) permite su uso fuera del entorno web.

Entre sus aplicaciones más importantes se encuentran Node.js —permitiendo a los desarrolladores construir aplicaciones del lado del servidor y de red con JavaScript—, Apache CouchDB o Adobe Acrobat. Sin embargo, su empleo más relevante para el proyecto es la creación de aplicaciones mediante el uso de *frameworks* como Vue, Angular, React y React Native.

TypeScript

JavaScript, a diferencia de otros lenguajes, carece de un sistema de verificación estática de tipos, ya que es un lenguaje de tipado dinámico. Esto significa que el tipo de cualquier variable o atributo de objeto puede cambiar durante la ejecución del programa. Esta flexibilidad en la gestión de tipos durante el desarrollo puede aumentar la complejidad y el potencial de errores, puesto que no se pueden establecer contratos estáticos a verificar para garantizar la coherencia de las funciones u objetos. Para hacer frente a esta problemática específica se desarrolló TypeScript[8,9].

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. Anders Hejlsberg, diseñador de C# y creador de Delphi y Turbo Pascal, ha trabajado en el desarrollo de TypeScript.

Con TypeScript se puede detectar errores de tipos tanto en tiempo de transpilación (conversión de un lenguaje a otro) a JavaScript como también durante la generación de código en los entornos de desarrollo más populares. En estos entornos el uso de este lenguaje también provee otras ventajas como sugerencias de código, teniendo en cuenta los contratos a cumplir e información de las interfaces de las entidades seleccionadas.

También brinda la ventaja que al utilizar tipos estáticos, el código tiende a ser más legible y autoexplicativo, ya que los tipos proporcionan documentación implícita sobre la estructura y el uso de los datos.

Elección

Frente a un proyecto que debe ser escalable y que potencialmente será entregado a un nuevo equipo de desarrollo a futuro, la mejor opción es utilizar TypeScript. Los flujos y las tareas serán más simples de comprender si se explicitan todas las interfaces en el propio código.

Arquitectura

Se optó por una estrategia de modularización del proyecto. Este enfoque permitió descomponer el sistema en módulos claramente definidos, logrando así un bajo acoplamiento entre las distintas partes del código. Gracias a esta estructuración, cada módulo mantiene una cohesión interna alta, contribuyendo a una organización más lógica y un mantenimiento simplificado del sistema. Además, la división modular facilitó la colaboración del equipo, posibilitando el desarrollo concurrente en diferentes segmentos del proyecto.

Módulos

Store

Para el manejo de estados de la aplicación se utilizó Redux, esta librería de JavaScript cuenta con madurez en el mercado y ya forma parte de un estándar bien conocido. Posibilita la actualización atómica de componentes visuales a lo largo de toda la aplicación de manera eficiente, limpia y fácil de testear.

Base de datos local

Se utilizó una base de datos local para almacenar los datos que se enviarán a la *app* mientras el sistema *cloud* no sea alcanzable, esto a su vez le otorga autonomía a la aplicación para poder realizar consultas más fácilmente sobre sus datos locales.

La base de datos elegida fue SQLite, una base de datos relacional ampliamente adoptada en el desarrollo móvil.

Conexión al dispositivo

Este módulo se encarga de manejar el estado de conexión con dispositivos, como también sus configuraciones, recepción de datos y tratamiento antes de persistirlos. Se encapsula la librería react-native-ble-plx.

Mediciones

Este módulo integra todas las definiciones y las funcionalidades necesarias para gestionar los datos de mediciones. Incluye tanto la lógica de procesamiento de datos como los componentes de la interfaz gráfica.

Calibraciones

El módulo de Calibraciones centraliza las definiciones y operaciones relacionadas con los datos de calibración. También define la interfaz de usuario como el flujo sobre ella.

Estadísticas

Su objetivo es mostrar al usuario datos recopilados a partir de las mediciones previamente realizadas, como la altura media o el peso de pasto medio —para lo cual permite elegir una calibración—. Provee la opción de filtrar los datos por fecha, potrero o sector.

Potreros

Permite la visualización de potreros y mediciones realizadas sobre un mapa de referencia. Además, exhibe la posibilidad de filtrar las mediciones a mostrar por fecha, potrero y sector. Finalmente, cuenta con la funcionalidad de descargar los mapas para su uso sin conexión.

Cloud

Hace posible el manejo de sesión de usuario, envío y recepción de datos al sistema *cloud*.

Diagrama de componentes

Este tipo de diagrama permite visualizar la estructura del sistema y cómo se comunican los distintos módulos principales entre sí y sus dependencias.

El conocimiento de estas dependencias clarifica la planificación del desarrollo, ya que explicita el orden de precedencia de cada módulo. A partir de esto, se puede deducir qué módulos pueden iniciarse primero y cuáles son necesarios antes de comenzar otros.

Por otra parte, permite visualizar de forma directa la interacción con otros sistemas y sus protocolos de comunicación.

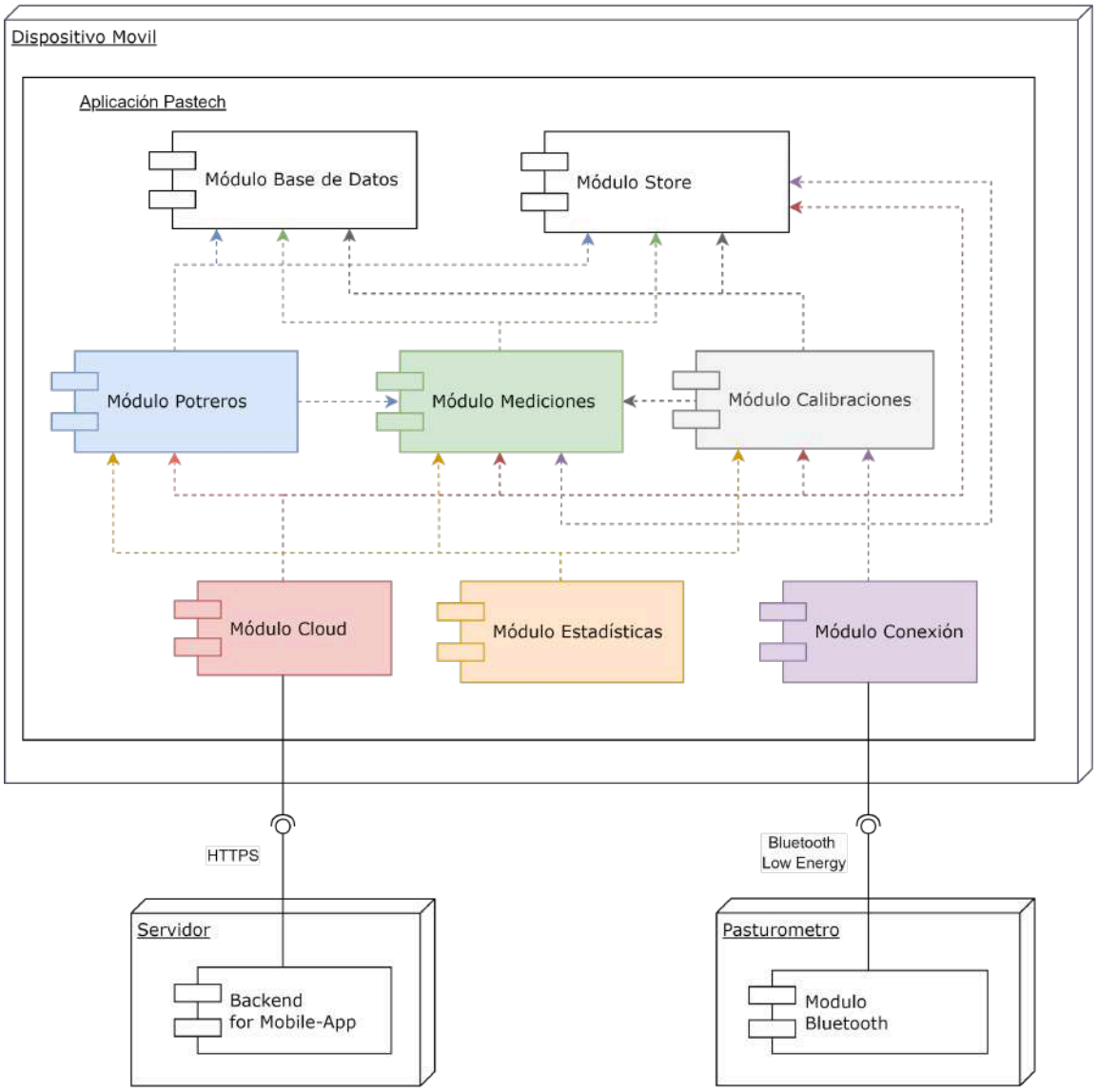


Figura 8 - Diagrama de componentes

Store

Un *Store* se utiliza para manejar en memoria un estado que sea accesible de forma global en la aplicación. De esta forma se puede mantener una estructura compleja sin generar problemas de rendimiento o consistencia a la hora de refrescar los componentes que son dependientes de tales datos.

Librería

En cuanto al manejo de datos para una aplicación React, se encontraron las siguientes principales alternativas:

- **Redux**[11]**:** Una biblioteca de gestión de estado predecible para aplicaciones JavaScript/TypeScript, ideal para aplicaciones con interacciones de estado complejas y manejo de datos asíncronos. Su arquitectura basada en el patrón Flux facilita el seguimiento de los cambios de estado a través del tiempo.
- **Recoil**[12]**:** Una biblioteca de gestión de estado para React con un enfoque más moderno, facilitando la gestión del estado con conceptos de átomos y selectores, lo que permite un flujo de datos más reactivos y una mejor integración con las características de React.
- **Context API**[13]**:** Incluida en React para compartir datos globalmente entre componentes. Es más simple y directa que Redux para casos de uso menos complejos, pero puede complicarse y generar problemas de rendimiento en aplicaciones grandes.
- **Zustand**[14] es una biblioteca minimalista y rápida para la gestión del estado en React. Destaca por su simplicidad y su aproximación directa, permitiendo crear "*Stores*" globales sin la complejidad de Redux o la necesidad de los proveedores de Context API. Es especialmente adecuada para proyectos que buscan una solución sencilla y eficaz para el manejo del estado, con una curva de aprendizaje baja y un enfoque más directo hacia la reactividad y el acceso al estado global.

Se eligió Redux por varias razones fundamentales:

- **Robustez y fiabilidad:** su arquitectura simple, pero poderosa, permite mantener el estado de la aplicación de manera predecible y escalable, lo que es crucial para el desarrollo de aplicaciones complejas a largo plazo al facilitar la depuración y el mantenimiento del código.
- **Ampliamente utilizado en la comunidad de React y React Native:** esto significa que hay una gran cantidad de recursos disponibles, como documentación detallada, librerías, ejemplos y soporte.

- **Escalabilidad:** al adoptar Redux, se está preparado para un crecimiento futuro. A medida que la aplicación evoluciona y se agregan más funcionalidades, Redux proporciona una estructura organizada y coherente para gestionar el estado de la aplicación, lo que facilita la incorporación de nuevas características y la gestión de un mayor volumen de datos sin comprometer la eficiencia ni la estabilidad.

Base de datos local

Dado el contexto en el que se utilizará la *app*, que es predominantemente rural, será común que no se disponga de una conexión directa a la nube. Esta condición conduce a la aplicación a ser capaz de gestionar un volumen de datos considerable de manera local. Debido a que los datos obtenidos por la aplicación no tienen un tiempo definido para su envío, no existe certeza sobre cuándo se podrán enviar. Por otra parte, tratar los datos de manera local brinda el beneficio de poder realizar estadísticas básicas, incluso en ausencia de conexión con el servidor.

Para lograr la persistencia de estos datos, se descartó la opción de simplemente guardarlos como archivos JSON en el sistema de archivos local. Esta decisión se basa en la consideración de que, con el tiempo, el volumen de datos podría llegar a ser significativo. Entonces, no se dispondría de las herramientas necesarias para el tratamiento de datos que ofrece un sistema de base de datos convencional.

El primer paso fue considerar qué tipo de bases de datos se utilizaría. Partiendo de la clasificación de base de datos relacionales y no relacionales.

Bases de datos relacionales y no relacionales

Las bases de datos relacionales y no relacionales son dos métodos de almacenamiento de datos para aplicaciones. Una base de datos relacional almacena los datos en formato tabular con filas y columnas. Las columnas contienen atributos de datos, mientras que en las filas hay valores de datos. Se pueden vincular las tablas de una base de datos relacional para obtener información más profunda sobre la interconexión entre diversos puntos de datos. Por otra parte, las bases de datos no relacionales utilizan diversos modelos de datos para acceder a estos y administrarlos. En estas es simple tener datos con mucha variabilidad de estructura y tamaños, siendo eficientes en cuanto al almacenamiento. Están optimizadas específicamente para aplicaciones que requieren grandes volúmenes de datos, baja latencia y modelos de datos flexibles, lo que se logra mediante la flexibilización de algunas de las restricciones de coherencia de datos en otras bases de datos[15]. Usualmente, se asocia el concepto de base de datos relacional con el del Structured

Query Language (SQL) que es su mayor exponente, mientras que las no relacionales se vinculan con las noSQL. Todas aquellas BD que no son categorizadas como SQL se les define noSQL.

Necesidades de la aplicación

Los datos que serán almacenados por la *app* pueden ser fácilmente representados por una estructura relacional. Los esquemas que modelan la aplicación se pueden considerar sencillos, ya que no varían ni su tamaño ni su estructura. Una base de datos noSQL no presenta ninguna ventaja en el manejo de información, puesto que las estructuras no son dinámicas. En cuanto a escalabilidad, al ser local solo impacta la vertical y para el caso de uso específico del proyecto SQL no presenta desventajas.

Un factor importante a analizar fue el conocimiento sobre estas tecnologías, el equipo de desarrollo contaba con una mayor experiencia en lenguajes relacionales, lo que redujo el tiempo de aprendizaje total. El precio no fue un punto a tener en cuenta, ya que ambos grupos tienen opciones gratuitas y de alta calidad. Además, se debe tener en cuenta la madurez de ambos modelos, SQL cuenta con casi 5 décadas en el mercado, mientras que las noSQL con poco más de 2.

Finalmente, con la información recabada se pudo concluir que utilizar una base de datos relacional o SQL era lo ideal. Se tuvo en cuenta principalmente el aprendizaje, la madurez del lenguaje y la naturaleza estática de las estructuras planteadas.

Motor de base de datos

Luego de haber elegido la clase de base datos fue necesario elegir el motor. Se partió con la limitación de tener que funcionar sobre un dispositivo móvil, debía funcionar para Android y iOS, a su vez se trabajó con Expo, así que la mejor opción era considerar las opciones que tuvieran mayor compatibilidad.

En la documentación de Expo se pueden encontrar las librerías recomendadas. La implementación propuesta por Expo es SQLite[16].

SQLite es un motor de base de datos escrito en el lenguaje de programación C. No es una aplicación independiente; en su lugar, es una biblioteca para incorporar en aplicaciones. Como tal, pertenece a la familia de bases de datos integradas. Es el motor de base de datos más ampliamente implementado, ya que es utilizado por varios de los principales navegadores web, sistemas operativos, teléfonos móviles y otros sistemas integrados. Es ideal para aplicaciones

que necesitan almacenar datos de forma local y que operan en dispositivos con recursos limitados, como aplicaciones móviles y de IoT. SQLite utiliza un archivo único para almacenar toda la base de datos, lo que facilita su portabilidad y gestión. A pesar de su tamaño compacto, SQLite ofrece soporte para la mayoría de las características estándar de SQL, incluidas transacciones ACID, índices, subconsultas y disparadores. Esto lo convierte en una opción popular para aplicaciones que requieren una base de datos local fácil de usar y de alto rendimiento[17] [18].

Expo SDK cuenta con una versión compilada de SQLite a módulos nativos de iOS y Android. Este paquete en particular cuenta con 20 mil descargas semanales y mantenimiento continuo.

Conexión al dispositivo

Librería

Para la implementación de la conexión con el dispositivo se analizaron las opciones disponibles, las opciones son pocas y los esfuerzos para este tipo de soluciones estaba distribuido principalmente en dos proyectos, react-native-ble-plx y react-native-ble-manager.

Popularidad y Soporte de la Comunidad: react-native-ble-plx tiende a ser más popular entre los desarrolladores, con más estrellas en GitHub y un número menor de problemas abiertos en comparación con react-native-ble-manager. Esto indica un soporte más robusto y una comunidad más activa, lo que es importante para la resolución de problemas y el desarrollo de nuevas características.

El equipo tiene la hipótesis de que la diferencia en popularidad entre react-native-ble-plx y react-native-ble-manager se atribuye a factores históricos asociados a Expo. React-native-ble-plx ofrece un plugin para su uso en proyectos de Expo a diferencia de react-native-ble-manager, esto presentaba una ventaja crucial antes de junio de 2020. Ya que antes de esta fecha, Expo no permitía la utilización directa de módulos nativos, lo que hacía esenciales dichos plugins para integrar capacidades BLE en proyectos de Expo.

Plugin de Expo: Si bien antes era indispensable para el desarrollo con Expo, hoy día facilita la automatización de la configuración de permisos en distintas plataformas durante la compilación, optimizando así el flujo de trabajo de CI/CD[19,20].

A partir de estos puntos se eligió utilizar react-native-ble-plx.

Integración

Para la integración, se crea una instancia única del administrador de Bluetooth Low Energy proporcionado por la librería *react-native-ble-plx*, utilizando el patrón singleton. Alrededor de esta instancia se desarrolla una interfaz funcional para gestionar la lógica específica del proyecto. Durante el proceso de conexión, se asigna un *callback* para el evento de recepción de paquetes BLE. Este controlador invoca funciones específicas para interactuar con la base de datos y el *Store*, con el fin de actualizar el modelo de datos de acuerdo a la información recibida.

Estados de conexión

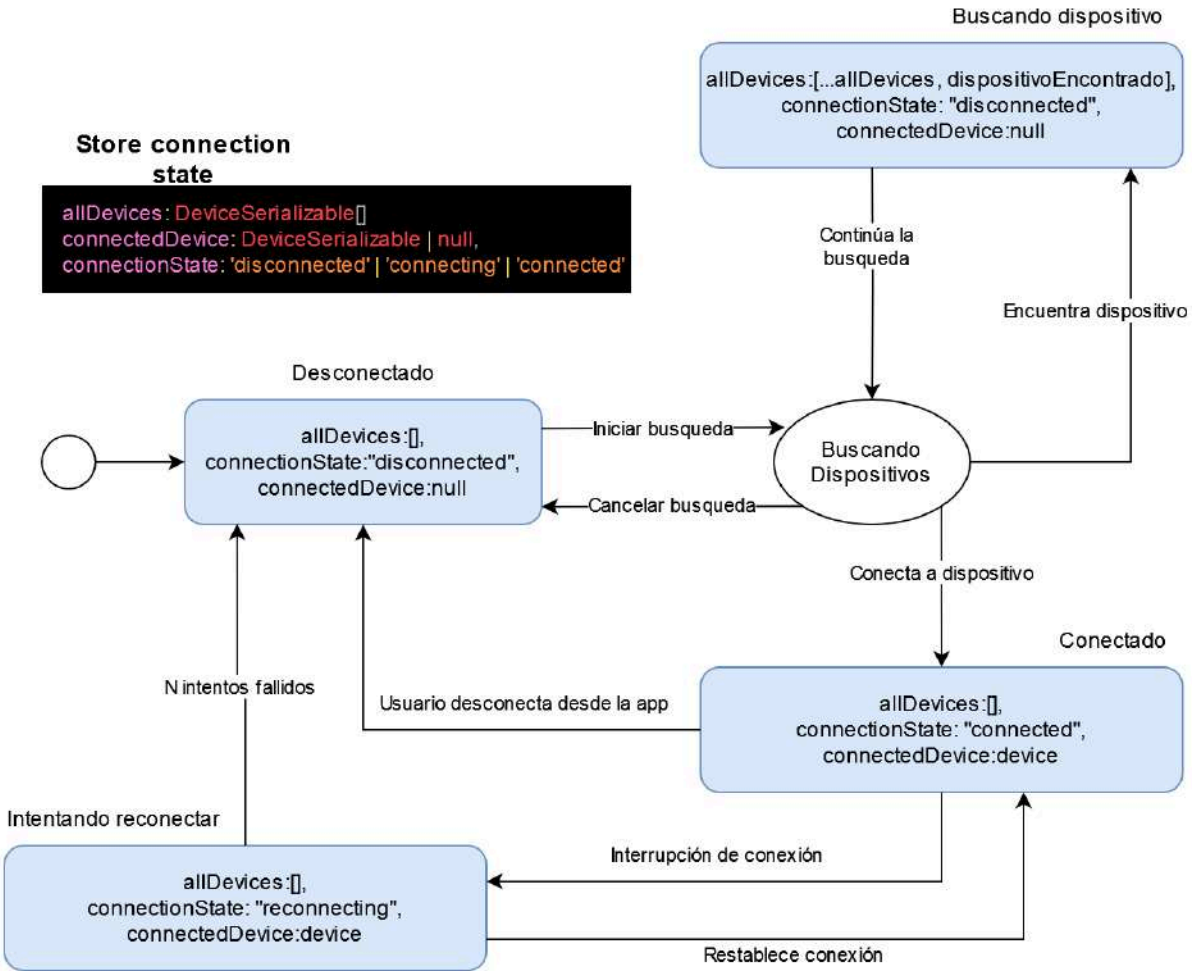


Figura 9 - Diagrama de estados de conexión.

El diagrama muestra como se ha modelado el cambio de estado en la aplicación a partir de las acciones que se realizan, existe un estado que se encuentra fuera del *Store* y se trata del de búsqueda. Según la estructura actual de la aplicación se incluye dentro del estado desconectado. No obstante, no hay restricciones que impidan que este estado de búsqueda se presente en otros contextos. Pudo haberse agregado un dato de tipo booleano al estado para indicar si se está buscando o no, pero por la dinámica y flujo actuales de uso de la aplicación tampoco se consideró necesario.

Diagramas de secuencia BLE

Buscar Dispositivo

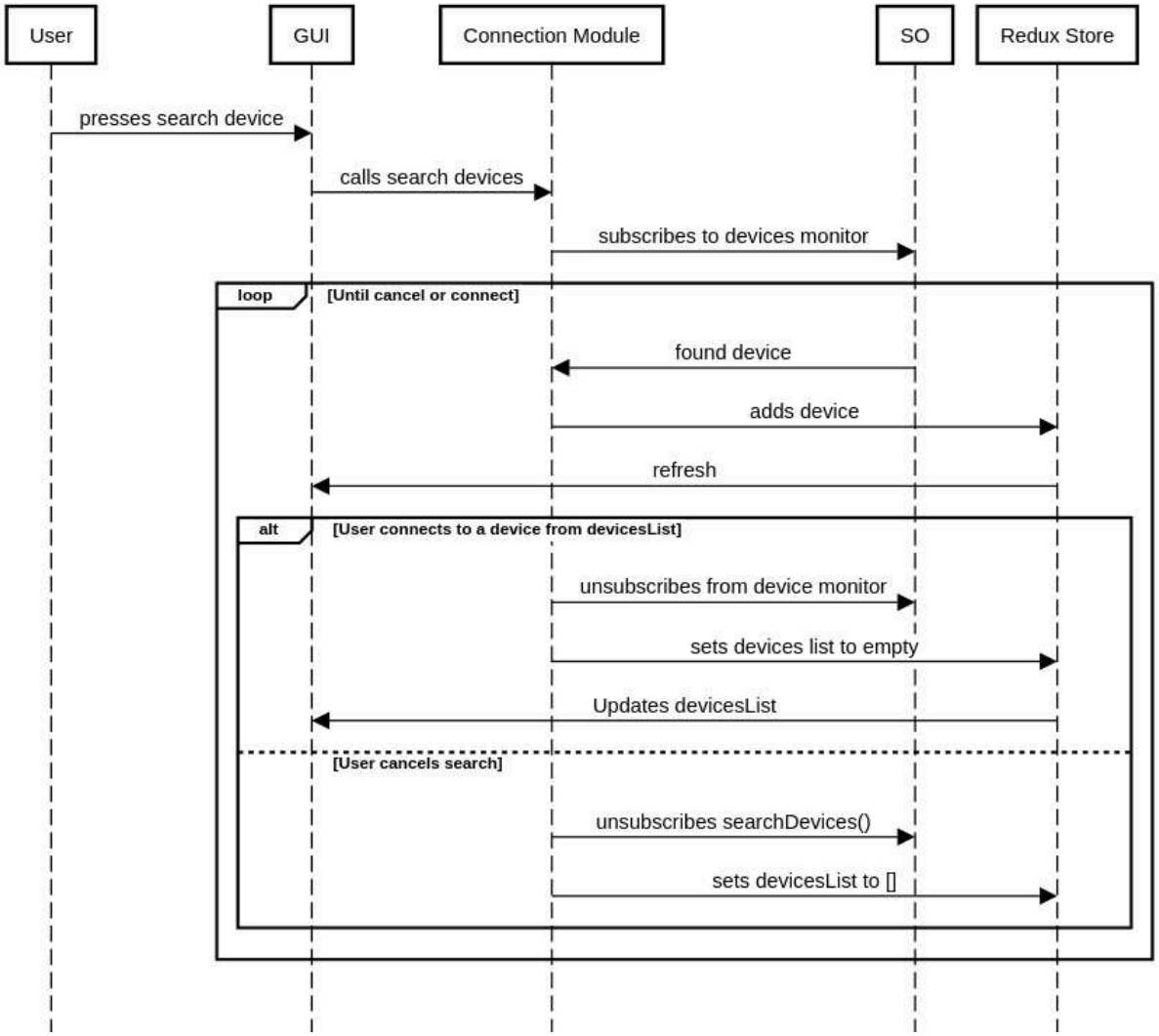


Figura 10 - Diagrama de secuencia Buscar Dispositivo

- **Subscribes to device monitor** se refiere a la función del sistema operativo para buscar y suscribirse a eventos de búsqueda Bluetooth, realizada a través de la biblioteca react-native-ble-plx.
- **Search devices** es una función dentro del módulo ble creado para el proyecto, encapsula el proceso de iniciar la búsqueda de dispositivos y manejar los dispositivos encontrados dentro de la funcionalidad Bluetooth de la aplicación. En el patrón MVC cumpliría la función de controlador.

Conectar Dispositivo

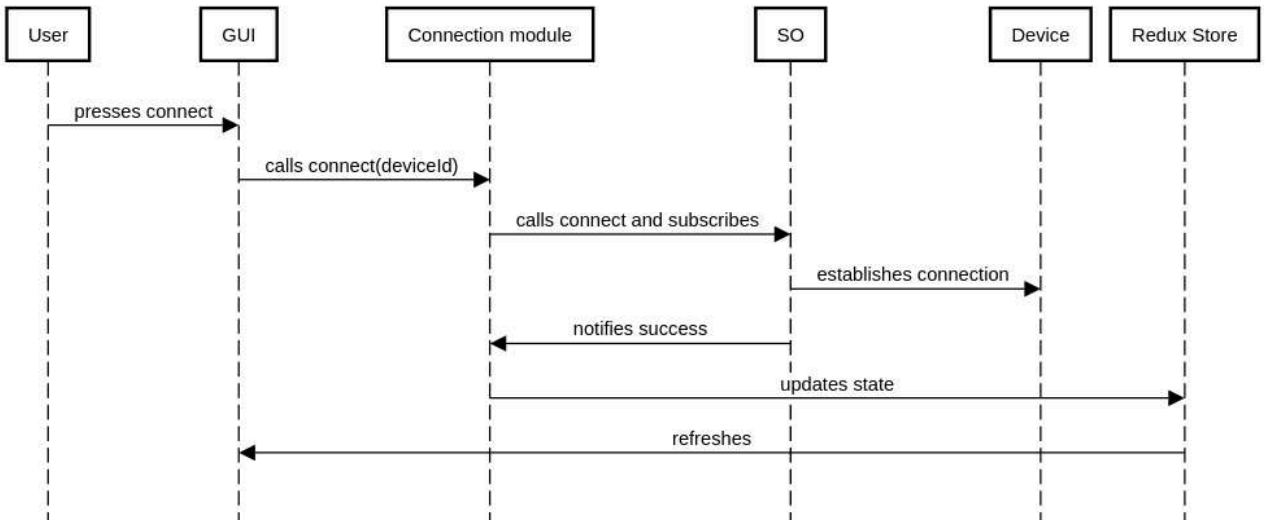


Figura 11 - Diagrama de secuencia Conectar Dispositivo.

Recibir datos desde el Dispositivo

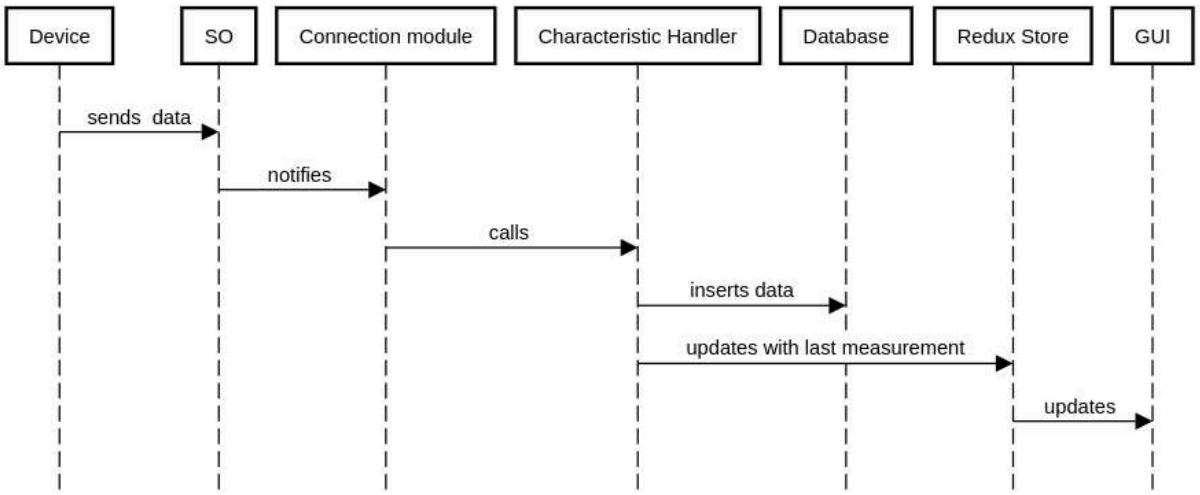


Figura 12 - Diagrama de secuencia de recepción de datos.

Characteristic Handler es una entidad creada para manejar los datos recibidos de las características Bluetooth de un dispositivo.

Potreros

Objetivo

La realización de este módulo requirió de la implementación de un mapa sobre el cual se pueden visualizar mediciones históricas, límites de potreros y sus terrenos circundantes. Para llevar a cabo esto se barajaron distintas librerías. Fue necesario que proporcionen también herramientas para localizar al usuario, utilizar mapas personalizados y permitir visualización sin conectividad a internet. Se priorizaría el menor costo posible tanto por factores económicos, como también por factores logísticos.

Librería

Para la elección de una librería que diera soporte a los mapas, el primer paso fue partir de la tecnología elegida. React Native cuenta con un catálogo extenso de proyectos compatibles para esta tarea, sin embargo, muchos de ellos carecían de un soporte activo o tenían un sistema de monetización con límites establecidos. Esto llevó a un análisis de las prioridades de la aplicación móvil.

React-native-maps (RNMaps)[21]**:** Como primera opción, esta librería ofrece una experiencia basada en Google Maps, utilizando su interfaz de usuario ya familiar para muchos usuarios. Cuenta con una amplia variedad de elementos y herramientas que son suficientes para cumplir con las necesidades de la aplicación. Con estas capacidades es posible manipular marcadores en el mapa, visualizar potreros (polígonos), utilizar mapas personalizados (mediante Tiles), localizar al usuario, entre otras opciones.

- **Soporte:** Alto, es la librería más activa de estas características y tiene una documentación amplia y detallada.

- **Precio:** Para su utilización es necesario proveer una API Key vinculada a una cuenta de Google Cloud. Con este servicio se ofrecen 200 USD en créditos mensuales gratuitos, posteriormente superando una cuota de usuarios se comienza a pagar proporcionalmente como se ve en la figura 12[22].

UA	Costo USD
Menos de 28500	Gratis ($200.00 siendo la totalidad del crédito)
Entre 28500 y 100000	$7.00 / 1000 UA
Más de 100000	$5.60 / 1000 UA

Figura 12 - Precios por Usuarios Activos (UA) RNMaps

Mapbox[23]**:** La segunda posibilidad se presenta en Mapbox, una librería que antaño ofrecía sus servicios de forma totalmente gratuita, sin embargo, cambió sus condiciones de uso recientemente. Utiliza su propio sistema de mapas que exhibe un rendimiento ligeramente mayor que aquel de Google Maps. Si bien tiene menos herramientas que RNMaps, cuenta con las necesarias para su implementación dentro de la aplicación. Admite el uso de marcadores, polígonos, Tiles y localización de usuario.

- **Soporte:** Bueno, tiene un gran soporte con actualizaciones frecuentes y documentación detallada, sin embargo, no recibe tantas actualizaciones como RNMaps y no cuenta con tantos ejemplos de uso.

- **Precio:** Se requiere una cuenta de Mapbox que ofrece de forma gratuita hasta 25000 Usuarios Activos (UA). Esta forma de presentación es más sencilla de visualizar que los 200 USD de crédito de RNMaps, sin embargo, como se vio en la figura 12, la utilización de todos los créditos ofrece hasta 28500 UA siendo ligeramente superior. Los precios de las distintas categorías se pueden ver en la figura 13.

UA	Costo USD
Menos de 25000	Gratis
Entre 25000 y 125000	$4.00 / 1000 UA
Entre 125000 y 250000	$3.20 / 1000 UA
Entre 250000 y 1250000	$2.40 / 1000 UA

Figura 13 - Precios por Usuarios Activos (UA) Mapbox

MapLibre[24]**:** Las librerías previamente mencionadas tienen un uso gratuito mientras no se superen ciertos límites. En la previsión actual del proyecto no es posible determinar si ese límite

es suficiente para el resto de la vida útil de la aplicación. Surge entonces una alternativa completamente gratuita que no requerirá inversión futura como lo es MapLibre. Nacido de la versión previa de Mapbox, versión que supo ser gratuita y que es retomada por este proyecto de menor calibre. Su utilización es mucho más limitada y marca un punto de ruptura para comenzar a verificar las necesidades reales de la aplicación.

- **Soporte:** Escaso, recibe actualizaciones cada varios meses e inicialmente parte de una versión antigua de Mapbox por lo que está altamente desfasada y lo estará más en el futuro. La documentación toma como base aquella de Mapbox y en reiteradas ocasiones no está modificada para MapLibre.
- **Precio:** Completamente gratuito.

Comparativa

Estas tres opciones coinciden en varias de las necesidades de la aplicación, permiten visualizar marcadores y polígonos, localizar al usuario, personalizar mapas y ver mapas *offline*. Existen otras opciones que no fueron tenidas en cuenta porque fallaban en alguno de estos aspectos que se consideraron críticos.

Durante el desarrollo se mantuvieron distintas ideas activas que se descartaron posteriormente, una de ellas fue la creación y edición de potreros dentro de la aplicación. Esta tarea requeriría de funcionalidades más avanzadas y herramientas que MapLibre no tenía al momento de realizar la elección. Fue por esto que se desarrollaron tanto RNMaps como Mapbox en simultáneo para verificar que fueran capaces de cumplir. Con el desarrollo de esta funcionalidad altamente avanzado, el cliente decidió descartar la idea, confiando la potestad de crear potreros únicamente a la aplicación web. Así, al no necesitar crear potreros se pudo elegir MapLibre, ya que sus capacidades eran suficientes para las otras necesidades. Por último, la documentación incompleta o errónea fue un factor que consumió tiempo, pero no representó un impedimento para la implementación de la librería.

Estadísticas

Objetivos

Luego de realizar mediciones en distintas ubicaciones geográficas, fechas y horarios, surgió la necesidad de presentar esa información de forma cómoda y entendible. Se buscó con la sección de estadísticas mostrar las mediciones realizadas en un periodo, potrero y/o sector específico, como así también su valor promedio y su peso estimado.

Librería

Se necesitó únicamente una librería para la selección de fechas para los periodos. La forma de presentación usual es como calendario o como rueda deslizante. Frente a la implementación de ambas opciones en el sistema, se consideró más apropiado la segunda, ya que no solo tenía mejor rendimiento, sino que también era más fácil de utilizar que su alternativa.

Presentación de estadísticas

Los datos a mostrar son extraídos de la base de datos aplicando los filtros necesarios. Para evitar las consultas excesivas se implementó un sistema que despacha una señal de actualización mediante Redux para notificar que se debe repetir la consulta. Esto se dará en los casos que se agregue una nueva medición o se modifique alguno de los filtros activos.

Esta pantalla se considera un punto de partida para la visualización de datos, se planteó para facilitar la implementación de nuevas estadísticas a futuro. Actualmente, se presenta la cantidad de mediciones a analizar, el promedio entre ellas, el periodo de muestra y el peso estimado utilizando la curva de calibración correspondiente.

El cálculo del peso requiere que se elija una calibración de una lista, estas calibraciones serán o bien por factor, tienen su polinomio definido de forma explícita, o bien definido por mediciones, necesitando este último que ya hayan sido cargadas al servidor *cloud*.

Filtros disponibles

- **Periodo:** Es el filtro más simple, cuando se realiza una medición esta se guarda junto a la hora en la que fue tomada. Se pueden definir entonces un límite superior e inferior que permitan discernir qué mediciones caen dentro de esos límites utilizando el valor guardado. Este sistema requiere de definir fechas, en un dispositivo móvil esto puede resultar engorroso, por eso se decidió utilizar un sistema de rueda deslizante por sobre un calendario tradicional. Sin embargo, esto sigue siendo tedioso para su uso continuo. Para solventar este problema se crearon periodos de selección rápida, como puede ser 24 horas, 36 horas o 1 semana hacia atrás respecto de la fecha actual. Entonces, al presionar un botón ya se puede aplicar un filtro rápido.

- **Potrero:** Para seleccionar un filtro por potrero se despliega una lista con cada potrero identificado por su nombre, como se ve en la pantalla de Mapas. Al seleccionar uno de ellos se filtrarán las mediciones que caigan fuera del polígono correspondiente. De esta forma, se mostrarán únicamente las que hayan sido tomadas dentro de dicha región. Esta verificación se hace en forma local en el dispositivo y se realiza con la ayuda de la librería Turf (@turf/turf), misma que se usó para los filtros en la pantalla de Mapas, en la funcionalidad descartada de crear potreros y además en el servidor web, lo cual mejora la consistencia entre los dos sistemas.

- **Sector:** El filtrado de sectores representa un desafío en sí mismo, ya que los sectores no tenían una implementación fija y se implementaron con límites temporales. Al presionar el botón de grabar potrero se almacena el momento exacto, que será considerado como límite temporal inferior. Luego, cuando se vuelve a presionar el botón, se cerrará el sector y el periodo de grabación quedará definido al establecer la hora de cierre como límite superior. Este funcionamiento reduce significativamente el tiempo necesario para filtrar las mediciones, como contrapartida no es posible diferenciar un sector de otro a la hora de mostrarlos en la lista de filtros. Para remediar esta problemática se plantearon distintas opciones.

  En un principio se consideró asignar un nombre aleatorio a los sectores creados, aunque esto podría resultar confuso para los usuarios. Una segunda opción contemplaba agregar una funcionalidad para elegir el sector antes de iniciar el proceso de grabación. Sin embargo, dado que esta opción requeriría una extensión adicional en el tiempo de

desarrollo, no resultó viable según el cronograma establecido.

Finalmente, se optó por la opción más sencilla: mostrar los límites temporales superior e inferior en la lista para filtrar. Esto resulta comprensible para cualquier usuario, ya que se trabaja únicamente con fechas y horarios. Se prestó especial atención a que estas fechas fueran legibles en cualquier dispositivo. Además, se utilizó un menú idéntico al empleado para filtrar potreros, garantizando así una consistencia que reduce la curva de aprendizaje del usuario.

Módulo *cloud*

La decisión de implementar una API RESTful, un estándar bien establecido en el desarrollo web, fue impulsada por su adecuación a los requisitos del proyecto. Dada la naturaleza del servidor siendo este web, orientado a la escalabilidad mediante una arquitectura de microservicios y los datos relativamente sencillos que maneja, se eligió una interfaz RESTful sobre GraphQL. Este último, aunque poderoso para operaciones de datos complejas, se consideró innecesario para las estructuras de datos más simples necesarias en este proyecto.

Para el intercambio de datos entre el cliente y el servidor, se desarrolló una serie de *endpoints*. Estos fueron diseñados teniendo en cuenta que la mayor parte del uso de la *app* se hará de manera *offline*. Por lo tanto, para asegurar que la aplicación pueda funcionar de manera efectiva sin conexión. Una parte significativa de la información mostrada por la aplicación se calcula con base en datos previamente obtenidos y almacenados, lo que permite un uso sin interrupciones cuando no se cuente con acceso a internet.

La aplicación intercambia información sobre potreros con el servidor, incluyendo el nombre y delimitaciones geográficas específicas de cada uno. Datos recogidos por el pasturómetro, que incluyen metadatos como marcas de tiempo y ubicaciones, son enviados al servidor para su procesamiento. Por último, se realizan y transmiten calibraciones basadas en estas mediciones. Cada calibración incorpora un conjunto de mediciones, identificadas individualmente por un ID. Fuera de la aplicación, a estas mediciones se les asignará un peso específico, con el cual se calculará un polinomio. Este polinomio, una vez generado, se envía desde el servidor a la aplicación, donde se utiliza para realizar análisis estadísticos al convertir con estos las mediciones de altura a kilogramos de pasto seco por hectárea.

La transmisión de datos entre la aplicación y el servidor se realiza en lotes y ocurre a intervalos establecidos, actualmente cada diez minutos. Este intervalo es configurable para adaptarse a diferentes necesidades operativas. Durante estos periodos, la aplicación intenta enviar todos los datos marcados como 'no enviados'. Cuando el proceso inicia, marca los datos en un estado intermedio, si el intento falla, los datos se marcan nuevamente como 'no enviados' en la base de datos local; si tiene éxito, se marcan como 'enviados'.

Los usuarios también tienen la capacidad de sincronizar manualmente en cualquier momento, una función que se ha mostrado especialmente útil en determinadas situaciones. Y además del botón de sincronización manual, se cuenta con una indicación del tiempo en el que se realizó la última actualización exitosa por el sistema o usuario.

Notificaciones

En una instancia avanzada de desarrollo se consideró e implementó un sistema de notificaciones utilizando las capacidades nativas de iOS y Android a nivel de sistema operativo, como también creando componentes a nivel aplicación para su manejo de forma coherente con su diseño. Estas fueron de sumo provecho para informar al usuario de eventos, incluso con la aplicación en segundo plano.

Mediciones

El módulo aborda las vistas, controladores y modelos relacionados. Esto incluye funciones para realizar consultas de mediciones en un período específico, la eliminación y la lógica involucrada en el proceso de medición.

En el proceso de medición, una vez que el pasturómetro ha transmitido los datos, se lleva a cabo un análisis para detectar posibles errores y también prepara las mediciones para su posterior envío. El dato recibido del dispositivo no es el mismo que el mandado al servidor posteriormente.

Se reciben N mediciones, una por cada sensor del pasturómetro, y esta cantidad N entre otros datos. A partir del conjunto de mediciones que representan un evento de medida se filtran anomalías. Es usual que en pastizales altos hojas pasen por encima del plato dando así alturas

erróneas al sensor que intercepten. Para afrontar esto, el algoritmo que se implementó fue el siguiente:

- Se calcula la mediana de las mediciones.
- Se filtran las mediciones que varían por encima de un umbral determinado en relación con la mediana. Esto se debe a que la geometría del plato puede causar variaciones en la medición de un sensor hasta cierto punto, pero si hay una diferencia abrupta, probablemente se deba a una hoja o un mal funcionamiento.
- Se verifica cuántos sensores quedan después de la filtración.
    - Si quedan menos de un número determinado, se considera que no hay suficiente consenso de sensores para garantizar que la medición sea válida. En este caso, la medición no se carga y se notifica al usuario.
    - Si se filtró más de un sensor, pero no suficientes para caer en el caso anterior, se notifica al usuario que hay sensores tapados, pero aun así se carga la medición.
    - En caso de que todos los sensores entreguen mediciones válidas, se carga la medición sin lanzar ninguna notificación al usuario.

Calibraciones

Las calibraciones forman parte del eje central de la aplicación, estas pueden representar un polinomio o un conjunto de mediciones para enviar al servidor. Frente a esta última opción, el sistema *cloud* será quien se encargue de calcular el polinomio y entregarlo a la *app*.
El polinomio es utilizado tanto en la aplicación móvil como web para convertir de cm de pasto medidos a kg de pasto seco por hectárea.

Este módulo involucra la creación del modelo, los controladores y las vistas para poder crear, editar, listar y borrar calibraciones de ambos tipos.

Wireframes

El grupo no estaba familiarizado con el desarrollo de aplicaciones móviles, la metodología que se halló en numerosas fuentes fue la utilización de “maquetas” llamadas *wireframes* para el proceso de análisis y a su vez de diseño.

Un *wireframe* es una representación visual en escala de grises de la estructura y funcionalidad de cada pantalla de una aplicación móvil. Los *wireframes* se utilizan al comienzo del proceso de desarrollo para establecer la estructura básica de una página antes de añadir el diseño visual y el contenido. Muestra la estructura y el diseño sin entrar en detalles estéticos específicos. Proporcionan una visión clara para los interesados, los equipos de desarrollo, los diseñadores y todos los demás asociados al proyecto.

Al utilizarlos se asegura que la aplicación se construya de acuerdo con los objetivos. Además, se establecen expectativas sobre cómo se implementarán las características, mostrando cómo funcionarán, dónde se ubicarán y cuánto beneficio aportarán. Una característica puede ser eliminada fácilmente si no se ajusta a los objetivos.

También permiten centrarse en la usabilidad al proporcionar una mirada a los flujos de usuario, la facilidad de uso, la navegación y la ubicación de las características. Los *wireframes* ayudan a identificar los defectos de diseño o las características y muestran que tan bien fluye desde la perspectiva del usuario.

También facilita notar qué características serían útiles de implementar en una futura instancia, modificando el diseño a partir de esto.

Existen distintas definiciones de *wireframes* que van desde las más abstractas a las más específicas. En el caso de este proyecto, teniendo en cuenta el nivel de detalle con el que se trabajó a la hora de bocetar, se optó por un *wireframe* de flujo de usuario.

Los *wireframes* de flujo de usuario logran mostrar cómo un usuario de la aplicación se moverá por el contenido de pantalla a pantalla. Esto se documentó mediante la utilización de rutas, que a su vez a futuro serían utilizadas para el diagrama de jerarquía de pantallas dentro de la aplicación.

Interfaz de usuario y experiencia de usuario

La interfaz de usuario o User Interface (UI) se define como el conjunto de elementos visuales que se presentarán ante el usuario en una aplicación visual. Por otro lado, la experiencia de usuario o User Experience (UX) engloba aquello que una persona siente mientras utiliza un producto o servicio digital y prioriza la interacción con el usuario por sobre el diseño visual.

El grupo carecía de experiencia en diseño de UI/UX, esto presentó un reto que requirió una profundización en el tema. Se encontró una bibliografía rica otorgada por Mozilla en su curso de diseño para desarrolladores y a partir de él se conoció la existencia de los denominados sistemas de diseño, entre otros estándares.

Estos sistemas de diseño definen pautas que se repetirán a lo largo de toda la aplicación, dándole una identidad y a su vez coherencia en sus patrones. Este tipo de detalles hacen que a simple vista una pantalla se pueda ver bella y profesional o malograda.

Al no tener pericia ni tampoco justificarse aportar tantas horas al diseño gráfico, se optó por una interfaz que sea simple para así cometer la menor cantidad de errores estéticos posibles.

La UX debe tener en cuenta el contexto particular de la aplicación y los distintos tipos de usuarios que la utilizarán. Lo primero que es importante notar es el entorno, este será de uso diurno a la intemperie donde existirá una significativa exposición a la luz. También los usuarios potenciales pueden presentar dificultades para comprender textos en pantalla. Se consideraron estos factores y más para obtener una UI que ofrezca la mejor UX posible.

Paleta de colores

Se ha optado por una paleta de colores claros de alto contraste, con un tono primario verde que es fácilmente distinguible por el ojo humano, es estéticamente agradable y también está intrínsecamente relacionado con el contexto de la aplicación que es la pastura. Se ha decidido no implementar un modo oscuro, ya que se prevé que la aplicación se utilizará principalmente al aire libre durante horarios laborales.

La paleta de colores también se mantiene minimalista para simplificar el diseño y evitar posibles errores, especialmente dado el nivel de experiencia limitado en diseño visual.

Dispositivos

Se desarrolló el sistema para ser fácilmente utilizado en dispositivos de gama baja, teniendo en cuenta las resoluciones limitadas que estos suelen presentar y también la amplia variedad de tamaños de pantallas en el mercado.

Facilidad de uso

La aplicación fue pensada para ser usada de manera intuitiva, por lo que se minimizaron los textos para favorecer la utilización de iconos. Aunque también pueden ser utilizados en conjunto para clarificar escenarios complejos. Se tuvo en cuenta la necesidad de un texto fácilmente legible a lo largo de las múltiples pantallas. Para lograrlo se optó por limitar la cantidad de información en cada una de ellas y así lograr una presentación limpia de los datos hacia el usuario.

Librerías

Las principales librerías que se encontraban a la hora de iniciar el proyecto eran[25]:

- Native-Base[26]
- React-Native-Paper[27]

Se eligió Native-Base por sobre React-Native-Paper en el inicio debido a que se vio como una ventaja su sistema de diseño más complejo y rico. Posteriormente, debido a un trabajo en simultáneo donde se utilizó React-Native-Paper, se supo que esta no había sido la decisión más óptima.

Si bien React-Native-Paper no tiene el sistema de diseño de Native-Base, permite declarar una paleta de colores, fuentes y ciertas constantes de estilos. Esto en conjunto con el diseño por componentes ofrecido por React es más que suficiente para crear de manera eficiente y más práctica un sistema de diseño propio. Sumado a esto, a finales del desarrollo de la aplicación Pastech se daba el mensaje vía la página oficial de la librería que Native-Base era discontinuada.

Interfaz lograda

Teniendo en cuenta todos los detalles presentados con anterioridad, se llegó a un sistema de diseño consistente a lo largo de toda la aplicación móvil Pastech. Este diseño buscó armonizar tanto con la página web del sistema de monitoreo (https://system.pastech.com.ar), como con el *branding* que se fue gestando a la par de la aplicación.

Figura 14 - Pantalla de Inicio y grabación de sectores

Figura 15 - Principales pantallas de sección Calibración

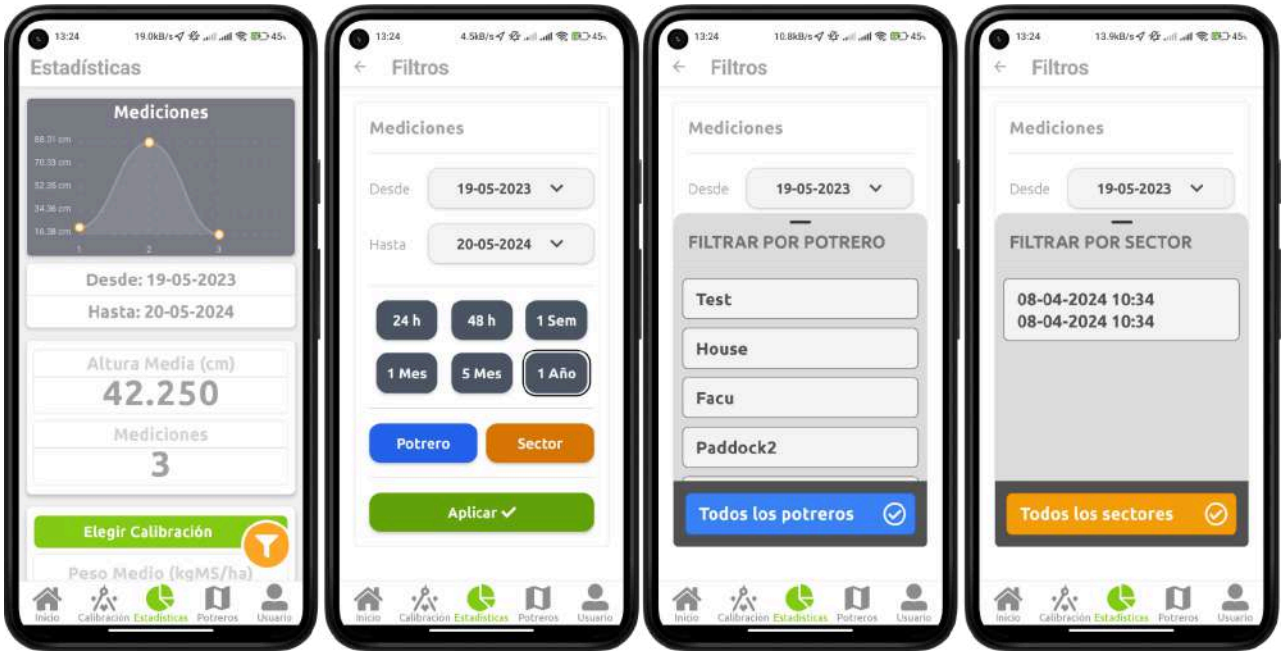


Figura 16 - Principales pantallas de sección Estadísticas

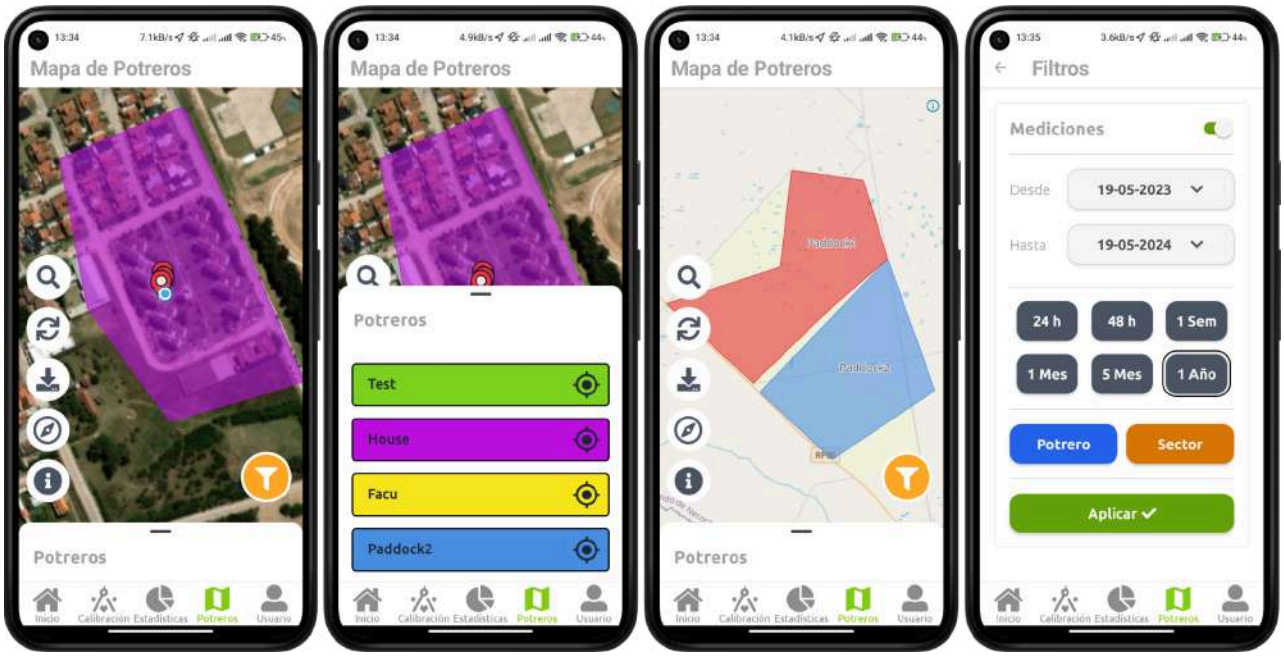


Figura 17 - Principales pantallas de sección Potreros

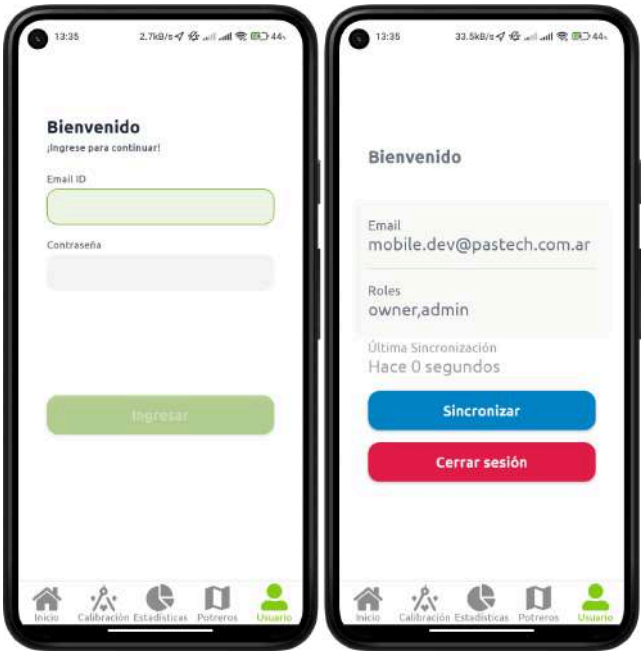


Figura 18 - Pantalla de inicio de sesión y sincronización

Figura 19 - Notificación dentro de la aplicación

Gestión de versionado

En la gestión de código de este proyecto, se optó por la utilización de GIT, una decisión fundamental en cualquier proyecto de desarrollo de *software*. GIT ofrece una serie de ventajas notables, siendo una de las más destacadas su capacidad de ramificación. Esta funcionalidad permitió la creación de ramas específicas para llevar a cabo el trabajo en paralelo en los diversos módulos de la aplicación.

El proceso de desarrollo se organizó en varias ramas, cada una dedicada a un componente o módulo en particular. Esto posibilitó que los miembros del equipo de desarrollo pudieran colaborar simultáneamente en diferentes partes del proyecto sin interferir entre sí. Cada rama representó una unidad de trabajo separada y fueron fusionadas en la rama principal, conocida como “Main”, una vez que se completaron y probaron satisfactoriamente.

La estrategia de ramificación en GIT no solo facilitó la colaboración efectiva, sino que también garantizó un control preciso del historial de versiones, lo que resultó esencial para el seguimiento y la documentación de los cambios en el código a lo largo del ciclo de desarrollo del proyecto.

Validación

Durante el desarrollo, se llevó a cabo un proceso de validación del producto mediante pruebas exhaustivas de cada funcionalidad a medida que se iban creando. Posteriormente, se realizaron pruebas de integración sobre los casos de uso una vez que estos estuvieron disponibles.

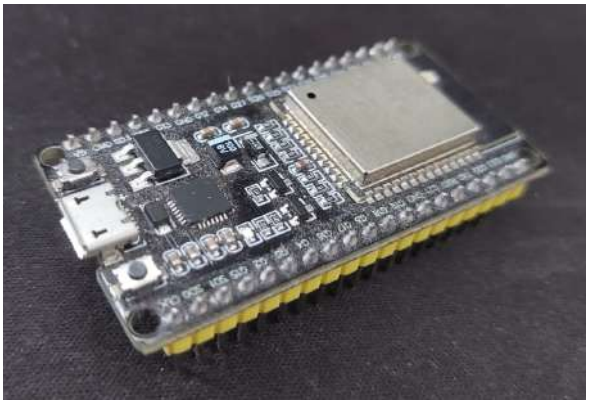


Figura 20 - SoC(System on Chip) ESP32 utilizado a lo largo del desarrollo

Para garantizar la integración con el dispositivo, se empleó un prototipo de pasturómetro que emulaba su comportamiento de medición. Este prototipo, equipado con el mismo *hardware* que la

versión definitiva (ESP32), fue utilizado a lo largo de todo el desarrollo para realizar pruebas que permitieron identificar y corregir errores antes de llevar a cabo las pruebas de campo con el equipamiento real.

Figura 21 - Pasturómetro Pastech

Sin embargo, durante una de las pruebas de campo, se descubrió un error de interpretación en los datos recibidos que no había sido evidente en las simulaciones. A pesar de ello, este error fue rápidamente identificado y corregido.

En cuanto a las pruebas de integración con el sistema en la nube, se trabajó en estrecha colaboración con el otro equipo de desarrollo. Se empleó la herramienta Postman para detectar si se trataban errores propios de la aplicación, de la interpretación de la API, o del servidor. Esta herramienta a su vez facilitó la comunicación entre equipos y permitió identificar y corregir los tres tipos de errores mencionados anteriormente.

Seguridad de los datos

Comunicaciones

Existen dos instancias de comunicación de la aplicación con entidades externas: a través de Bluetooth Low Energy con el pasturómetro y mediante HTTPS con el servidor de *backend*.

En el caso de la comunicación con el pasturómetro, se evaluó la necesidad de cifrar los datos. La aplicación tiene la capacidad de utilizar una encriptación estándar con la librería elegida, sin

necesidad de configuraciones adicionales, pero esta función requeriría cambios en el *firmware* del pasturómetro. La conclusión fue que el cifrado no era justificado, ya que los datos transmitidos no son sensibles; se trata únicamente de mediciones de pasto, y adquirir este dato directamente sería más conveniente que interceptarlo. En la aplicación, los mensajes se sanitizan a números. No se consideró justificable implementar un mecanismo para evitar ataques de intermediario (MiTM), dado que los datos no son sensibles y la cobertura limitada del Bluetooth reduce el riesgo.

En lo que respecta a la comunicación con el Sistema *Cloud* esta se lleva a cabo de manera segura a través del protocolo HTTPS, estableciendo un canal cifrado para la transmisión de datos. La comunicación se realiza mediante una API RESTful y en cada petición se envía un *token* de autenticación en el encabezado (*header*) de la solicitud, una vez que el usuario ha iniciado sesión correctamente. Este *token* sirve como mecanismo de seguridad para validar la identidad del usuario y autorizar el acceso a los recursos solicitados en la API.

Datos locales

Al iniciar sesión, la aplicación envía tanto el correo como la contraseña al servidor, que responde proporcionando un *access token* y un *refresh token* para su uso en las subsiguientes solicitudes. El *refresh token* junto con otros datos de sesión enviados por el servidor se almacenan persistiendo la sesión. Estas credenciales tienen una validez limitada, así que la aplicación las refresca periódicamente mientras está abierta, postergando su fecha de caducidad. En el caso de que se cierre por completo la aplicación o el dispositivo móvil se apague, este refresco no se podrá concretar y los datos de acceso podrán vencer, en cuyo caso se deberá volver a iniciar sesión.

Por último, el acceso a la base de datos local no cuenta con encriptación. Esta instancia de BD, en la que se almacenan las mediciones, calibraciones, y potreros, es únicamente accesible por el dispositivo donde está instalada la aplicación. Dado que no se almacenan datos sensibles y esta base de datos solo está expuesta al mismo dispositivo no se consideró el peligro de su filtrado como justificación al uso de opciones pagas de base de datos que posibilitan la encriptación, ni tampoco realizarla por separado debido a las complejidades y desventajas que conllevan.

Memorias del proyecto

Trabajo conjunto

Durante el proyecto se desarrolló una aplicación móvil, mientras que otro equipo, integrado por los estudiantes Tobias Demoor y Leopoldo Lening Celaya, se encargó de implementar la API REST. Desde el inicio, ambas partes acordaron una definición clara y detallada de la API. Aunque no fue necesario modificarla posteriormente sí surgieron malentendidos durante su implementación en un principio.

La colaboración entre ambos equipos fue directa, con un enfoque en la resolución rápida de problemas. Si bien cada equipo tenía enfoques y responsabilidades diferentes, el canal de comunicación se mantuvo fluido para gestionar imprevistos sin afectar el progreso global del proyecto.

Un aspecto crítico fue que el equipo encargado de la API tenía previsto finalizar su trabajo en junio, mientras que el equipo de la aplicación móvil continuaría hasta septiembre. Esta diferencia en los plazos hizo imprescindible definir correctamente la API desde el principio, ya que no habría margen para solicitar cambios una vez concluido el desarrollo de la API.

Para mitigar posibles inconvenientes, se priorizó el cierre de todas las etapas dependientes de la API antes de su finalización. Además, se diseñaron algunos módulos del sistema de forma desacoplada, lo que permitió avanzar en el desarrollo de la aplicación móvil sin requerir ajustes adicionales del otro equipo.

Metodología aplicada

En la elección del método se priorizó el dinamismo e inmediatez que se presenta en un desarrollo móvil. Al emplear un enfoque incremental se logró una distinción marcada de las etapas del proyecto, como así también de los distintos módulos que se debían atender. Resultó ser efectivo en la separación de dichos módulos y la planificación de su desarrollo, ya que se evitaron en gran medida los conflictos en el desarrollo paralelo al dotarlos de cierta autonomía.

El equipo de desarrollo adoptó correctamente la metodología, que permitió la división de tareas con suma facilidad. Además, abrió la posibilidad de presentar las iteraciones al cliente sin necesidad de llevar todos los módulos el mismo nivel de avance.

Como se presentó anteriormente, GitHub fue el sistema de gestión de versionado elegido. La experiencia con el enfoque adoptado fue mayormente positiva, al tener un flujo de trabajo que priorizó integrar los cambios en la rama principal se logró reflejar rápidamente los cambios para el resto del equipo y se evitaron grandes conflictos en etapas avanzadas. Sin embargo, surgieron algunos imprevistos debido a un testeo no lo suficientemente exhaustivo. Al mantener una comunicación estable, el conjunto del equipo los resolvió rápidamente. El mayor inconveniente ocurrió durante una prueba de campo, en la que cierta funcionalidad fue omitida de la *build* presentada. El equipo preparado para hacer frente a alguna eventualidad utilizó una versión de desarrollo con un equipo portátil y se realizaron las pruebas pertinentes.

Gestión del tiempo

Como se supo presentar en el cronograma, el periodo planteado originalmente no terminó reflejando la fecha real de finalización del proyecto. Esta planificación se planteó en términos ideales donde se podrían emplear determinadas horas por semana al proyecto. Sin embargo, las circunstancias particulares como exámenes, presentaciones, horario laboral, viajes o necesidad de terceros llevó a la extensión en el tiempo del periodo de desarrollo. Aun extendiéndose este margen, el tiempo de desarrollo total fue bastante similar al visto en el cronograma.

Demoras particulares en el desarrollo del proyecto

Algunos casos tuvieron un mayor peso sobre la demora final y requirieron un accionar especial que se detalla.

- **React Native:** El propio proceso de adopción del *framework* llevó más tiempo del esperado, ya que el salto desde React no fue completamente directo. Para solventar rápidamente esta problemática, el equipo enfocó sus esfuerzos en un proceso de aprendizaje conjunto. Finalizado el proceso de aprendizaje, el equipo pudo separarse nuevamente para realizar tareas paralelas.

- **Mapas:** La necesidad de una pantalla de creación de potreros mantuvo la problemática del proveedor de mapas activa. Esto llevó a un desarrollo paralelo de múltiples proveedores. Cuando finalmente se decidió descartar la creación de potreros, se definió el proveedor gratuito MapLibre y el desarrollo continuó de forma fluida. Como este hecho

escapaba a las decisiones del equipo, se decidió mantener conjuntamente las distintas opciones, ya que empezar desde cero hubiera hecho la demora aún más pronunciada.

- **Integración de nuevos requerimientos:** El desarrollo de un *software* suele ser un proceso dinámico que requiere revisitar las distintas etapas varias veces, así se debieron atender los requerimientos que surgieron durante el desarrollo. Algunos cambios pequeños de usabilidad, como la incorporación de botones para la aplicación de filtros rápidos, y otros que requirieron más tiempo, como la incorporación del sistema de sectores, que no había sido previamente introducido en las reuniones iniciales. La modularización del desarrollo permitió la implementación de estos requerimientos sin tanta complejidad.
- **Trabajo remoto**: El equipo realizó gran parte del desarrollo separado en distintas localidades. Este hecho no resultó un gran problema en el desarrollo, debido a las herramientas disponibles para el trabajo remoto, pero sí que trajo algunas dificultades. Se utilizó un dispositivo con sensores idénticos al del pasturómetro para probar la recepción Bluetooth con la aplicación. Dicho elemento estaba en posesión del equipo, pero al solo contar con una unidad dificultó el proceso de testeo. Para solventar al máximo esta problemática se separaron las tareas de tal forma que se necesitó al mínimo el dispositivo físico. Cabe aclarar que es posible emular la funcionalidad del dispositivo, pero no es un fiel reflejo de su funcionamiento real, según se probó.

Estimación original de tiempos

Inicio 01/01/2023 - Fin 22/06/2023 - Duración 25 semanas	enero	febrero	marzo	abril	mayo	junio	julio	agosto	septiembre	octubre	noviembre	diciembre	enero	febrero	marzo	abril	Total
ANÁLISIS	70																70
DISEÑO		120															120
IMPLEMENTACIÓN y VALIDACIÓN					200												200
CIERRE						80											80
																	470

Tiempos reales de ejecución

Inicio 01/01/2023 - Fin 20/3/2024 - Duración 64 semanas	enero	febrero	marzo	abril	mayo	junio	julio	agosto	septiembre	octubre	noviembre	diciembre	enero	febrero	marzo	abril	Total
ANÁLISIS	48	4	6														58
DISEÑO		44	16														60
IMPLEMENTACIÓN y VALIDACIÓN			41		82	50	71	27	41	13							325
CIERRE											13	45	34	16	16	20	32	176
																	619

Figura 21 - Cuadros comparativos de tiempos estimados y reales

Para el cálculo inicial del tiempo se estimó un promedio de 80 horas mensuales en total entre ambos integrantes. Esta estimación se fundamenta en una dedicación diaria de lunes a viernes de 2 horas por cada integrante.

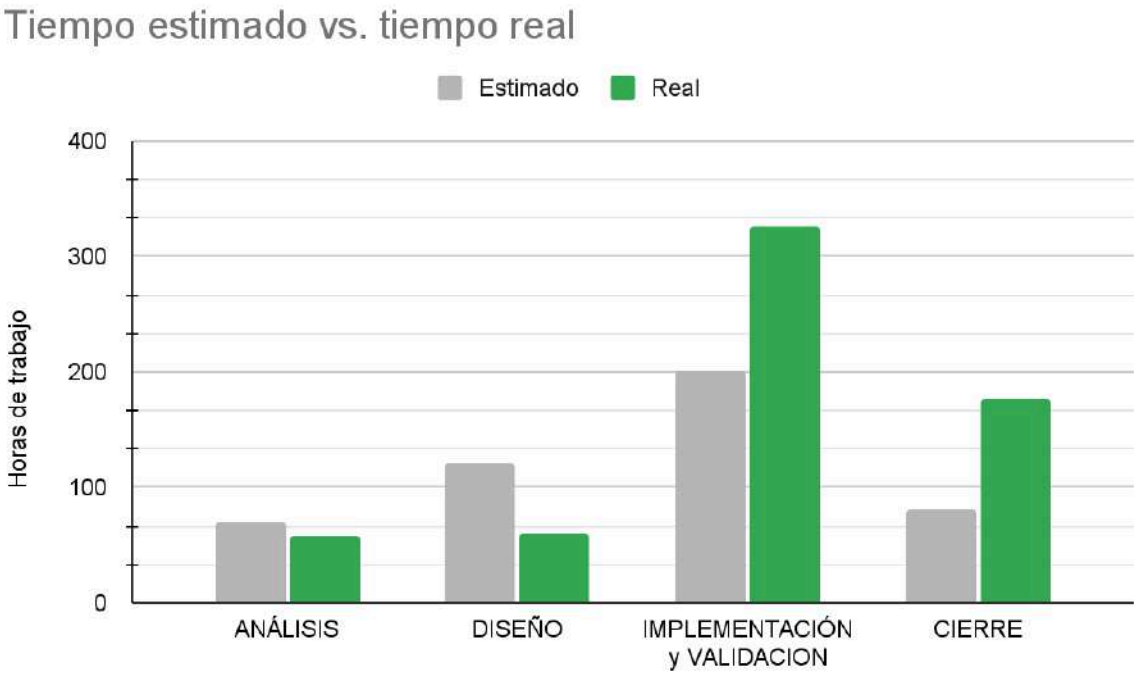


Figura 22 - Cuadro comparativo de horas estimadas vs. reales

Análisis

La fase inicial del análisis comenzó con la recopilación de datos, se trataba de un dominio inexplorado para los miembros implicados. Se mantuvieron charlas con el cliente para relevar la información necesaria, mediante las cuales se encontraron y definieron los requerimientos iniciales. El análisis fue revisitado al momento de integrar nuevos requerimientos, ya que se necesitó tener en cuenta la repercusión de estas modificaciones desde las bases del proyecto.

A esta etapa le bastó una dedicación menor a la esperada, aun así la duración de la misma no disminuyó debido a que parte de esta se define por la posibilidad de reunirse con los referentes funcionales para relevar requerimientos y posteriormente validarlos. Se había estimado la utilización de 70 horas de trabajo, de las cuales terminaron siendo necesarias 58 horas.

Hay dos instancias que se encuentran fuera del primer mes, la primera abarcó la validación y retroalimentación de ciertos diseños, mientras que la segunda consistió en acordar una

interfaz común para interactuar con el sistema *cloud*. Esto se hizo de manera conjunta con el equipo de desarrollo de este sistema.

Diseño

Esta etapa se definió en un marco de desarrollo móvil, esto implicó usar herramientas específicas para su diseño. Los aspectos básicos que se plantearon derivaron en una comprensión interna más detallada del problema, se plantearon diagramas de componentes y diagramas de flujo. Sin embargo, aprovechando la naturaleza gráfica y palpable de las aplicaciones móviles, se desarrollaron prototipos de rápida elaboración y funcionalidad mínima llamados *wireframes*. Al plantear las distintas pantallas y el flujo entre ellas, permitió mostrar al cliente fácilmente las conclusiones que se habían alcanzado. Esta dinámica resultó ser efectiva y sencilla para ambas partes.

Los cambios pertinentes a los nuevos requerimientos trajeron consigo la modificación de varios aspectos de diseño. El quitar la pantalla de creación de potreros se reflejó como la eliminación del botón ‘Crear Potrero’ y el botón ‘Editar Potrero’ de los *wireframes*. Mientras que la adición de los sectores significó modificar el *wireframe* de la pantalla inicial.

Parte importante de esta etapa se trató de la elección de las tecnologías a utilizar, se trataba de un desafío nuevo para los integrantes y no contaban con un gran conocimiento del panorama de desarrollo móvil previo al proyecto. Por lo tanto, se sobreestimó la etapa, finalmente se vio que la mayoría de elecciones, dadas las condiciones del proyecto, tendieron a reducirse a no más de un par de candidatos. Esto por la experiencia previa en desarrollo web, donde el panorama de opciones puede ser abrumador, resultó inesperado para los integrantes. Lo que concluyó en un tiempo menor de diseño al esperado, utilizándose finalmente 60 horas frente a las 120 horas estimadas.

Desarrollo y validación

Después de dividir la aplicación en módulos, se procedió con su implementación. En esta etapa, cada integrante del equipo comenzó a trabajar de manera independiente, enfocándose en módulos específicos. Sin embargo, debido a problemas personales, no se pudo avanzar simultáneamente como estaba planeado. Lo que originalmente debía tomar 2

meses de trabajo conjunto en distintos módulos se extendió a 5 meses. Uno de los integrantes inició e implementó sus módulos primero, mientras que el otro continuó con los suyos cuando fue posible.

Además, en el mes de abril, ninguno de los integrantes pudo avanzar debido a la suma de exámenes parciales sobre sus respectivas rutinas.

A estos desafíos se sumaron problemas adicionales, como la aparición de nuevos requerimientos y errores durante las validaciones de campo, que requirieron tiempo adicional para su resolución y acuerdo de nuevos encuentros para su revalidación. Como resultado, esto produjo una diferencia de 125 horas adicionales de trabajo sobre las 200 horas estimadas inicialmente.

Atraso en el informe

El mayor retraso con una diferencia de un orden de magnitud en prolongación respecto a lo estimado fue la etapa de escritura del informe. Lo particular de esta etapa respecto a las otras fue que a los ojos de los integrantes la demora no afectaba a terceros, a diferencia del funcionamiento de la aplicación que si afectaba al cliente de forma directa, ya que la salida tardía del producto podría ser decisoria en cuanto la viabilidad del proyecto Pastech. Por otro lado, la tarea de escritura dista mucho del proceso de desarrollo, cuyo objetivo es crear un *software* palpable y que puede proveer un *feedback* directo al realizar los *tests* o pruebas correspondientes.

El grupo no se conocía previamente del proyecto y siendo el caso que tampoco se encontraban en la misma ciudad, no se pudo generar un lazo como sí hubiera ocurrido en el caso contrario. Esto afectó tanto a la moral como a la dinámica entre los integrantes, y esto llevó a una dificultad para brindar motivación o presión mutua a la hora de cumplir plazos.
Por otra parte, si bien existieron motivos personales para cierto grado de retraso, la demora se debió principalmente a la caída en un círculo vicioso de aplazamiento a la hora de escritura, que una vez ya atrasados se llegó a creer que *poco podría hacer de diferencia un día más* en la mente los integrantes.

En retrospectiva se reconoce una falla crítica en este aspecto. Si bien en la etapa de desarrollo se cumplieron en la medida lo posible los plazos planteados, se vio que la falta de

una definición más granular a la hora de la escritura del informe sumado a la falta de motivación personal, poca experiencia en este aspecto, prolongaron esta tarea de 3 semanas, que fue una aproximación optimista en retrospectiva, hasta 7 meses.

Las horas requeridas para esta etapa fueron 176, muy por encima de las 80 horas estimadas. Si se hubiera planteado este valor inicialmente, utilizando el criterio originalmente aplicado, representaría al menos 2 meses de trabajo.

Las enseñanzas que esto dejó al equipo son las siguientes:

- Redefinir plazos a la hora de atrasos por más abruptos que estos sean.
- Tomar con seriedad la tarea de escritura de informes, atribuirle el tiempo necesario y planificarlos con sus subtareas en caso de ser necesarios. Brindarles la misma prioridad respecto a las demás tareas.
- A la hora de realizar tareas que no resulten placenteras, hacer mayor foco en el seguimiento de tiempos para no caer en círculos viciosos.
- No considerar únicamente las dificultades técnicas a la hora de afrontar un proyecto, sino también tener en cuenta las de índole personal, en este caso la motivación, y así poder generar estrategias para afrontarlas.

Trabajos futuros

Requerimientos descartados

Durante la realización del proyecto se identificaron requerimientos que posteriormente fueron revisados y algunos de ellos descartados. Dependientes del contexto particular, ya sea por el alcance del proyecto o la complejidad de este, estos requerimientos no fueron tenidos en cuenta. Sin embargo, esto no quita que fueran compatibles con la aplicación o que no sean necesarios en el futuro. Por esto, inicialmente sea podrían retomar:

Creación de potreros

Los primeros datos e ideas recabados apuntaban a la posibilidad de una aplicación móvil con independencia que no requiera del sistema *cloud*. Surgieron inconvenientes organizacionales y administrativos que escapan al proyecto. Finalmente, la idea fue descartada en una etapa avanzada, en este punto la característica era funcional y entraba en las primeras etapas de testeo. Sin embargo, no se utilizó y quedó desfasada respecto al resto del proyecto. Por lo tanto, no debería representar gran problema integrarla nuevamente si se requiere en el futuro.

Calcular calibraciones *in-app*

Tener una aplicación independiente también requeriría calcular las curvas aproximantes para las calibraciones mediante mediciones. En este proyecto se optó por enviar las mediciones al sistema *cloud* y esperar su respuesta ya calculada. En caso de avanzar con esta idea se deberá confirmar que exista una consistencia entre los datos calculados por la *app* y el *backend*, preferiblemente que se utilice el mismo método de cálculo.

Sincronizar sectores con la nube

Los sectores son una característica que surgió en etapas avanzadas del desarrollo, por lo que no existen en el sistema *cloud*. Si se decide agregar, la *app* también deberá actualizarse para poder llevar a cabo una sincronización. Se podrían descargar los sectores o subirlos de ser necesario.

Sectores geométricos

En la actualidad los sectores delimitan una franja horaria en la que se tomaron las mediciones. Sin embargo, se concibió originalmente como una región específica en el espacio. Se podría agregar la posibilidad de agregar sectores y definirlos geométricamente mediante coordenadas que representan un polígono. Para que esto funcione, la *app* deberá integrar sectores geométricos que se podrán visualizar en la pantalla de potreros. Por último, existe la posibilidad de definir el sector de una medición automáticamente dependiendo de la posición dónde se tomó la medición.

Multiplicidad de mapas y consistencia *app-web*

La pantalla de potreros cuenta con un mapa donde visualizarlos. Se puede optar por un mapa plano donde solo se muestran calles o un mapa satelital donde se puede ver el terreno como fotos. Actualmente, esto es una configuración inflexible, tomada arbitraria y unilateralmente, ya que la *webapp* no cuenta con la posibilidad de cambiar mapas. Existe un problema de consistencia entre los mapas presentados en la aplicación móvil y la aplicación web. Entonces, si se agrega la misma funcionalidad a la web*,* se debería definir una configuración única para ambos sistemas. Cabe destacar la importancia de los mapas porque sobre ellos se definen y visualizan los potreros, siendo casi imposible definirlos en un mapa sin relieve.

Desarrollo móvil y distribución

Una característica diferenciadora del producto pasturómetro es que utiliza el teléfono celular para abaratar costos de producción del dispositivo pasturómetro, esto se logra al delegar funcionalidades al teléfono como son el GPS y la transmisión de datos. Teniendo en cuenta esto, se necesita que la aplicación esté disponible en la mayor cantidad de plataformas posibles. Como se presentó anteriormente para desarrollar la aplicación, se empleó Expo, que ayudó en la creación de la aplicación tanto para Android como para iOS. Sin embargo, la funcionalidad de Expo no se detiene allí, ya que también es de suma importancia en la distribución de la aplicación, creando versiones específicas y certificados para las tiendas que lo requieren. Cabe destacar que publicar una *app* tanto en la Play Store como en la App Store requiere de investigación e implicación al respecto. Particularmente crear una aplicación para iOS y publicarla presenta una mayor dificultad, ya que requiere utilizar *hardware* específico para ello (dispositivos con macOS) y además para su publicación puede ser necesario ofrecer capturas en un dispositivo iPhone real.

Conclusiones

En retrospectiva, el proyecto de la aplicación móvil Pastech cumple sus objetivos. Se desarrolló una aplicación que facilita la tarea de realizar mediciones de pasturas en entornos rurales para su monitoreo. La aplicación permite recibir mediciones vía Bluetooth y automatiza su carga en un sistema de monitoreo. Además, brinda las funciones necesarias para crear y cargar calibraciones, así como para mostrar información que orienta a quien realiza las mediciones sobre estadísticas básicas, mediciones anteriores y delimitación de potreros. El conjunto de estas funcionalidades logra reducir en un 50% el tiempo necesario para la tarea de medición, al evitar múltiples pasos que anteriormente se realizaban de forma manual.

En cuanto a la gestión del proyecto, el desarrollo de la aplicación se dio en un tiempo mayor al planeado, pero en una proporción abarcada dentro de lo esperado debido a la falta de experiencia de los integrantes. Esto último era crucial debido a la importancia de entrar rápidamente en el mercado ganando una posición competitiva.

Existió un desfase en el último tramo del proyecto de escritura del informe, el cual en su inicio se minimizó y no se consideró la dificultad que presentaba su naturaleza, no se pensó que esta etapa no sería tan emocionante para los integrantes como las anteriores y al no prever esto se generó una pérdida de inercia en el proyecto. Esto a su vez dejó una lección de suma importancia en el equipo para futuros proyectos. Tomar en consideración no solo las dificultades técnicas de un proyecto, sino también las de índole personal para generar estrategias y así afrontarlas. Las demás etapas se desarrollaron dentro de los tiempos estimados, debido principalmente a la motivación del equipo y placer que involucró investigar, diseñar y desarrollar un producto que sea de utilidad para la sociedad. Además, las herramientas que se aprendieron en el transcurso fueron de sumo interés para el equipo y hasta lograron brindarles oportunidades laborales con su uso.

En conclusión, el proyecto cumplió con sus objetivos, el equipo encontró problemas no contemplados a la hora de la planificación del proyecto que aun así fueron superados y dejaron importantes lecciones para sus integrantes. En su totalidad el proyecto fue sumamente enriquecedor, brindó la experiencia de enfrentar un problema real, hacer uso de las herramientas adquiridas a lo largo de la formación académica, adquirir nuevas, trabajar

en colaboración, planificar y gestionar un proyecto de principio a fin. Todas estas son esenciales en el desempeño de un Ingeniero Informático.

Anexo

Análisis FODA del producto

Fortalezas

- **Fácil e intuitivo**: El producto se diseñó y desarrolló para ser accesible e intuitivo para todo tipo de usuarios.
- **Rendimiento**: Su diseño simple y funcionalidad sencilla permiten un buen rendimiento en la mayoría de los dispositivos móviles del mercado.
- **Mercado potencial amplio:** La solución puede ser implementada y comercializada en general para todos los productores ganaderos, tanto a nivel nacional como internacional.
- **Uso de tecnología móvil:** El producto aprovecha las características de los teléfonos móviles para su funcionamiento, lo que reduce los costos de producción del dispositivo pasturómetro y su precio de venta, brindando así una ventaja competitiva adicional.
- **Potencial comercial**: La solución propuesta puede aplicarse y comercializar tanto a nivel local como internacional, ofreciendo una herramienta útil a productores ganaderos dentro y fuera del país.

Oportunidades

- **Mercado local sin explotar**: Existe una oportunidad de ofrecer una alternativa económica adaptada a las necesidades específicas del mercado local, cubriendo un vacío en el sector.

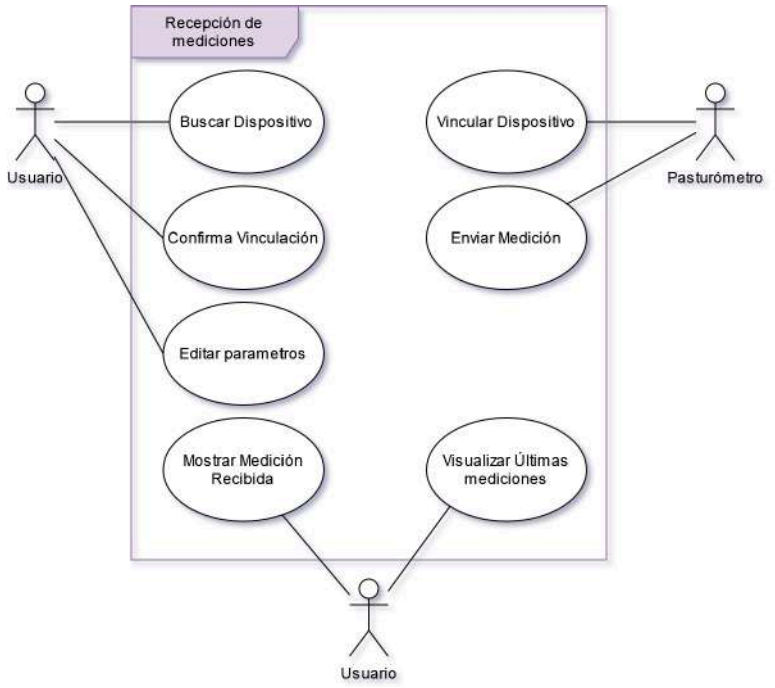
Debilidades

- **Mantenimiento**: El mantenimiento post lanzamiento será limitado en su comienzo debido a la naturaleza académica del proyecto.

Amenazas

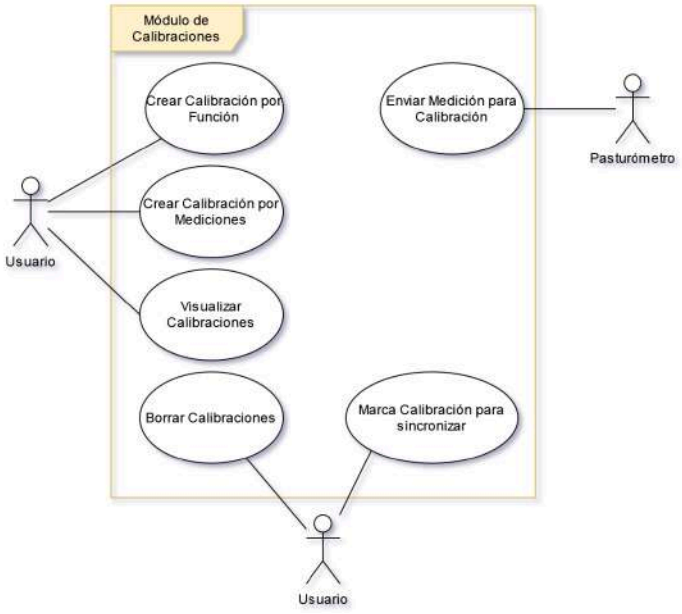
- **Resistencia al cambio**: Los productores agropecuarios podrían mostrar reticencia a adoptar nuevas tecnologías, prefiriendo los métodos tradicionales.
- **Competencia**: Posible aparición de un competidor en el mercado objetivo.

Diagramas de casos de uso



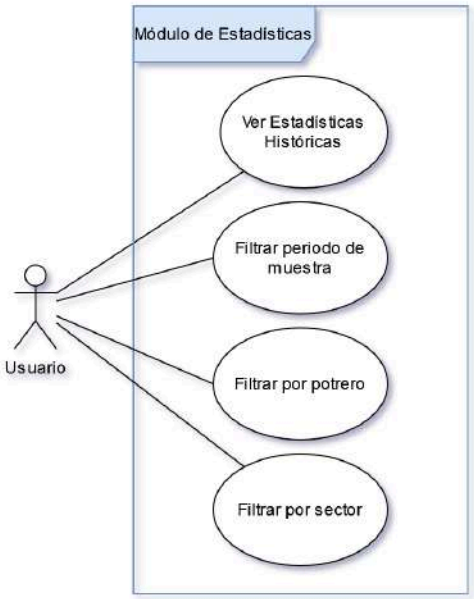
Recepción de mediciones

- Recibir las mediciones del pasturómetro.
- Almacenar mediciones recibidas.
- Mostrar últimas mediciones.
- Notificar fallas de mediciones.



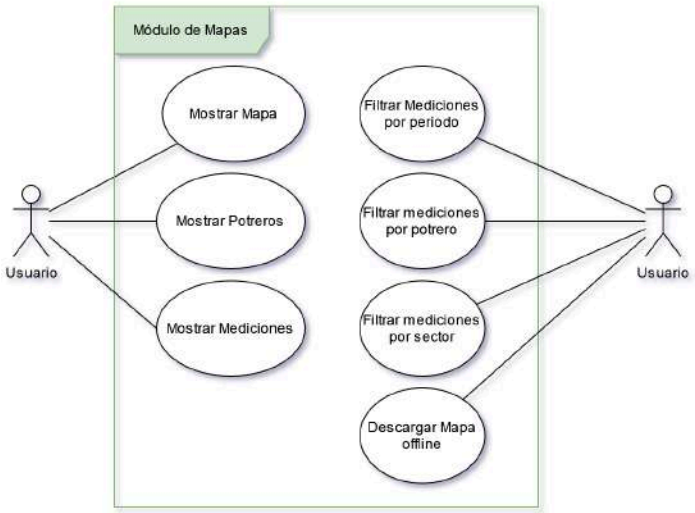
Calibraciones

- Crear calibraciones a partir de una función o a partir de mediciones.
- Visualizar calibraciones.
- Cargar mediciones para una calibración.



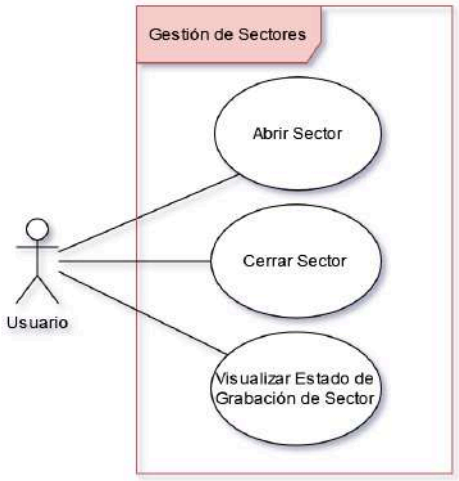
Estadísticas

- Mostrar mediciones históricas.
- Cambiar el periodo a mostrar.
- Filtrar por potrero.
- Filtrar por sector.



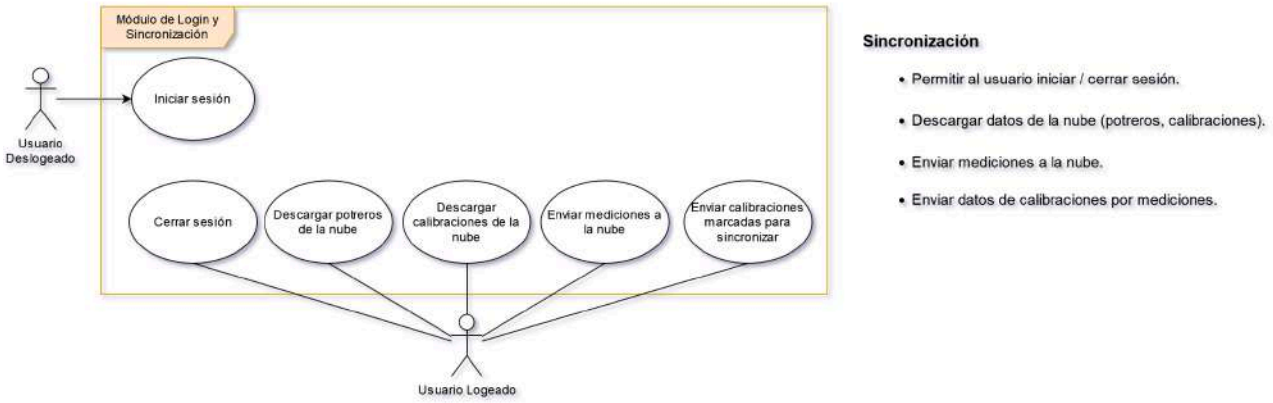
Mapas

- Mostrar un mapa que permita ubicar los potreros.
- Permitir mostrar mediciones en su respectiva ubicación.
- Permitir filtrar mediciones por periodo, potrero y/o sector.
- Permitir descargar zonas del mapa para visualización *offline*.



Sectores

- Abrir sector que permita organizar temporalmente las mediciones.
- Cerrar sector.



Página oficial de Pastech

En el siguiente link se puede acceder a la página oficial de Pastech para informarse del proyecto en su totalidad.

https://pastech.com.ar

Casos de uso completos

Módulo calibraciones

1. Caso de Uso: Crear Calibración por Mediciones

- **Resumen:** Permite al usuario crear una calibración basada en regresión lineal a partir de un conjunto de mediciones seleccionadas.
- **Tipo:** Primario, funcional.
- **Precondición:** El usuario debe estar autenticado y tener acceso a mediciones existentes.
- **Postcondición:** Se crea una calibración del tipo por mediciones y se almacena en el sistema a la espera de ser completada con datos adicionales.
- **Flujo Principal:**
    1. El usuario accede a la sección de creación de calibraciones.
    2. Selecciona la opción para crear una calibración por mediciones.
    3. La aplicación solicita un nombre para la calibración.
    4. El usuario proporciona un nombre y confirma la creación de la calibración.
    5. La aplicación crea una calibración preliminar, pendiente de asignación de mediciones y pesos específicos.

2. Caso de Uso: Asignar Mediciones a Calibración

- **Resumen:** Permite al usuario asignar mediciones específicas y sus correspondientes pesos a una calibración por mediciones para la regresión lineal.
- **Tipo:** Secundario, funcional.
- **Precondición:** Debe existir al menos una calibración por mediciones creada y mediciones disponibles para asignar.
- **Postcondición:** Las mediciones y los pesos asociados se asignan a la calibración seleccionada.
- **Flujo Principal:**
    1. El usuario accede a la pantalla de “Medición de Calibración”.
    2. Selecciona una calibración existente a la que asignar mediciones.
    3. La aplicación muestra una lista de IDs de mediciones disponibles.
    4. El usuario selecciona una medición y proporciona el peso correspondiente que ha medido manualmente.
    5. La aplicación asocia la medición con su peso a la calibración.
    6. Este proceso se repite para todas las mediciones que el usuario desea incluir en la calibración.

3. Caso de Uso: Marcar calibración como lista para envío y procesamiento en sistema Cloud

- **Resumen:** El usuario marca una calibración como lista para ser procesada por el servidor.
- **Tipo:** Primario, funcional.
- **Postcondición:** La calibración se marca como lista para procesamiento. En cuanto haya conexión con el servidor.
- **Flujo Principal:**
    1. Después de asignar mediciones, el usuario marca la calibración como lista.
    2. La aplicación verifica la conexión a internet. Si hay conexión, procede al paso 3. Si no hay conexión, espera hasta que se establezca una conexión para proceder.
    3. Una vez conectada a internet, la aplicación envía los datos de la calibración al servidor para su procesamiento.

Módulo cloud

1. **Caso de Uso: Iniciar Sesión**
    - **Resumen:** Permite al usuario iniciar sesión en la aplicación para acceder a funciones protegidas.
    - **Tipo:** Primario, esencial.
    - **Precondición:** El usuario debe tener una cuenta registrada.
    - **Postcondición:** El usuario accede a su cuenta en la aplicación.
    - **Flujo Principal:** El usuario ingresa sus credenciales y la aplicación valida y concede acceso.
2. **Caso de Uso: Cerrar Sesión**
    - **Resumen:** Permite al usuario cerrar su sesión actual en la aplicación.
    - **Tipo:** Primario, esencial.
    - **Precondición:** El usuario debe estar actualmente conectado.
    - **Postcondición:** El usuario cierra su sesión en la aplicación.
    - **Flujo Principal:** El usuario selecciona la opción de cierre de sesión y la aplicación termina su sesión.
3. **Caso de Uso: Descargar Potreros de Nube**
    - **Resumen:** Permite al usuario descargar información de potreros desde el sistema en la nube.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** El usuario debe estar conectado a internet y haber iniciado sesión.
    - **Postcondición:** La información de potreros se descarga y almacena localmente.
    - **Flujo Principal:** La aplicación luego de un determinado tiempo intenta descargar los potreros definidos en el sistema *Cloud*.
    - **Flujo Alternativo:** El usuario selecciona la opción sincronizar en la pantalla de cuenta.
4. **Caso de Uso: Descargar Calibraciones de Nube**
    - **Resumen:** Permite descargar calibraciones desde el sistema en la nube.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** El usuario debe estar conectado a internet y haber iniciado sesión.
    - **Postcondición:** Las calibraciones se descargan y almacenan localmente.

- **Flujo Principal:** La aplicación luego de un determinado tiempo intenta descargar.
- **Flujo Alternativo:** El usuario selecciona la opción sincronizar en la pantalla de cuenta.

5. **Caso de Uso: Enviar Mediciones a la Nube**
   - **Resumen:** Permite al usuario enviar mediciones almacenadas localmente al sistema en la nube.
   - **Tipo:** Secundario, funcional.
   - **Precondición:** El usuario debe tener mediciones almacenadas y estar conectado a internet.
   - **Postcondición:** Las mediciones se suben al sistema en la nube.
   - **Flujo Principal:** La aplicación luego de un determinado tiempo intenta enviar las mediciones realizadas.
   - **Flujo Alternativo:** El usuario selecciona la opción sincronizar en la pantalla de cuenta.

6. **Caso de Uso: Enviar Calibraciones Marcadas como listas**
   - **Resumen:** Envía calibraciones específicas marcadas por el usuario al sistema en la nube.
   - **Tipo:** Secundario, funcional.
   - **Precondición:** El usuario debe haber marcado calibraciones para enviar y estar conectado a internet.
   - **Postcondición:** Las calibraciones marcadas se suben al sistema en la nube.
   - **Flujo Principal:** La aplicación luego de un determinado tiempo intenta enviar las calibraciones marcadas como listas.
   - **Flujo Alternativo:** El usuario selecciona la opción sincronizar en la pantalla de cuenta.

Módulo de mapas

1. **Caso de Uso: Mostrar Mapa**
   - **Resumen:** Muestra un mapa interactivo en la aplicación.
   - **Tipo:** Primario, informativo.
   - **Precondición:** La aplicación debe tener acceso a datos de mapa.
   - **Postcondición:** El usuario visualiza el mapa en la aplicación.

- **Flujo Principal:** El usuario accede a la sección del mapa y la aplicación muestra el mapa interactivo.

2. **Caso de Uso: Mostrar Potreros**
   - **Resumen:** Muestra los potreros en el mapa.
   - **Tipo:** Secundario, informativo.
   - **Precondición:** Debe existir información de potreros disponible.
   - **Postcondición:** Los potreros se visualizan en el mapa.
   - **Flujo Principal:** El usuario selecciona la opción de visualizar potreros y la aplicación los muestra en el mapa.
3. **Caso de Uso: Mostrar Mediciones**
   - **Resumen:** Muestra las mediciones en el mapa.
   - **Tipo:** Secundario, informativo.
   - **Precondición:** Deben existir mediciones disponibles.
   - **Postcondición:** Las mediciones se visualizan en el mapa.
   - **Flujo Principal:** El usuario selecciona la opción de visualizar mediciones y la aplicación las muestra en el mapa.

4. **Caso de Uso: Filtrar Mediciones por Periodo**
   - **Resumen:** Permite filtrar las mediciones mostradas en el mapa por un periodo específico.
   - **Tipo:** Secundario, funcional.
   - **Precondición:** Deben existir mediciones disponibles.
   - **Postcondición:** Se muestran en el mapa solo las mediciones del periodo seleccionado.
   - **Flujo Principal:** El usuario selecciona un periodo y la aplicación filtra y muestra las mediciones correspondientes.
5. **Caso de Uso: Filtrar Mediciones por Potrero**
   - **Resumen:** Filtra las mediciones en el mapa basándose en el potrero seleccionado.
   - **Tipo:** Secundario, funcional.
   - **Precondición:** Deben existir mediciones en varios potreros.
   - **Postcondición:** Se muestran en el mapa solo las mediciones del potrero seleccionado.
   - **Flujo Principal:** El usuario selecciona un potrero y la aplicación filtra y muestra las mediciones correspondientes.

6. **Caso de Uso: Filtrar Mediciones por Sector**
    - **Resumen:** Filtra las mediciones en el mapa basándose en el sector seleccionado.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** Deben existir mediciones en varios sectores.
    - **Postcondición:** Se muestran en el mapa solo las mediciones del sector seleccionado.
    - **Flujo Principal:** El usuario selecciona un sector y la aplicación filtra y muestra las mediciones correspondientes.
7. **Caso de Uso: Descargar Mapa para Uso Offline**
    - **Resumen:** Permite al usuario descargar el mapa para su uso sin conexión a internet.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** El usuario debe tener acceso a internet para la descarga inicial.
    - **Postcondición:** El mapa queda disponible para su uso sin conexión.
    - **Flujo Principal:** El usuario selecciona la opción de descargar el mapa y la aplicación realiza la descarga.

Módulo de Estadísticas

1. **Caso de Uso: Ver Estadísticas Históricas**
    - **Resumen:** Muestra estadísticas históricas relacionadas con las mediciones.
    - **Tipo:** Primario, analítico.
    - **Precondición:** Deben existir datos históricos de mediciones.
    - **Postcondición:** El usuario visualiza estadísticas históricas.
    - **Flujo Principal:** El usuario accede a la sección de estadísticas y la aplicación muestra los datos históricos.
2. **Caso de Uso: Filtrar Periodo de Muestra**
    - **Resumen:** Permite filtrar las estadísticas mostradas por un periodo específico.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** Deben existir datos de mediciones para varios periodos.
    - **Postcondición:** Se muestran estadísticas del periodo seleccionado.
    - **Flujo Principal:** El usuario selecciona un periodo y la aplicación muestra las estadísticas correspondientes.

3. **Caso de Uso: Filtrar por Potrero**
    - **Resumen:** Filtra las estadísticas mostradas basándose en el potrero seleccionado.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** Deben existir datos de mediciones en varios potreros.
    - **Postcondición:** Se muestran estadísticas del potrero seleccionado.
    - **Flujo Principal:** El usuario selecciona un potrero y la aplicación muestra las estadísticas correspondientes.
4. **Caso de Uso: Filtrar por Sector**
    - **Resumen:** Filtra las estadísticas mostradas basándose en el sector seleccionado.
    - **Tipo:** Secundario, funcional.
    - **Precondición:** Deben existir datos de mediciones en varios sectores.
    - **Postcondición:** Se muestran estadísticas del sector seleccionado.
    - **Flujo Principal:** El usuario selecciona un sector y la aplicación muestra las estadísticas correspondientes.

API Sistema Cloud

- **Endpoint:** /mobile-api/v1/auth/login
- **HTTP Method:** POST
- **Roles:** *
- **Request:** { email: string, password:string }
- **Response:** { access_token: string, expires_in: int, refresh_expires_in: int, refresh_token: string }
- **Descripción:** Autentica al usuario y retorna tokens para la gestión de sesión.

- **Endpoint:** /mobile-api/v1/auth/refreshToken
- **HTTP Method:** POST
- **Roles:** *
- **Request:** { refresh_token: string }
- **Response:** { access_token: string, expires_in: int, refresh_expires_in: int, refresh_token: string }
- **Descripción:** Refresca la sesión de autenticación emitiendo un nuevo token de acceso.

- **Endpoint:** /mobile-api/v1/auth/verifyToken
- **HTTP Method:** GET
- **Roles:** *
- **Request:** N/A
- **Response:** { data: UserResponse, message: string }
- **Descripción:** Verifica la validez del token del usuario.

- **Endpoint:** /mobile-api/v1/paddocks
- **HTTP Method:** GET
- **Roles:** OWNER, ADMIN, WORKER, VIEWER
- **Request:** query parameters: { pageNumber?: number, pageSize?: number }
- **Response:** { data: Page<PaddockResponse>, message: string }
- **Descripción:** Recupera una lista paginada de potreros.

- **Endpoint:** /mobile-api/v1/paddocks/{uid}
- **HTTP Method:** GET
- **Roles:** OWNER, ADMIN, WORKER, VIEWER
- **Request:** N/A
- **Response:** { data: PaddockResponse, message: string }
- **Descripción:** Obtiene información detallada de un potrero específico.

- **Endpoint:** /mobile-api/v1/paddocks/{uid}
- **HTTP Method:** PATCH
- **Roles:** OWNER, ADMIN
- **Request:** { name?: string, speciesUid?: string, calibrationUid?: string, geofence?: [latitude, longitude][] }
- **Response:** { message: string }
- **Descripción:** Actualiza los detalles de un potrero específico.

- **Endpoint:** /mobile-api/v1/paddocks/
- **HTTP Method:** POST
- **Roles:** OWNER, ADMIN
- **Request:** { name: string, speciesUid?: string, calibrationUid?: string, geofence: [latitude, longitude][] }
- **Response:** { data: string (new paddock uid), message: string }
- **Descripción:** Crea un nuevo potrero con los detalles especificados.

- **Endpoint:** /mobile-api/v1/calibrations
- **HTTP Method:** GET
- **Roles:** OWNER, ADMIN, WORKER, VIEWER
- **Request:** query parameters: { pageNumber?: number, pageSize?: number }
- **Response:** { data: Page<CalibrationDto>, message: string }
- **Descripción:** Recupera una lista paginada de calibraciones.

- **Endpoint:** /mobile-api/v1/calibrations/{uid}
- **HTTP Method:** GET
- **Roles:** OWNER, ADMIN, WORKER, VIEWER
- **Request:** N/A
- **Response:** { data: CalibrationResponse, message: string }
- **Descripción:** Obtiene información detallada de una calibración específica.

- **Endpoint:** /mobile-api/v1/calibrations/
- **HTTP Method:** POST
- **Roles:** OWNER, ADMIN, WORKER
- **Request:** { name?: string, degree?: number, measurements: { id: string, measurement: MeasurementDto }[] }
- **Response:** { data: string(calibration), message: string }
- **Descripción:** Crea una nueva calibración basada en las mediciones proporcionadas.

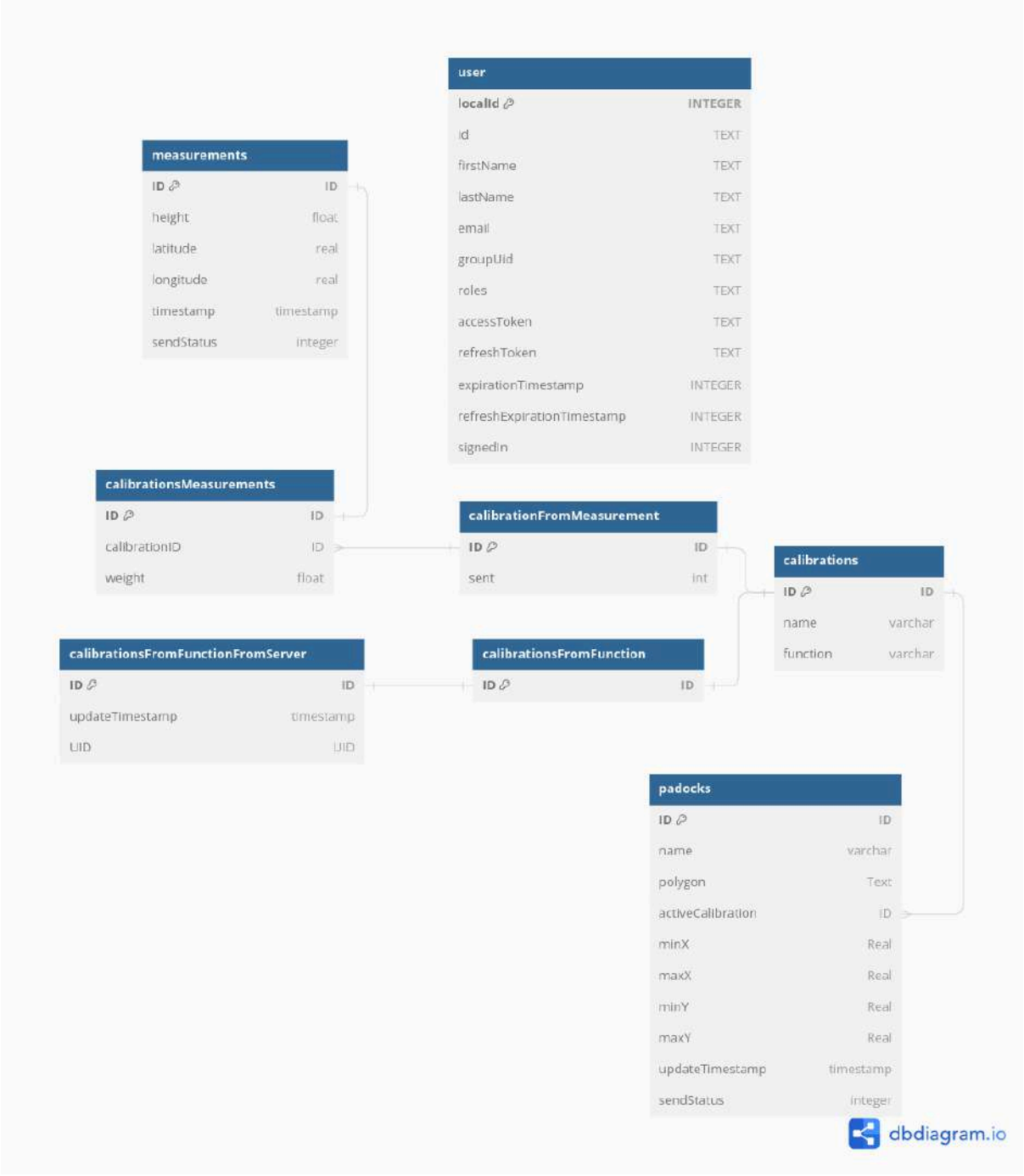
- **Endpoint:** /mobile-api/v1/measurements
- **HTTP Method:** GET
- **Roles:** OWNER, ADMIN, WORKER, VIEWER
- **Request:** query parameters: { pageNumber?: number, pageSize?: number }
- **Response:** { data: Page<MeasurementDto>, message: string }
- **Descripción:** Recupera una lista paginada de mediciones.

- **Endpoint:** /mobile-api/v1/measurements/
- **HTTP Method:** POST
- **Roles:** OWNER, ADMIN, WORKER
- **Request:** { measurements: MeasurementDto[] }
- **Response:** { message: string }
- **Descripción:** Envía un lote de mediciones para su procesamiento.

Repositorio

Enlace al código fuente del proyecto.
https://github.com/veitchm/past-ar

DER


user
localId INTEGER
id TEXT
firstName TEXT
lastName TEXT
email TEXT
groupUid TEXT
roles TEXT
accessToken TEXT
refreshToken TEXT
expirationTimestamp INTEGER
refreshExpirationTimestamp INTEGER
signedIn INTEGER
measurements
ID ID
height float
latitude real
longitude real
timestamp timestamp
sendStatus integer
calibrationsMeasurements
ID ID
calibrationID ID
weight float
calibrationFromMeasurement
ID ID
sent int
calibrations
ID ID
name varchar
function varchar
calibrationsFromFunctionFromServer
ID ID
updateTimestamp timestamp
UID UID
calibrationsFromFunction
ID ID
padocks
ID ID
name varchar
polygon Text
activeCalibration ID
minX Real
maxX Real
minY Real
maxY Real
updateTimestamp timestamp
sendStatus integer
dbdiagram.io

Bibliografía

1. Ciclo de vida de aplicación móvil. https://blog.kms-solutions.asia/mobile-app-development-lifecycle
2. Ciclo de vida de aplicación móvil. https://medium.com/swlh/a-guide-to-the-5-step-mobile-app-development-lifecycle-dad5122f882
3. Ciclo de vida de aplicación móvil. https://www.netguru.com/blog/mobile-app-development-lifecycle
4. Página oficial Jenquip https://www.jenquip.nz/
5. Documentación oficial de Flutter. Flutter. https://docs.flutter.dev/
6. Documentación oficial de React Native. React Native. https://reactnative.dev/
7. Documentación oficial de Expo. Expo. https://docs.expo.dev/
8. JavaScript. Mozilla. https://developer.mozilla.org/en-US/docs/Web/javascript
9. Typescript. Typescriptlang. https://www.typescriptlang.org/
10. Typescript. Wikipedia. https://es.wikipedia.org/wiki/TypeScript
11. Documentación oficial de React-Redux. React-Redux. https://react-redux.js.org/
12. Documentación oficial de Recoil. RecoilJS. https://recoiljs.org/docs/introduction/motivation
13. React Context API. React. https://react.dev/reference/react/useContext
14. Documentación oficial Zustand https://docs.pmnd.rs/zustand/getting-started/comparison
15. Comparación base de datos relacionales y no relacionales. Amazon. https://aws.amazon.com/es/compare/the-difference-between-relational-and-non-relational-databases/
16. Documentación oficial de SQLite. Expo. https://docs.expo.dev/versions/latest/sdk/sqlite/
17. Información general de SQLite. Wikipedia. https://en.wikipedia.org/wiki/SQLite
18. Documentación oficial de SQLite. SQLite. https://www.sqlite.org
19. Documentación oficial de React-Native-BLE-PLX. React Native BLE PLX. https://dotintent.github.io/react-native-ble-plx/
20. Documentación oficial de React-Native-BLE-Manager. React Native BLE Manager. https://innoveit.github.io/react-native-ble-manager/

21. Documentación oficial de react-native-maps
https://github.com/react-native-maps/react-native-maps
22. Precios de Google Maps API
https://mapsplatform.google.com/pricing/
23. Precios de Mapbox
https://www.mapbox.com/pricing
24. Proyecto MapLibre
https://maplibre.org/
25. Librerías de interfaz visual más populares para React Native. Philip Daineka.
https://flatlogic.com/blog/top-react-native-ui-components-libraries/
26. Documentación oficial de Native-Base. Native Base
https://nativebase.io/
27. Documentación oficial de React-Native-Paper. React Native Paper.
https://reactnativepaper.com/
28. Comparación del entorno de desarrollo de React Native con el de Expo.
https://retool.com/blog/expo-cli-vs-react-native-cli/
29. Utilización de wireframes en diseño UX/UI.
https://www.lucidchart.com/pages/wireframe
30. Diseño para desarrolladores. Mozilla.
https://developer.mozilla.org/en-US/curriculum/core/design-for-developers/