Trabajo de Fin de Grado

Grado en Ingeniería de Tecnologías de Telecomunicación

Desarrollo de Aplicación Híbrida para la Visualización de Datos Generados en Tecnologías del Internet de las Cosas

Autor: José Manuel Maldonado López

Tutor: Juan Antonio Becerra González

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Grado
Ingeniería de Tecnologías de Telecomunicación

Desarrollo de Aplicación Híbrida para la Visualización de Datos Generados en Tecnologías del Internet de las Cosas

Autor:

José Manuel Maldonado López

Tutor:

Juan Antonio Becerra González

Profesor Sustituto Interino

Dpto. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Desarrollo de Aplicación Híbrida para la Visualización de Datos Generados en Tecnologías del Internet de las Cosas

Autor: José Manuel Maldonado López

Tutor: Juan Antonio Becerra González

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocal/es:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia

A mis maestros

AGRADECIMIENTOS

Resulta bastante complicado incluir en este apartado a todas las personas que han estado al pie del cañón desde que empezó toda esta travesía, pero quiero dejar constancia de todos ellos aquí para tenerlos siempre.

Empezando por mis amigos, pero los de verdad, los que se cuentan con una mano, Antonio y Juan, creyendo en mí desde tiempos inmemoriales, ni cuando yo mismo lo hacía. A mi amigo Salvador, amistad desde que tengo uso de razón que retomamos en esta carrera, cuantas charlas acerca del sentido de la vida y de si todo esto realmente merecía la pena, insustituible. Y finalmente a Sergio, cuantas experiencias juntos y apoyo en la distancia.

A mis compañeros de carrera, que ha habido muchos, pero por nuestro recorrido juntos y el enorme equipo que formamos a Pepe y Ángel, gracias a ellos todo ha sido un poco más fácil.

A mi tutor, Juan Antonio, por en su día darme la oportunidad de hacer unas prácticas de empresa que me han ayudado mucho en mi formación y ahora por acompañarme en el desarrollo de este TFG que tanta ilusión sabe que me hacía.

Como no a toda mi familia, sin que falte ninguno, los que más han creído en mí. Destacando por encima de todo a mis dos pilares, mis padres, tarea inconmensurable la suya, paciencia infinita y apoyo incondicional, siempre han estado ahí, siempre ayudándome a despejar las piedras que se han puesto por delante, siempre. Y especial mención a mi abuelo, que donde quiera que esté sé perfectamente que está orgulloso de mí como el que más.

Y a mi compañera de viaje, Mar, conexión instantánea desde que nos conocimos, apoyo ilimitado desde el minuto cero, no hicieron falta explicaciones.

Gracias a todas y cada una de estas personas por el apoyo absoluto, jamás tendré las suficientes palabras de agradecimiento para vosotros, ahora toca afrontar con pasión todo lo que venga.

José Manuel Maldonado López

Sevilla, 2020

RESUMEN

El sector agrario en España es un sector estratégico por su repercusión económica, social, territorial y medioambiental, de hecho, la mitad de la superficie de España se destina a actividades agrícolas o ganaderas.

El presente Trabajo de Fin de Grado se centra en el desarrollo de una aplicación híbrida cuyo objetivo es facilitar al usuario (agricultor) la visualización de datos que han sido recogidos por sensores ubicados en múltiples localizaciones y almacenados en un servidor para su tratamiento tanto en tiempo real (por ejemplo: consulta de temperatura y humedad del cultivo en este instante) como para una explotación posterior basada en la secuencia histórica de datos almacenados.

Este trabajo recorre los aspectos fundamentales a tener en cuenta a la hora de afrontar el desarrollo de una aplicación, desde el "**front-end**" (lo que el usuario puede ver) hasta el "**back-end**" (la infraestructura que lo soporta), abordando tanto el intercambio de información con la **base de datos** donde se encuentra la información como la revisión para mejor comprensión de la infraestructura del internet de las cosas (IoT, del inglés Internet of Things) que permite, a través de los protocolos de comunicación, la recogida de los mismos para su ulterior almacenaje y explotación.

ABSTRACT

The agricultural sector in Spain is a strategic sector due to its economic, social, territorial and enviromental repercussions. In fact, half of the surface of Spain is used for agricultural or livestock activities.

The present Final Project work focuses on the development of a hybrid application whose objetive is to facilitate the user (farmer) the visualization of data that has been collected by sensors located in multiple locations and stored on a server for its treatment both in real time (for example: query of temperature and humidity of the crop at this momento) as for a subsequent exploitation based on the historical sequence of stored data.

This work covers the fundamental aspects to take into account when facing the development of an application from the "**front-end**" (what the user can see) to the "**back-end**" (the infraestructura that supports it). Addressing both the exchange of information with **database** where the information is located and the review for a better understanding of the Internet of Things infraestructure that allows through communication protocols the collection of them for their subsequent storage and exploitation.

ÍNDICE ABREVIADO

ÍNDICE

NOTACIÓN

Acrónimos

APP	Aplication
AVG	Automatic Guidance Vehicle
BBDD	Base de datos
CSS	Cascading Style Sheets
EU	European Union
GPRS	General Packet Radio Service
GSM	Global Systems for Mobile
HTML	Hypertext Markup Language
IBM	International Business Machine Corporation
IEEE	Institute of Electrical and Electronics Engineers
IIRA	International Industrial Relations Association
IoT	Internet of Things
IP	Internet Protocol
JS	JavaScript
LoRa	Long Range (Modulation)
M2M	Machine to Machine
OSI	Open System Interconnection
QoS	Quality of Service
RFID	Radio-Frequency Identification
SDK	Software Development Kit
WiFi	Wireless Fidelity
WSN	Wireless Sensor Network
3G	Third Generations
4G	Fourth Generations
5G	Fifth Generations

1 INTRODUCCIÓN

"Inteligencia es la habilidad de adaptarse a los cambios"

- Stephen Hawking -

La agricultura es un sector bastante tecnificado y en esta línea de innovación surge la oportunidad de mayor aplicación de la tecnología, combinando Internet of Things (sensores ubicados en el terreno que aportan la lectura de todos los elementos clave a tener en cuenta para optimizar el cultivo), recogida y almacenamiento de datos en un servidor (que ofrece el dato en tiempo real para el agricultor y la serie histórica para facilitar la toma de decisiones predictivas en base a lo observado) y, como parte central de este trabajo, la aplicación, que permitirá al agricultor, con una experiencia de usuario realmente sencilla y tan cómoda como usar una APP desde su smartphone, visualizar cualquier ubicación, leer los valores clave de la ubicación elegida y tomar decisiones adecuadas en el momento adecuado. Por ejemplo, detectar una helada a tiempo permite al agricultor activar el riego por aspersión como medida de protección, basándose en el efecto de la liberación de calor cuando el agua pasa de líquida a 0ºC a sólida a la misma temperatura.

1.1 Objetivos

El objetivo principal del trabajo es el desarrollo e integración de la aplicación con los distintos elementos que intervienen en el ecosistema de desarrollo de aplicaciones basdas en IoT. A su vez, el objetivo de la aplicación es la correcta visualización de la información de los sensores instalados en las distintas facilities (extensiones de terreno) del usuario, combinando las opciones de ubicación en el mapa y búsqueda de facilities y sensores con la mencionada visualización de los datos y gráficos generados por éstos, a través de una experiencia de usuario amigable y simplificada.

Esta visualización facilitará al usuario el proceso de toma de decisiones basando éstas en los datos observados en tiempo real, en primera instancia y, en futuros desarrollos, en análisis de secuencias históricas de datos observados y resultados alcanzados. La finalidad última es obtener la máxima eficiencia de la cosecha, con independencia del tipo de cultivo.

1.2 Alcance

El alcance del presente trabajo es el estudio teórico de cuestiones relativas a elección de tecnología para el desarrollo de aplicaciones y argumentos para la elección entre tecnologías nativas e híbridas. Recorre la implementación de todos los elementos que intervienen en el proceso completo, desde la toma de datos a través de sensores, pasando por los protocolos de comunicaciones que utilizan, contemplando igualmente el almacenamiento e intercambio de información existente en BBDD y, por supuesto, la visuación de la información en la interfaz de usuario.

1.3 Estructura

Este trabajo se desarrollará en 5 capítulos, comenzando por este primer capítulo de introducción, a continuación, en el capítulo segundo, se aborda desde un plano teórico el Ecosistema de Desarrollo de Aplicaciones basadas en IoT (estructurando este capítulo en dos bloques centrados en IoT y Desarrollo de APPs). A continuación, en Materiales y Métodos, se hace un recorrido por el diseño global y específico, la arquitectura aplicada, los lenguajes de programación y bases de datos. Finalizaremos con los capítulos cuarto y quinto donde se aborda en profundidad el diseño experimental con la arquitectura de la aplicación y la secuencia de pantallas para finalizar con las conclusiones y una propuesta de líneas futuras de implementación.

2 ECOSISTEMA DE DESARROLLO DE APLICACIONES BASADAS EN IOT

"La única manera de hacer un trabajo GENIAL, es AMAR lo que haces"

- Steve Jobs -

A continuación se desarrolla una revisión teórica de los distintos apartados clave que intervienen en el ecosistema tecnológico que culminará con la interacción del usuario a través de la aplicación para mostrar y explotar la información. Comenzaremos revisando el IoT desde una visión completa partiendo del marco contextual y finalizando con el futuro del IoT, continuaremos con una revisión del complejo mundo del desarrollo de APPs que ofrece multitud de alternativas que cambian y evolucionan a gran velocidad y finalizaremos con un estudio de opciones.

2.1 Internet of Things (IoT)

El internet de las cosas es un campo emergente de extraordinaria importancia técnica, social y económica. Circulan multitud de estudios que pronostican que en pocos años conviviremos con decenas de miles de millones de dispositivos conectados a IoT con un impacto económico billonario.
Podemos identificar, además del desafío tecnológico que será donde nos centraremos en este trabajo, otros aspectos relevantes como son el regulatorio, jurídico, político y todo lo relativo a la ciberseguridad.

2.1.1 Introducción. Marco Contextual

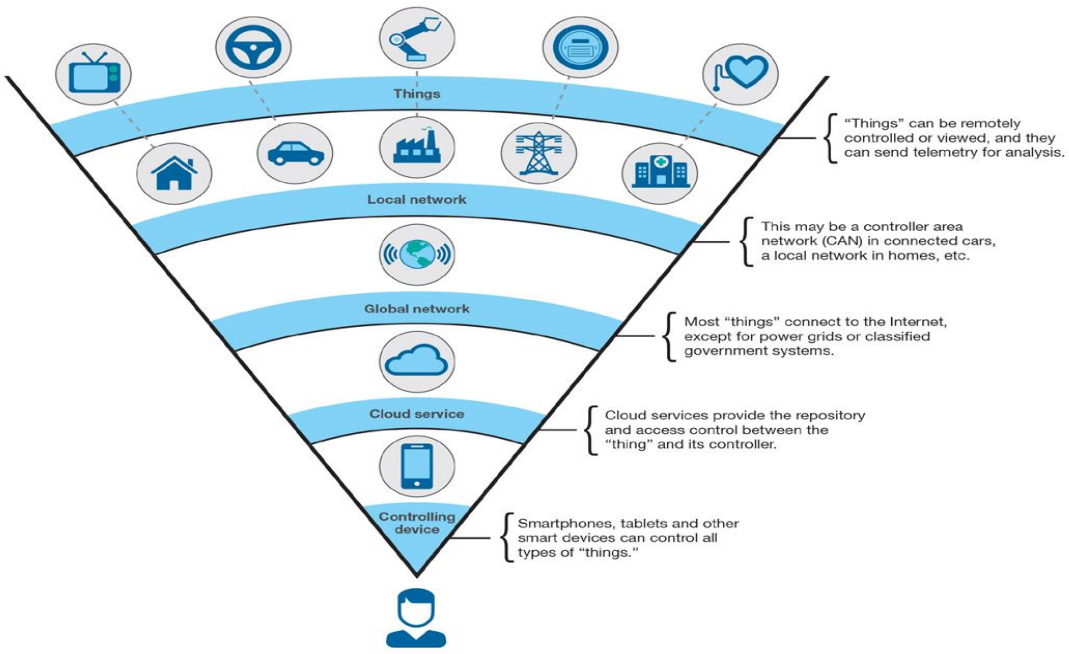


Figura 2.1.1 Visión humana del Internet de las Cosas. Fuente: X-Force Research and Development

No existiendo una definición única y universal podríamos decir que el término IoT se refiere a escenarios donde la conectividad de red y la capacidad de cómputo se extienden a sensores y objetos de uso diario que no son computadoras permitiendo que estos dispositivos generen, intercambien y consuman datos con una mínima intervención humana.

Estas tecnologías instrumentales incluyen de manera omnipresente la conectividad, la adopción generalizada de redes basadas en el protocolo IP, la economía en la capacidad de cómputo, la miniaturización, los avances en el análisis de datos y el surgimiento de la computación en la nube.

Se utilizan diferentes modelos de conectividad, resultado de la flexibilidad en la forma de conexión de los dispositivos, esencialmente 4, a saber:

1. **Device-to-Device.**
2. **Device-to-Cloud.**
3. **Device-to-Gateway.**
4. **Back-End Data-Sharing.**

Identificamos 5 areas temáticas clave de la IoT para explorar desafíos y cuestiones relacionadas con esta tecnología:

1. **Seguridad.**
   A mayor nivel de seguridad, mayor confianza del usuario en los dispositivos conectados.

2. **Privacidad.**
   Es evidente la necesidad de desarrollar estrategias que respeten las opciones de privacidad individual sin dejar de fomentar la innovación en nuevas tecnologías y servicios.

3. **Estándares.**
   La flexibilidad en la integración de dispositivos IoT requiere unos estándares claros y generalmente aceptados que faciliten el proceso de integración y no alteren las reglas de internet. Los usuarios serán los grandes beneficiados de la existencia de estándares genéricos, abiertos y ampliamente disponibles (como el protocolo de internet).

4. **Aspectos Legales.**
   En ocasiones se suele decir que la tecnología avanza más rápido que las leyes. Aspectos como la gestión de datos transfonterizos, jurisdicción, tratamiento de datos personales, aspectos relativos a la seguridad, derechos civiles, etc dejan clara la complejidad de esta temática, así como la necesidad de asumir principios rectores de los principales organismos que velan por el adecuado uso de la tecnología.

5. **Aspectos Globales.**
   Las Naciones Unidas ven en el IoT una herramienta para alcanzar los objetivos de Desarrollo Sostenible [1] (agricultura sostenible, calidad y uso del agua, cuidado de la salud, industrialización y medio ambiente) que no se limitará a los países industrializados por lo que será preciso el diálogo y la colaboración de todas las partes interesadas.

2.1.2 Arquitectura IoT

La arquitectura describe la estructura de la solución de IoT incluyendo aspectos físicos y virtuales.

Es habitual modular la arquitectura por niveles para facilitar la comprensión del funcionamiento de cada capa antes de ser integrada. El enfoque modular ayuda a gestionar la complejidad de las soluciones IoT. La arquitectura de tres niveles sería el mínimo número de niveles referidos a la gestión de dispositivos, conectividad y comunicación y Analítica y Aplicaciones. En la arquitectura de tres niveles se definen los siguientes:

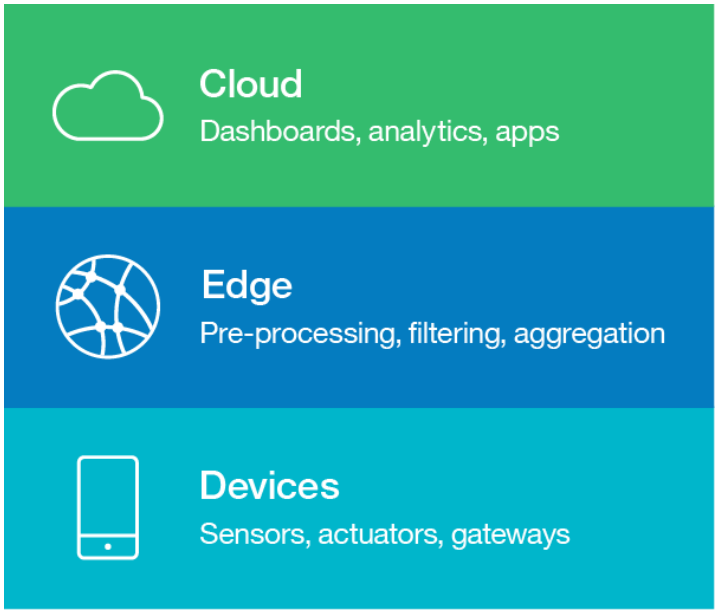


Figura 2.1.2.a Ejemplo de arquitectura de tres niveles. Fuente: developer.ibm.com

1. **Capa de Dispositivos.**

   Los componentes de la capa de dispositivos incluyen sensores físicos y actuadores que están conectados a dispositivos de IoT. Las arquitecturas describen la incorporación del dispositivo, la actualización del firmware del dispositivo, la aplicación de configuraciones nuevas y el desencadenamiento de operaciones remotas como la desactivación, la activación o el desmantelamiento de dispositivos.

2. **Capa Edge.**

   Es la capa relacionada con servicios de analítica y preprocesado que se ubican en el límite de la red y ocurren en tiempo real. La gestión de la conectividad y de la comunicación bidireccional es otra de las capacidades que se suelen describir dentro de las arquitecturas de referencia de IoT.

3. **Capa Cloud.**

   Es en esta capa donde se envían los datos para procesarlos aún más, almacenarlos y utilizarlos dentro de las aplicaciones de la nube. Habitualmente se complementan con aplicaciones móviles y con aplicaciones de clientes basadas en la web, que presentan los datos a los usuarios finales que brindan acceso a herramientas para explorar y analizar más a fondo.

Arquitecturas de referencia.

La necesidad de estandarización para mejorar la interoperabilidad dio lugar a iniciativas promovidas por proveedores de plataformas de IoT e investigadores para definir las denominadas arquitecturas de referencia que actúan como cimientos de la arquitectura que describen los bloques de construcción de alto nivel que se utilizan dentro de las soluciones de IoT y establecen una terminología compartida.

Algunas arquitecturas de referencia de IoT con extensas referencias incluyen:

1. **Internet de las Cosas – Architectura (IoT-A).**

   El modelo y la arquitectura de preferencias de IoT-A se desarrollaron en 2013 a través de un proyecto insignia de la UE. El IoT-A se diseñó para ser construido con la finalidad de desarrollar arquitecturas concretas que son aplicables en variedad de dominios.

2. **IEEE P2413 – Estándar para una Infraestructura Arquitectónica para el Internet de las Cosas [2].**

   Este proyecto de estandarización contínua de IEEE tiene el objetivo de identificar semejanzas entre los dominios de IoT, incluso la fabricación, los edificios inteligentes, las ciudades inteligentes, los sistemas de transporte inteligentes, la red eléctrica inteligente y el cuidado de la salud.

3. **Arquitectura de Referencia de Internet Industrial (IIRA).**

   Industrial Internet Consortium, que fue fundado en marzo de 2014 por AT&T, Cisco, General Electric, IBM e Intel, desarrolló específicamente el IIRA para aplicaciones de IoT industriales.

Requerimientos de Arquitectura IoT.

Los aspectos clave que envuelven la arquitectura son:

1. **Gestión y control de dispositivos.**
   i. La posibilidad de desconectar un dispositivo.
   ii. La habilidad de actualizar el software de un dispositivo.
   iii. La actualización de credenciales de seguridad.
   iv. Autorizar o denegar algunas capacidades del hardware remotamente.
   v. Localizar dispositivos perdidos.
   vi. Limpiar información confidencial de un dispositivo robado.
   vii. Reconfigurar parámetros de Wi-Fi, GPRS u otras redes remotamente.
2. **Recolección, análisis y actuación de los datos.**
3. **Escalabilidad.**
4. **Flexibilidad.**
5. **Alta disponibilidad.**
6. **Integración.**
7. **Seguridad.**
   i. Riesgos inherentes que los diseñadores IoT no tengan conciencia de ellos.

ii. Riesgos específicos de los dispositivos IoT.

iii. Seguridad de que no se causan daños por el mal uso de los actuadores.

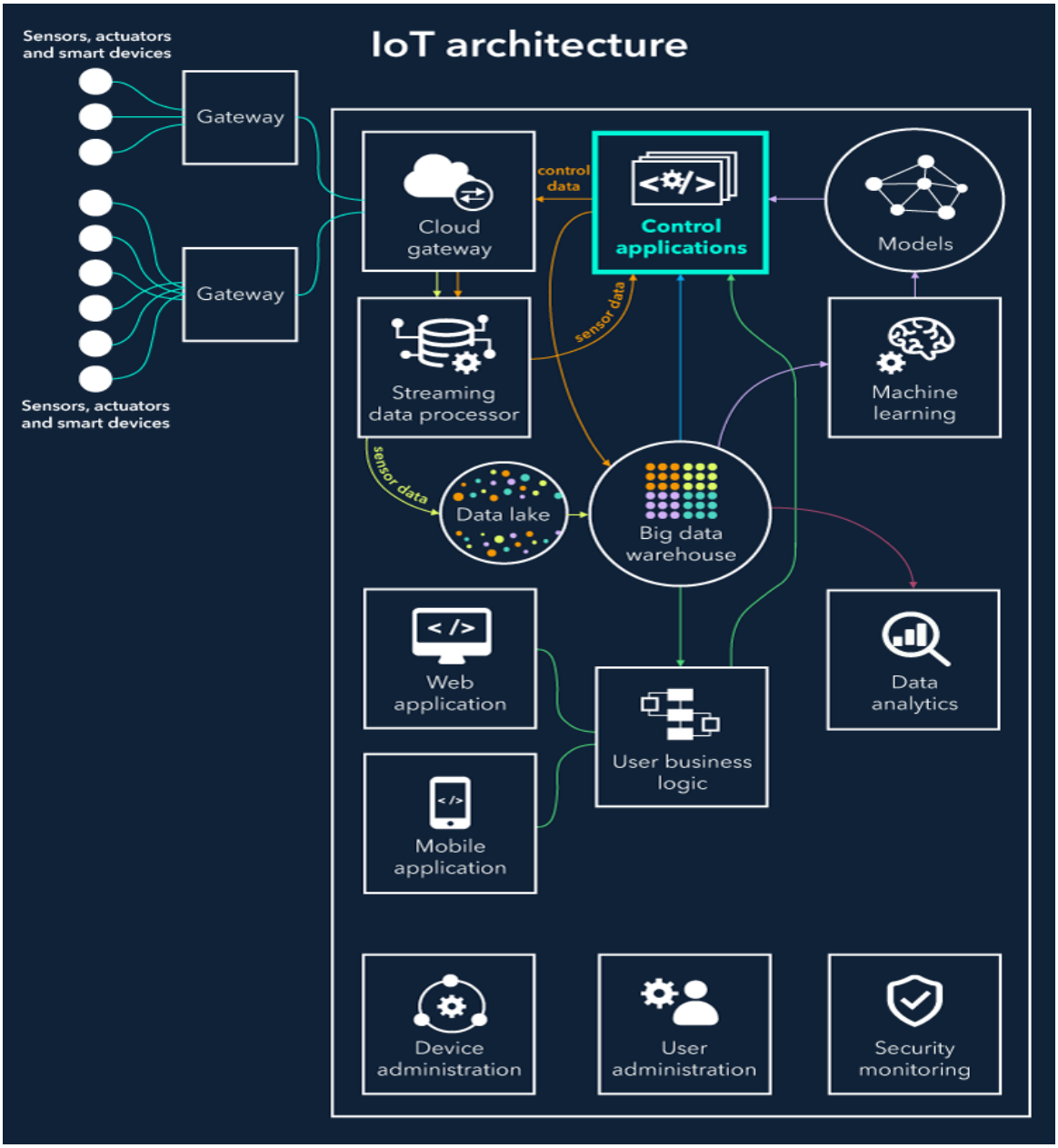
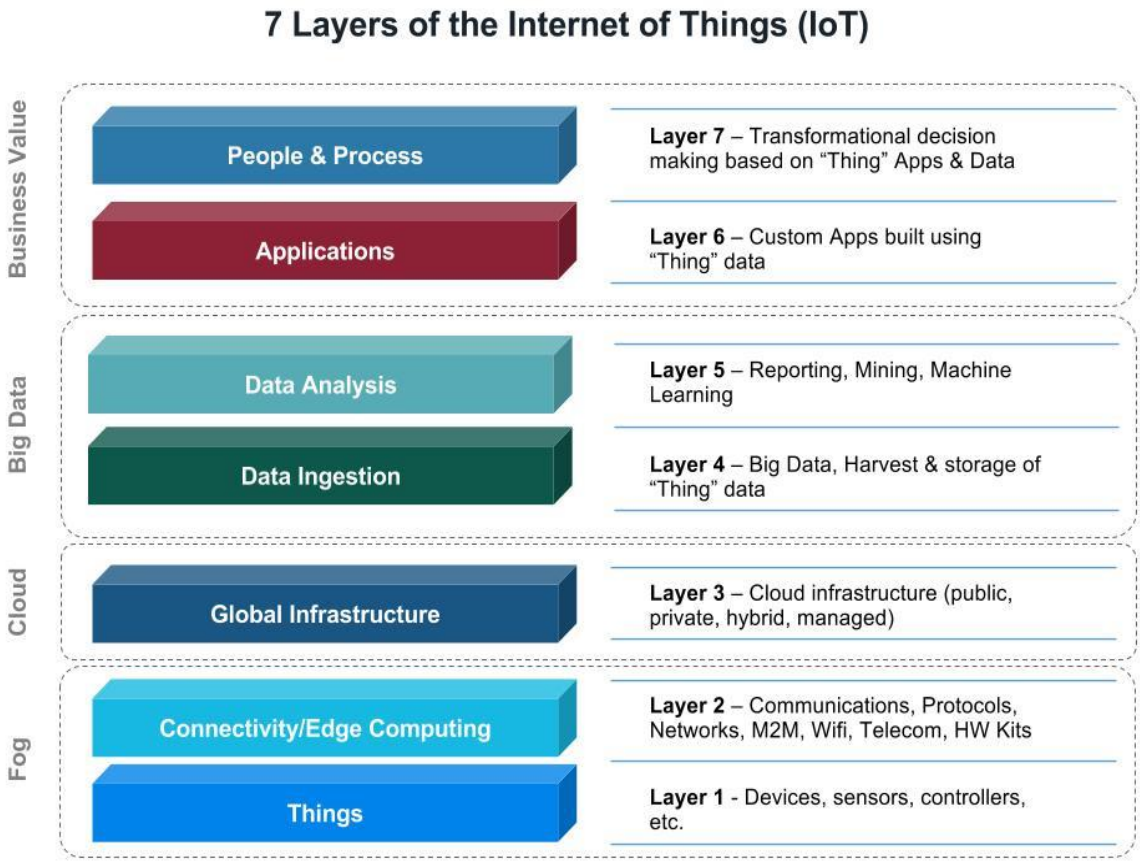


Figura 2.1.2.b Componentes que puede tener una arquitectura IoT. Fuente: scnsoft.com

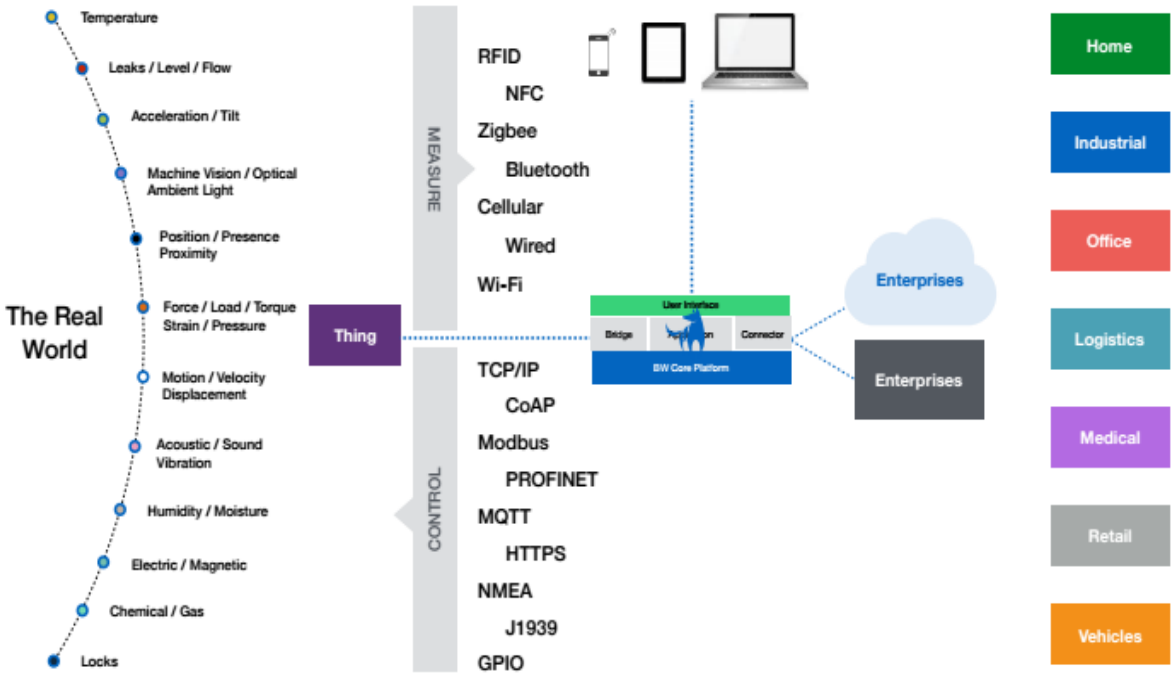
En la imagen podemos observar:

- Cosas equipadas con sensores para recoger datos y actuadores para realizar comandos recibidos desde la nube.
- Gateways para filtrar, preprocesar y mover datos a la nube y viceversa.
- Pasarelas en nube (Cloud Gateways) para garantizar la transición de datos entre las pasarelas sobre el terreno y los servidores centrales de IoT.
- Procesadores de datos en tiempo real para distribuir los datos procedentes de los sensores entre los componentes de las soluciones de IoT.
- Bases de Datos para almacenar todos los datos de valor definido e indefinido.
- Big Data Warehouse para la recogida de datos valiosos.
- Aplicaciones de control para enviar comandos a los actuadores.

- Machine Learning para generar los modelos que luego son utilizados por las aplicaciones de control.
- Aplicaciones de usuario que permiten a los usuarios monitorizar el control de sus cosas conectada.



Figuras 2.1.2.c Modelo de 7 capas de arquitectura IoT. Fuente: researchate.net



Figuras 2.1.2.d Ejemplo del mundo real de arquitectura IoT. Fuente: scielo.mec.pt

2.1.3 Protocolos de Comunicación IoT

Existen muchas opciones de protocolos de comunicación para nuestros dispositivos IoT, algunos cuyo uso ya se ha familiarizado y otros que pueden resultar más propicios para algunas instalaciones, pero son menos conocidos. A continuación, revisamos los principales:

1. **Bluetooth.**

   Importante tecnología de comunicaciones de corto alcance que funciona a 2’4 GHz. Actualmente con dos variantes: El Bluetooth Classic (normal) y el Bluetooth Low Energy (BLE) específicamente destinado a dispositivos que utilizan menos datos.

2. **Wi-Fi.**

   Actualmente la opción más popular para dispositivos IoT en el entorno doméstico. Sus principales ventajas son la comodidad y la capacidad de transferir grandes cantidades de datos de manera muy rápida. También podemos encontrar problemas relativos al mayor consumo de energía (para eso existe el WiFi-ah, versión de bajo consumo y largo alcance del estándar IEEE 802.11 que son muy útiles a la hora de conectar sensores y controladores a larga distancia con una baja velocidad de datos) y relativos a privacidad, retardo y fiabilidad puesto que habitualmente estos dispositivos suelen funcionar desde la nube del fabricante.

3. **ZIGBEE.**

   Protocolo similar al Bluetooth pero que funciona creando una red de área local (LAN) de malla. Se basa en el protocolo IEEE802.15.4, una tecnología de redes inalámbricas a 2’4 GHz y que funciona bien para intercambios de datos poco frecuentes, a bajas velocidades y a distancias cortas (casas o edificios). Ha tenido épocas de mucha popularidad en sistemas domóticos, pero parece que en la actualidad está perdiendo algo de protagonismo. Sin embargo, la incorporación reciente de Apple, Google, Amazon al desarrollo y adopción de

zigbee mediante la creación del Proyect Connected Home over IP va a potenciar el estándar. La especificación rf4ce de Zigbee tiene algunas ventajas significativas en sistemas complejos que ofrecen un funcionamiento de baja potencia, alta seguridad, robustez y alta escalabilidad con un alto número de nodos. Esta diseñado para proporcionar un control de baja potencia y baja latencia para una amplia gama de productos que incluyen dispositivos de entretenimiento para el hogar, sistemas de entrada sin llave, etc.

4. **Z-WAVE.**

Protocolo de comunicaciones inalámbricas utilizado para domótica. Utiliza una red mallada que conecta los diferentes dispositivos Z-Wave mediante una onda de radio de baja energía, permitiendo el control inalámbrico de electrodomésticos, otros dispositivos, como control de iluminación, sistemas de seguridad, termostatos, ventanas, cerraduras, piscinas y garaje abrepuestas. Tiene la ventaja de que las conexiones individuales tienen mayor alcance que las de Zigbee, sin embargo, la señal solo puede hacer 3 saltos. Además, también consume más.

5. **SYMPHONY LINK.**

Es un estándar de red en estrella que tiene la ventaja de conseguir un alcance mucho mayor que Zigbee y Z-Wave. Por lo tanto, es ideal para empresas y clientes industriales que necesitan una conexión fiable y segura. Symphony Link también es mucho más fácil de escalar y es más fiable.

6. **Conexiones Móviles.**

Existe la posibilidad de conectar los dispositivos IoT muy alejados entre sí mediante comunicaciones celulares GSM, 3G, 4G ó 5G. A mayor velocidad de transferencia de datos también aumenta el coste y el consumo de energía. Sus usos ideales son los basados en sensores con poco volumen de transmisión de datos.

7. **RFID.**

La identificación por radiofrecuencia es un protocolo de IoT donde el uso inalámbrico de campos electromagnéticos ayuda a identificar objetos. Las etiquetas de lectura pueden almacenar información y no requieren energía para funcionar. Gracias a esto y su bajo coste, la RFID se ha convertido en una tecnología ampliamente utilizada en las tiendas, peajes, acceso a edificios, recogida de datos en fábricas, etc.

8. **SIGFOX.**

Es una tecnología que, en términos de alcance, se encuentra entre Wi-Fi y las conexiones móviles. Sin embargo, una de las ventajas es que usa las bandas ISM, por lo que no es necesario adquirir ninguna licencia para su uso. Supera el corto alcance del WiFi y es menos costosa y consume menos energía que las conexiones móviles. Utiliza la tecnología UNB (Ultra Narrow Band o Banda Ultra Estrecha) que es capaz de transmitir datos a bajas velocidades (de 10 a 1.000 bits por segundo) utilizando muy poca potencia. Consume solo 50 microvatios en comparación con los 5.000 microvatios de las conexiones móviles.

9. **NEUL.**

   Es similar a Sigfox y opera en la banda de sub-1GHz, sin embargo, requiere el uso de un operador de telecomunicaciones ya que usa franjas muy pequeñas del espectro del espacio blanco de la TV para ofrecer redes de cobertura más amplia y altamente escalables. Se conoce como tecnología Weightless y es compatible con otros protocolos como GPRS, 3G, CDMA y LTE WAN. Su velocidad puede ir desde unos pocos bits por segundo a 1000 Kbps usando el mismo enlace. El consumo de energía de los dispositivos es muy bajo por lo que también aumenta su vida útil.

10. **LoRa.**

    Es un protocolo inalámbrico de largo alcance para dispositivos inalámbricos IoT y M2M en redes regionales, nacionales y globales, por lo que también necesitaremos un operador de telecomunicaciones. Puede ser optimizado para un bajo consumo de energía y también puede manejar aplicaciones urbanas e industriales inteligentes conectando millones de dispositivos. Las velocidades de datos pueden variar entre 0'3 Kbps y 50 Kbps. Las señales pueden atravesar obstáculos y viajar a través de largas distancias gracias a los chips LoRa y a una estrategia de espectro ensanchado que puede transmitir a través de varias frecuencias y varias velocidades de datos.

Resultará importante analizar a fondo los protocolos de comunicación IoT existentes, teniendo en cuenta que se pueden combinar varias tecnologías diferentes en una misma infraestructura, para elegir la que mejor se adapta en cada caso.

El internet de las cosas sigue creciendo y los dispositivos que requieren conectividad a internet verdaderamente de extremo a extremo podrán utilizar iPv4 o IPv6.

IPv6 es una actualización del protocolo original de Internet (el Protocolo de Internet o Protocolo IP), que soporta todas las comunicaciones a través de internet. IPv6 es necesario debido a que Internet se está quedando sin direcciones IPv4 originales. Mientras que IPv4 puede soportar 4300 millones de dispositivos conectados a Internet, IPv6 soporta 2128 direcciones, por lo que, a efectos prácticos, es inagotable. Esto representa alrededor de 3'4 x 1038 direcciones, que satisfacen sobradamente la demanda de los 100 mil millones de dispositivos de la IoT que se estima entrarán en servicio.

Los principales desafíos para los desarrolladores del IoT es que IPv6 no es interoperable con IPv4 en forma nativa y que la mayor parte del software de bajo costo fácilmente disponible para embeber dispositivos de IoT solo implementa IPv4. A pesar de esto, muchos expertos creen que IPv6 es la mejor opción de conectividad y permitirá que IoT realice su potencial.

2.1.4 Redes de Sensores

Las redes de sensores inalámbricas (WSN, Wireless Sensor Networks) están formadas por dispositivos autónomos, distribuidos a lo largo de un área de interés y cuyo objetivo es monitorizar parámetros físicos o ambientales tales como temperatura, sonido, vibraciones, presión, movimiento o agentes contaminantes. Se considera una de las tecnologías clave para implementar el Internet de las cosas.

A propósito de la industria 4.0 expertos y desarrolladores de soluciones IoT estiman que vivimos la era de los sensores y las redes. De hecho, estos componentes son los que definen el internet

industrial de las cosas o IIoT.

Brevemente, el IIoT abarca el conjunto de herramientas y máquinas que se conectan a la red y comparten data mediante sensores y aplicaciones. Por tal razón, estos instrumentos se consideran Smart devices, ya que permiten procesar los datos obtenidos en toda la cadena de suministro con plataformas analíticas.

Hoy existen inumerables sensores IoT de uso industrial para automatización de procesos y logística, medición de flujo, identificación, control de movimientos y seguridad, entre otros. Pero en el corto plazo se estima una evolución crucial en herramientas como:

1. **Sensor para AGVs.**

   Una innovación reciente en este sentido es la integración de sensores avanzados y cámaras en estos vehículos para detectar su velocidad y dirección.

2. **Sistemas de visión.**

   Es decir, cámaras 2D integradas a soluciones informáticas utilizadas en inspecciones de calidad, que incluso son empleables para la selección de robots. Sus versiones 3D se utilizan para captar y analizar piezas.

3. **Sensores inteligentes.**

   Estos dispositivos están superando sus límites de aplicación para la detección y medición. Estos datos pueden compartirse con tecnologías de mayor nivel como MES (Sistema de Ejecución de manufactura) o ERPs (Sistemas de planificación de recursos)

2.1.5 Futuro de IoT

La tecnología IoT evoluciona de manera acelerada y se involucra cada vez más en nuestra cotidianidad. Aunque ni siquiera expertos y analistas en esta materia se ponen de acuerdo sobre la cantidad de objetos conectados actualmente y los que habrá en 2025 sí resulta acertado predecir que los proyectos de startups en IoT captarán un mayor interés de las entidades de capital riesgo en el corto plazo.

En Davos 2020 se ha celebrado la 50 edición del World Economic Forum en el que, un año más, los principales líderes del mundo se han dado cita para debatir sobre la actualidad y para identificar las tendencias que afectarán a personas y negocios en los próximos años.

Según el secretario general de las naciones unidas, Antonio Gutiérrez, “el estado del mundo actualmente se define con dos palabras: Incertidumbre e Inestabilidad”. En este contexto, este año han destacado el cambio climático, el crecimiento económico responsable y las tensiones comerciales. Entre las principales Conclusiones y Resultados del World Economic Forum 2020 se han desarrollado iniciativas para fomentar una cuarta revolución industrial cohesionada y sostenible en ámbitos como las tecnologías emergentes o la ciberseguridad. En lo relativo al IoT se ha presentado un protocolo de actuaciones para acelerar la adopción exitosa de las tecnologías industriales del internet de las cosas por parte de pequeñas y medianas empresas. De igual manera se han presentado Protocolos de gestión para acelerar el impacto de la Cuarta Revolución

Industrial, así como los Principios de Seguridad en Internet.

Estas iniciativas hacen pensar que se está fomentando el conocimiento y la confianza del consumidor en los ecosistemas IoT además de impulsar una implementación más inclusiva de 5G y conectividad de próxima generación, trabajando en las 5 áreas temáticas clave expuestas en el marco contextual que representan hoy los desafíos a superar para el adecuado y prometedor desarrollo del IoT.

Por todo ello, resulta pertinente afirmar que los avances en comunicación M2M y M2P contribuirán de manera notable a la optimización de la productividad y a la mejora de nuestra calidad de vida.

2.2 Desarrollo de APPs

Hoy en día, el mercado de las aplicaciones móviles se está expandiendo rápidamente a medida que nuestra sociedad depende cada vez más de la tecnología digital y, más concretamente, de los smartphones. La gran demanda de aplicaciones móviles hace que este mercado sea altamente competitivo.

La creación de una aplicación móvil es un proceso costoso ya que requiere tiempo y experiencia.

Figura 2.2.a Desarrollo de APPs. Fuente: fiverr.com

Las aplicaciones -también llamadas APPs- están presentes en los teléfonos desde hace tiempo, una aplicación no deja de ser un software. Las aplicaciones son para los móviles lo que los programas son para los ordenadores de escritorio. Actualmente encontramos aplicaciones de todo tipo, forma y color, pero en los primeros teléfonos estaban enfocadas en mejorar la productividad personal: se trataba de alarmas, calendarios, calculadoras y clientes de correo. Hubo un gran cambio con la aparición en el mercado del iPhone ya que se generaron nuevos modelos de negocio que hicieron de las aplicaciones algo rentable tanto para desarrolladores como para los mercados de aplicaciones como App Store, Google Play y Windows Phone Store (actualmente en desuso). También mejoraron las herramientas de las que disponían diseñadores y programadores para desarrollar apps, facilitando la tarea de producir una aplicación y lanzarla al mercado.

Las aplicaciones comparten la pantalla del teléfono con las webs móviles, pero mientras que las primeras tienen que ser descargadas e instaladas antes de usar, a una web puede accederse simplemente usando internet y un navegador; sin embargo, no todas pueden verse correctamente desde una pantalla generalmente más pequeña que la de un ordenador de escritorio: las que se adaptan especialmente a un dispositivo móvil se llaman “Web Responsives” y son ejemplo del diseño líquido, ya que se puede pensar en ellas como un contenido que toma la forma del contenedor.

Figura 2.2.b El diseño responsive se adapta dependiendo del dispositivo donde es visualizado. Fuente: fiverr.com

Quienes cuentan ya con una web responsive pueden plantearse la necesidad de diseñar una aplicación en función de entender los objetivos de negocio y las características que diferencian las aplicaciones de las webs (las aplicaciones suelen ofrecer una mejor experiencia de uso, evitando tiempos de espera excesivos y logrando una navegación más fluida ya que pueden verse aun cuando se está sin conexión a internet y pueden acceder a ciertas características de hardware del teléfono).

Antes de centrarnos en la elección de la tecnología, que será el punto fundamental de este apartado, introducimos lo que podríamos denominar los pasos básicos aplicables cuando nos enfrentamos al desarrollo de una APP.

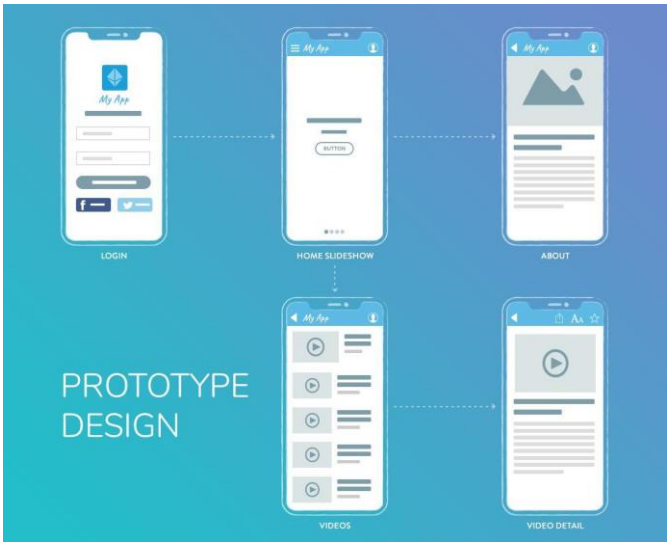


Figura 2.2.c Desarrollo APPs Protype Design. Fuente: fiverr.com

1. **Definición de objetivos.**

   La idea de una APP puede surgir en cualquier momento y en cualquier ámbito de aplicación. Podemos abarcar desde la transformación de una página web a una app, recepción de ayuda para tareas cotidianas, entretenimiento o centrarnos en productos de marketing o un negocio (en todo o en parte). Sea cual fuere la idea, es muy importante entender lo que puede aportar, en términos de resultados ya que marcará el camino a seguir para el diseño y el desarrollo de funcionalidades, así como una mayor visión de las características de la app. Preguntas que nos pueden ayudar en este apartado serían:

   ¿Cuáles son los beneficios de tu proyecto?
   ¿Cómo va a ayudar/entretener al usuario final?
   ¿Ves el producto evolucionar en el futuro?
   ¿Cómo quieres que los usuarios interaccionen con él?

   Desde un punto de vista de Marketing es preciso considerar como tu producto va a entrar en el mercado y cuales son las etapas claves que se quieren alcanzar en términos de desarrollo de la base de usuarios. Desarrollar una comunidad potente es uno de los factores claves del éxito de los productos digitales y por esa razón no hay que despreciar esta etapa. Habría que preguntarse:

   ¿Cuál será el público objetivo?
   ¿Cómo lo alcanzarás?
   ¿Cómo convertirás usuarios potenciales?
   ¿Cómo mantedrás tus usuarios para que se queden comprometidos?

   El tercer paso será estudiar la financiación. Las siguientes preguntas pueden ayudarnos:

   ¿Cómo vas a financiar el proyecto?

¿Es un producto con o sin fin de lucro?
¿Cuáles son los ingresos esperados (publicidad, suscripciones, pago por uso, etc)?

Tener definidos correctamente estos tres apartados en un el plan de objetivos resultará práctico, nos ayudará para planificar correctamente el desarrollo y para marcar la diferencia con nuestra APP, aunque obviamente, los objetivos marcados al principio puedan evolucionar.

2. Búsqueda del mercado objetivo.

Una vez madurada la idea nos centraremos en la ejecución, validación de hipótesis iniciales y ajuste de objetivos llegado el caso. ¿Qué información nos puede ayudar?

1. **Los datos demográficos.**
   Por ejemplo: edad, situación financiera, ubicación y conectividad de usuarios, así como tipo de dispositivos que utilizan.

2. **Tendencias de comportamiento.**
   ¿Prefieren usar web o apps móviles? ¿Están dispuestos a descargar tu app?, ¿Dónde buscan información?.

Es habitual crear un perfil tipo del usuario objetivo que nos aportará una información muy valiosa para desarrollar nuestro producto y responder a las necesidades del usuario.
También es importante tener en cuenta las tendencias habitualmente aplicables con carácter universal (independientemente del proyecto):

i. Deseo decreciente de descargar apps en dispositivos.
ii. Muy poca paciencia ante procesos lentos de descargas.
iii. Cada vez menor tolerancia a la falta de seguridad
iv. Fuerte valor otorgado a la utilidad de un producto.
v. Supresión de elementos que no aportan valor añadido.

3. Definición del concepto.

Cuando hayamos finalizado los primeros análisis, sacado las primeras conclusiones y establecido los objetivos de partida podremos entender con mayor precisión cuál es la audiencia a la que nos dirigimos. Ahora toca entender cuáles serían las funcionalidades tangibles que constituirán el modelo de negocio y como vamos a alcanzarlo. Los objetivos/exigencias más comunes son:

1. **Monetizar.**

   i. ¿podrías tener acceso pagado a tu producto?
   ii. ¿podría ser gratuito, pero incluir otras opciones de monetización, como la publicidad?
   iii. ¿podría el modelo freemium ser una opción?

2. **Hacer correr la voz.**

   i. ¿cuáles serán los esfuerzos que aportar para asegurar la visibilidad de tu app?
   ii. ¿cómo puedes fomentar tus usuarios a compartir tu contenido?
   iii. ¿cuáles son los obstáculos que podrías eliminar y que podrían obstaculizar el acceso directo a tu mensaje?

3. **Reforzar un negocio/proyecto existente.**

   i. ¿cuáles son las comodidades/valores añadidos que le faltan a tus usuarios?
   ii. ¿cuáles son las razones corrientes por las que los clientes desaparecen y como resolverlas gracias a una app?
   iii. ¿Qué impide a tus usuarios descubrirte y darte una oportunidad?

Las respuestas a estas preguntas nos ayudarán a organizar y estructurar nuestra app, incluyendo funcionalidades claves y los beneficios buscados (para el desarrollador y para el usuario).

Una vez que tengamos todas las ideas y conceptos madurados, será más fácil determinar la importancia de cada funcionalidad, así como la arquitectura de la app. Es un buen ejercicio para entender cómo realizar el mapa de ruta del desarrollo de la app.

4. **Elección de Tecnología.**

   Nos centraremos en este paso en el apartado 2.2.1.

Tras la elección de la tecnología a aplicar, quedaría la exploración de las distintas opciones disponibles, la gestión del diseño, contenido y complementos y, para finalizar, la prueba, publicación y la gestión de mejoras y actualizaciones.

2.2.1 Elección de tecnología: Nativo vs híbrido

Cuando se decide crear una aplicación móvil, nos enfrentamos inmediatamente a la elección de "desarrollo de aplicaciones nativas o híbridas" y a la búsqueda de formas eficientes de implementar la idea. A continuación, vamos a hacer una revisión de aplicaciones nativas, aplicaciones híbridas, diferencias entre ellas e influencia de cada tipo de tecnología en el proceso de desarrollo general y el rendimiento. Analizaremos las tecnologías utilizadas para las plataformas iOS y Android en función de la diferencia de aplicaciones nativas e híbridas para poder elegir la más adecuada.

La aplicación móvil se considera una de las herramientas comerciales más dinámicas, se ha convertido en el nuevo estándar para establecer una conexión con los clientes. Existen distintas maneras para implementar aplicaciones para dispositivos móviles, pero en este trabajo fin de grado nos centraremos en aplicaciones nativas e híbridas, las cuales cuentan con ventajas y desventajas que se ajustan a ciertos factores a la hora de implementación en cuanto a tiempo, desarrollo, performance e inversión a realizar.

2.2.1.1 Aplicaciones Nativas

Figura 2.2.1.1.a Figura Aplicaciones Nativas

Las aplicaciones nativas son aquellas que han sido desarrolladas con el software que ofrece cada sistema operativo a los programadores, llamado genéricamente Software Development Kit o SDK.

Las apps para sistemas iOS son desarrolladas en lenguaje Objetive C y Swift.

Las apps para sistemas Android son desarrolladas en lenguaje Java y Kotlin.

Este tipo de apps se descargan e instalan desde las tiendas de aplicaciones sacando buen partido de las diferentes herramientas de promoción y marketing de cada una de ellas.

Las aplicaciones nativas se actualizan frecuentemente y en esos casos, el usuario debe volver a descargarlas para obtener la última versión, que a veces corrige errores o añade mejoras.

Figura 2.2.1.1.b Native App Development. Fuente: existek

Desarrollamos a continuación las tecnologías utilizadas para el desarrollo de aplicaciones nativas:

1. **Tecnologías para el desarrollo de aplicaciones nativas de iOS.**
    i. **Objective-C.**
        Este lenguaje de programación es conocido por su gran experiencia en desarrollo, bibliotecas disponibles y un vasto grupo de expertos. Objective-C es un framework maduro, que también obtiene compatibilidad con otras tecnologías de programación. A pesar de esos beneficios, nos damos cuenta de que aparecen nuevos idiomas y, en algún momento, no tendrá la capacidad de admitir todas las funciones más recientes.

    ii. **Swift.**
        Es un marco de programación relativamente nuevo introducido por Apple que se ha convertido en la alternativa para crear aplicaciones nativas de iOS. Los desarrolladores han señalado su rendimiento más rápido y es más fácil de aprender y trabajar. Está en constante desarrollo, por eso puede que le falten algunos componentes. Sin embargo, se dice que podría reemplazar a Objective-C en el futuro.

2. **Tecnologías para el desarrollo de aplicaciones nativas de Android.**

   i. **Java.**

   El lenguaje de programación que no solo se usa principalmente para aplicaciones móviles de Android, sino también para otros fines. Gran parte del desarrollo web y de escritorio se basa en Java. Se ha convertido en un sistema especial de herramientas, los desarrolladores tienen acceso a una poderosa biblioteca. Ayuda a simplificar el proceso de programación general. Sin embargo, las aplicaciones en Java requieren más memoria y funcionan más lentamente en comparación con otros marcos.

   ii. **Kotlin.**

   Es un lenguaje de programación desarrollado específicamente para trabajar con Java y Java Virtual Machine. Por eso, su uso es compatible y aprobado por Google para el desarrollo de aplicaciones de Android. Una de las principales ventajas de Kotlin sobre Java normal es que su interfaz de tipo permite trabajar con sintaxis más corta. Este hecho reduce el tiempo de programación necesario para realizar una aplicación para Android. Ahora se incluye como alternativa al compilador estándar de Java para Android Studio. Expedia, Square, Pinterest y Flipboard pueden mencionarse entre los ejemplos más demostrativos de las empresas que utilizan Kotlin para sus aplicaciones de Android.

Todas las tecnologías mencionadas se caracterizan por las siguientes pecualiaridades:

1. **Ventajas del desarrollo de aplicaciones nativas.**

   i. **Mejor rendimiento.**
   Cuando analizamos el rendimiento de las aplicaciones híbridas frente a las nativas, queda claro que las aplicaciones nativas serán más rápidas. Están construidos con un marco que es nativo de la plataforma.

   ii. **Protección de datos.**
   Es mucho más fácil hacer que la aplicación nativa sea segura. Esa es la ventaja que muchas empresas están interesadas en brindar a sus clientes, especialmente en el sector empresarial, fintech y aplicaciones con datos sensibles.

   iii. **Funcionalidad general.**
   la aplicación tendrá la importante capacidad de conectar las funciones de hardware del dispositivo y diferentes bases de datos. No se necesitan complementos ni herramientas adicionales.

   iv. **Experiencia del cliente.**
   Definitivamente tendrán un alto rendimiento. Además, pueden trabajar en modo fuera de línea, lo que sigue siendo un problema para los ejemplos de aplicaciones híbridas.

   v. **Comprensión.**
   Los desarrolladores ya conocen todas las fortalezas y debilidades del uso de tecnologías bien establecidas. Ayudarán a encontrar el enfoque correcto para recibir resultados finales deseables.

2. **Desventajas del desarrollo de aplicaciones nativas.**

   i. **Consume tiempo y dinero.**
   Definitivamente se requiere tiempo para crear un software complejo. La distribución de usuarios en dos plataformas principales duplica la cantidad de trabajo y pruebas necesarias para mantener dos aplicaciones separadas para iOS y Android en funcionamiento.

   ii. **Base de código distribuida**.
   Tener algunas funciones no disponibles para iOS o Android es una práctica infame que tiene su lugar incluso en 2020. Esto sucede debido a limitaciones

en el presupuesto o restricciones de la plataforma. A veces, las aplicaciones de la App Store pueden abandonarse durante años mientras la versión de Android recibe actualizaciones periódicas y viceversa. Por el contrario, la aplicación híbrida anima a los desarrolladores a abordar la interfaz de usuario y las características de una manera más reflexiva e introducir solo características que pueden funcionar en ambos sistemas operativos.

2.2.1.2 Aplicaciones Web

La base de programación de las aplicaciones web (también llamadas *webapps)* es el HTML, que conjuntamente con JavaScript y CSS, son las herramientas ya conocidas para los programadores web.

En este caso no se emplea un SDK, lo cual permite programar de forma independiente al sistema operativo en el cual se usará la aplicación. Por eso, estas aplicaciones pueden ser fácilmente utilizadas en diferentes plataformas sin mayores inconvenientes y sin necesidad de desarrollar un código diferente para cada caso particular.

Las aplicaciones web no necesitan instalarse, ya que se visualizan usando el navegador del teléfono como un sitio web normal. Por esta misma razón, no se distribuyen en una tienda de aplicaciones, sino que se comercializan y promocionan de forma independiente.

Al tratarse de aplicaciones que funcionan sobre la web, no es necesario que el usuario reciba actualizaciones, ya que siempre va a estar viendo la última versión. Pero, a diferencia de las apps nativas, requieren de una conexión a Internet para funcionar correctamente.

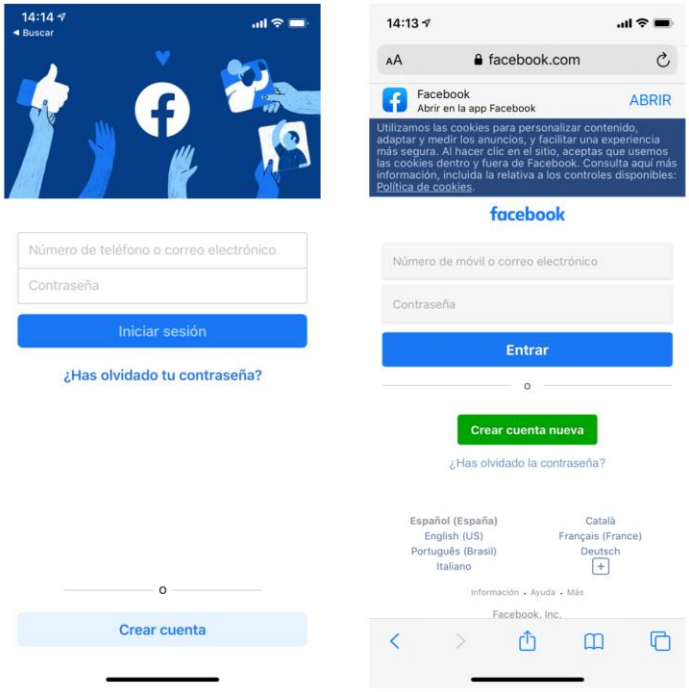


Figura 2.2.1.2 Facebook cuenta tanto con una webapp como con una app nativa

Adicionalmente, tienen algunas restricciones e inconvenientes en factores importantes como gestión de memoria y no permiten aprovechar al máximo la potencia de los diferentes componentes de hardware del teléfono.

Las aplicaciones web suelen tener una interfaz más genérica e independiente de la apariencia del sistema operativo, por lo que la experiencia de identificación del usuario con los elementos de navegación e interacción suele ser menor que en el caso de las nativas.

2.2.1.3 Aplicaciones Híbridas

Figura 2.2.1.3.a Tecnologías que componen la base para la arquitectura de aplicaciones híbridas

Las aplicaciones híbridas deben su nombre a la combinación de lenguajes de programación que pueden ser ejecutados en los distintos móviles. El desarrollo híbrido se basa, por tanto, en el funcionamiento multiplaforma.

Suelen tener una base de lenguaje HTML5, CSS y marcos de Javascript, los cuales son lenguajes también usados para el desarrollo de web apps.

Es habitual utilizar framework que nos ayuden a facilitar el trabajo del desarrollo y además la creación de los ejecutables para cada una de las plataformas.

Podríamos decir que las aplicaciones híbridas son sitios web empaquetados en contenedores que emulan el comportamiento del software para que cada plataforma funcione y se vea de forma natural.

Desarrollamos a continuación las tecnologías utilizadas para el desarrollo de aplicaciones híbridas. Hay tres marcos populares para el desarrollo de aplicaciones híbridas:

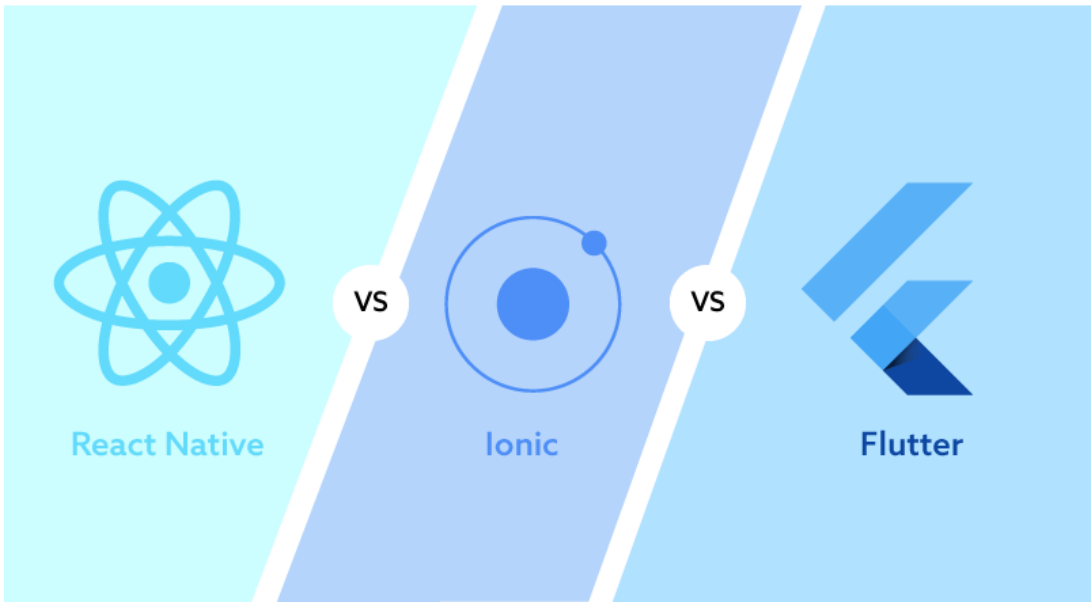


Figura 2.2.1.3.b Frameworks más usados para el desarrollo de aplicaciones híbridas. Fuente: codeburst.io

1. **React Native.**

   Facebook ha creado esta tecnología de código abierto para la compatibilidad multiplataforma. Las UI (User Interface) simplificadas han aumentado

considerablemente el rendimiento específicamente al recargar. React Native se utiliza principalmente debido al corto tiempo de desarrollo. Los ingenieros de software tienen acceso a unidades listas para aplicar, sin embargo, el marco puede carecer de ciertos componentes. Tanto Facebook como la comunidad proporcionan actualizaciones con regularidad.

2. **Ionic.**

   Este marco produce aplicaciones móviles con tecnologías web estándar como JavaScript, CSS, HTML, Angular, etc. Los desarrolladores también tienen muchos componentes de interfaz de usuario accesibles que son fáciles de manejar. El tiempo de desarrollo es tan bueno como con todos los demás marcos para ejemplos de aplicaciones híbridas. A pesar de todos los beneficios, el mantenimiento puede resultar más complicado. Ionic otorga muchos complementos, por lo que cuando surja el problema, es posible que requiera ajustes manuales adicionales.

3. **Flutter.**

   Flutter es un SDK de código fuente abierto de desarrollo de aplicaciones móviles creado por Google. Al ser el Framework elegido para el desarrollo de este TFG, lo trataremos en profundidad en el capítulo 3, Materiales y Métodos.

Todas las tecnologías mencionadas se caracterizan por las siguientes pecualiaridades:

1. **Ventajas del desarrollo de aplicaciones híbridas.**

   i. **Base de código única.**
   Estas aplicaciones son las preferidas por empresas y desarrolladores, ya que pueden funcionar en ambas plataformas. No será necesario crear dos códigos separados para iOS y Android debido a la funcionalidad común del código.

   ii. **Menor costo.**
   La empresa puede lograr grandes ahorros al desarrollar una aplicación móvil mientras se dirige a los usuarios en diferentes plataformas. Tiene la necesidad de contratar un equipo con cierta experiencia en ambos, pero solo una buena experiencia en desarrollo web sería suficiente. Tus gastos serán casi los mismos que construir solo uno diseñado para funcionar en todas partes.

   iii. **Más simple de construir y probar.**
   El equipo alcanzará el resultado esperado más rápido. No tratan con cada plataforma por separado. El código se crea una vez y el tiempo de prueba también se reducirá.

   iv. **Más fácil de mantener.**
   Todos los cambios y actualizaciones necesarios se mantendrán simultáneamente en ambas plataformas. No solo es conveniente para los desarrolladores, sino también para los usuarios. Es posible solucionar muchos problemas desde el lado del servidor, y el usuario simplemente obtendrá las actualizaciones automáticamente.

   v. **Tiempo de entrega más rápido.**
   Como mencionamos anteriormente, no es necesario tener dos equipos de iOS y Android o un gran equipo multifuncional, solo necesita encontrar

un equipo relativamente pequeño de profesionales. Pueden crear un producto comercializado inteligentemente que resultará interesante para diferentes usuarios. Las aplicaciones híbridas se han recomendado a sí mismas como orientadas al contenido.

2. **Desventajas del desarrollo de aplicaciones híbridas.**

   i. **Eficiencia limitada.**
   El marco de la plataforma cross depende de los complementos que se conectarán con las funciones del dispositivo. A veces, los desarrolladores tienen que crearlos manualmente para abordar la función particular del dispositivo.

   ii. **Conexión a internet.**
   En cuanto a la eficiencia del software, existe una diferencia significativa entre la aplicación nativa y la aplicación híbrida. Los marcos multiplataforma requieren una conexión a internet regular. Algunas funciones no estarán disponibles cuando el usuario no tenga conexión.

La principal ventaja de las aplicaciones híbridas es la facilidad de desarrollo de éstas debido a que se realiza una única implementación y que luego tiene como salida los ejecutables para los distintos sistemas operativos de dispositivos móviles, con el consiguiente ahorro de costes que implica a efectos de desarrollo.

La principal desventaja está relacionada con el hecho de que al ser usados lenguajes no nativos y ser una combinación de lenguajes más genéricos, se limita el uso de recursos de los dispositivos, lo cual aumenta la complejidad del desarrollo de ciertas aplicaciones si éstas requieren hacer usos del hardware extra de los móviles.

2.2.1.4 Conclusiones Nativas vs Híbridas

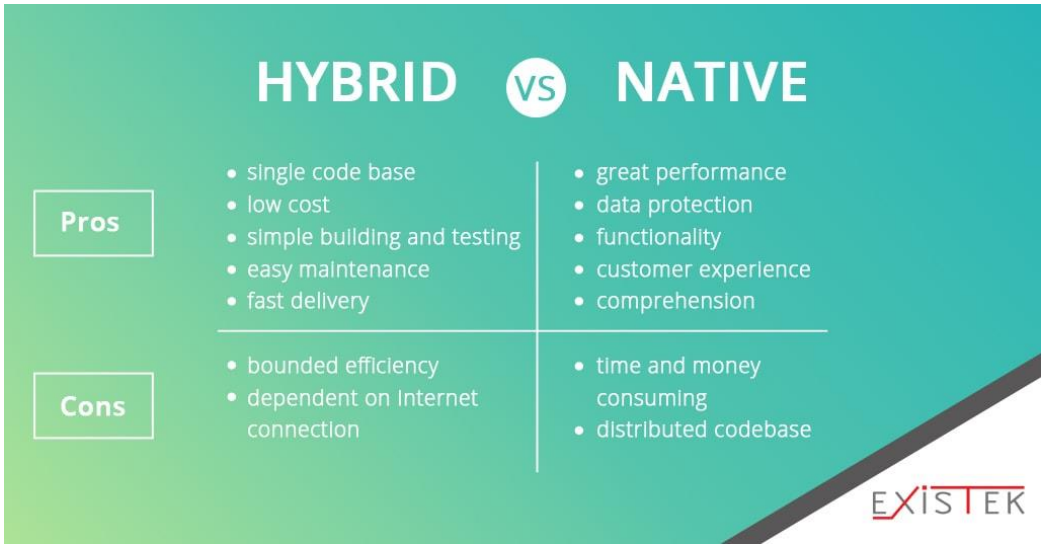


Figura 2.2.1.4.a Pros y contras de las tecnologías nativas e híbridas. Fuente: existek

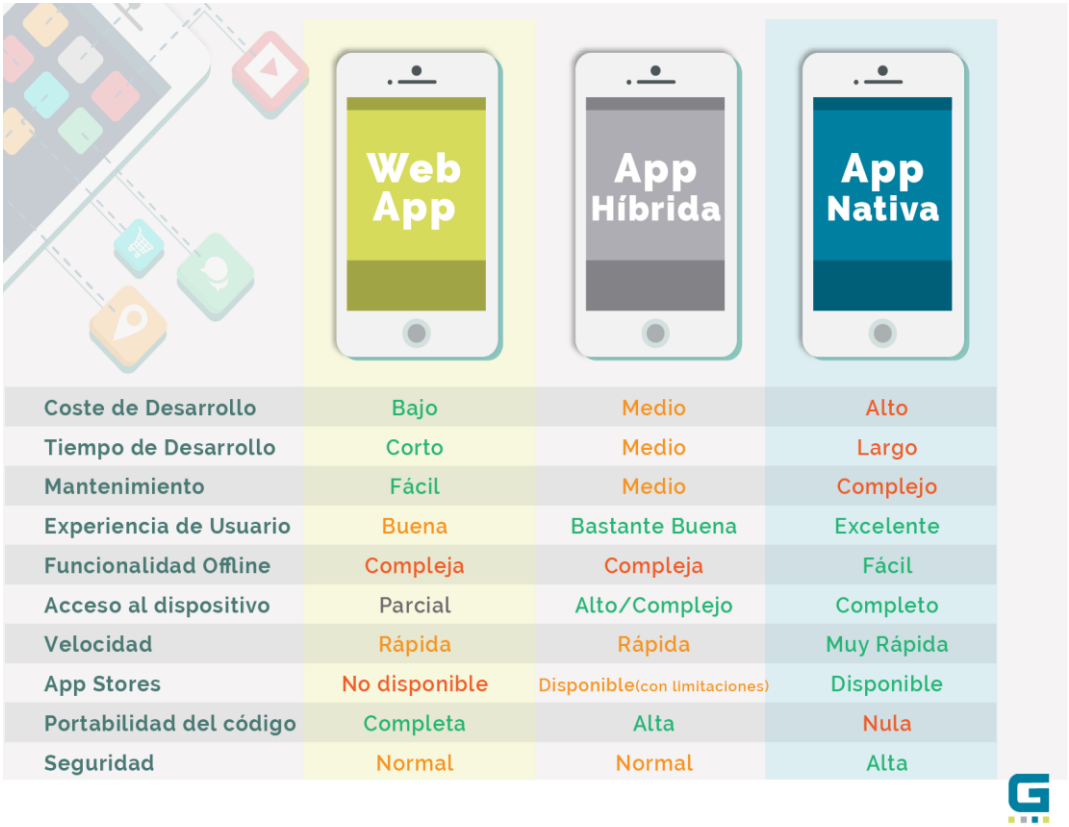


Figura 2.2.1.4.b Comparativa de aspectos claves entre tipos de desarrollo de aplicaciones. Fuente: gsoft.es

Las figuras anteriores (2.2.1.4.a y 2.2.1.4.b) resumen las principales conclusiones en cuanto a desarrollo utilizando aplicaciones nativas vs híbridas.

Al crear una aplicación, antes de elegir la tecnología a aplicar, se debe tener en cuenta cuales van a a ser las funciones que contendrá y preguntarse si es conveniente hacerla nativa donde tendrá un mejor performance y adaptación a los dispositivos con mayor coste de desarrollo o si se desea ahorrar el coste de desarrollo aplicando tecnología híbrida, pero teniendo en cuenta que podría tener alguna limitación con ciertos recursos de los dispositivos móviles.

2.2.2 Lenguajes de Programación

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (si es necesario) y se mantiene el código fuente de un programa informático se le llama programación.

El lenguaje de programación es el puente para traducir el pseudocódigo legible por el humano a instrucciones legibles por la máquina. Sirve, en primer lugar, para poder programar. Dado que existen muchos lenguajes de programación distintos, podríamos sacar en conclusión que cada uno sirve para algo diferente, y es más o menos a´si. Por ejemplo, HTML y Javascript son lenguajes que se usan en el campo del desarrollo y diseño web, junto con CSS. Sin embargo, C y

C++ son algo enfocado completamente al desarrollo del software. Desde una perspectiva empresarial, el lenguaje es lo que hace que el programador pueda hacer funcionar una web y construirla desde cero, o desarrollar una app para iOS y Android de forma híbrida, como es el objetivo del presente trabajo.

Se clasifican, principalmente, en lenguajes de alto nivel (próximo al lenguaje humano) y bajo nivel (próximo al lenguaje máquina).

A continuación, revisaremos una serie de clasificaciones que habitualmente se aplican a los lenguajes de programación.

Clasificación **(Según finalidad y herramientas)**	**Descripción**
Lenguaje Máquina	Es el más primitivo de los códigos y se basa en la numeración binaria, todo en 0 y 1. Este lenguaje es utilizado directamente por máquinas o computadoras.
Lenguaje de Programación de Bajo Nivel	Es un lenguaje de programación un poco más fácil de interpretar, pero puede variar de acuerdo a la máquina o computadora que se esté programando
Lenguaje de Programación de Alto Nivel	En esta categoría se encuentran los más utilizados. Se usan palabras del inglés lo cual facilita que una persona pueda intervenir más fácil que en los dos anteriores. Nos permiten escribir códigos de computadora usando instrucciones que se asemejen al lenguaje hablado cotidiano que luego se traducen al lenguaje máquina antes de que puedan ser ejecutados. Algunos lenguajes usan un compilador para realizar esta traducción y otros usan un intérprete.

Tabla 2.2.2.a Clasificación de Lenguajes de Programación según finalidad y herramientas

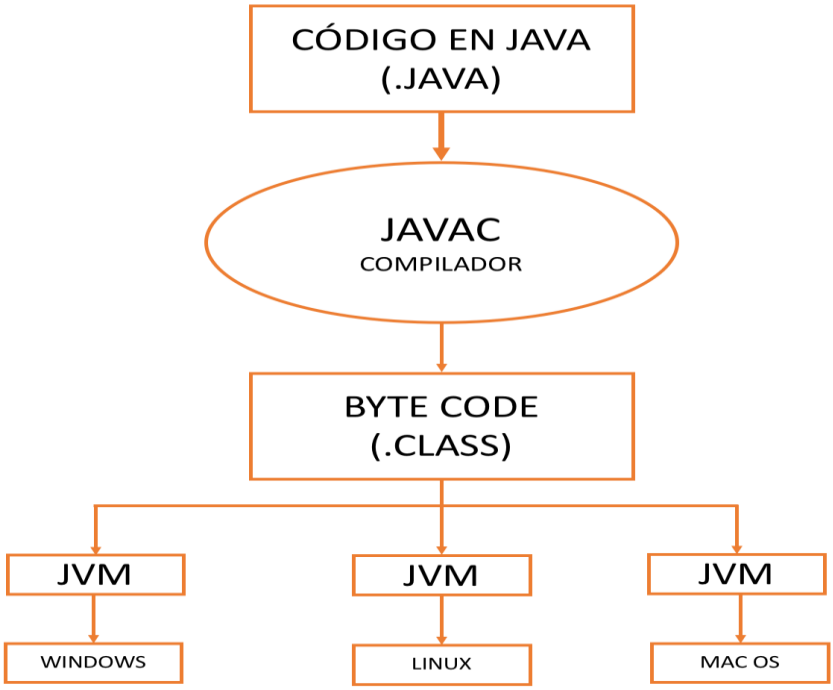


Figura 2.2.2.a Función del compilador. Fuente: profejavaoramas.blogspot.com

Clasificación (Según generación y orden cronológico)	**Observaciones**
1GL	El lenguaje máquina y el ensamblador
2GL	Encontramos los primeros lenguajes de programación de alto nivel, ejemplos de ellos son FORTRAM, COBOL.
3GL	En esta generación encontramos los lenguajes de programación de alto nivel imperativo, pero mucho más utilizados y vigentes en la actualidad (ALGOL 9, PL/I, PASCAL, MODULA, C)
4GL	Más cercanos a la época actual, es común encontrarlos en aplicaciones de gestión, manejo de bases de datos y scripts. (NATURAL, SQL). Idiomas que consisten en declaraciones similares a las declaraciones en un lenguaje humano.
5GL	Estos son los más avanzados y fueron pensados para la inteligencia artificial y para el procesamiento de lenguajes naturales (LISP, PROLOG). Contienen herramientas visuales para ayudar a desarrollar un programa. Un buen ejemplo es Visual Basic.

Tabla 2.2.2.b Clasificación de Lenguajes de Programación según generación y orden cronológico

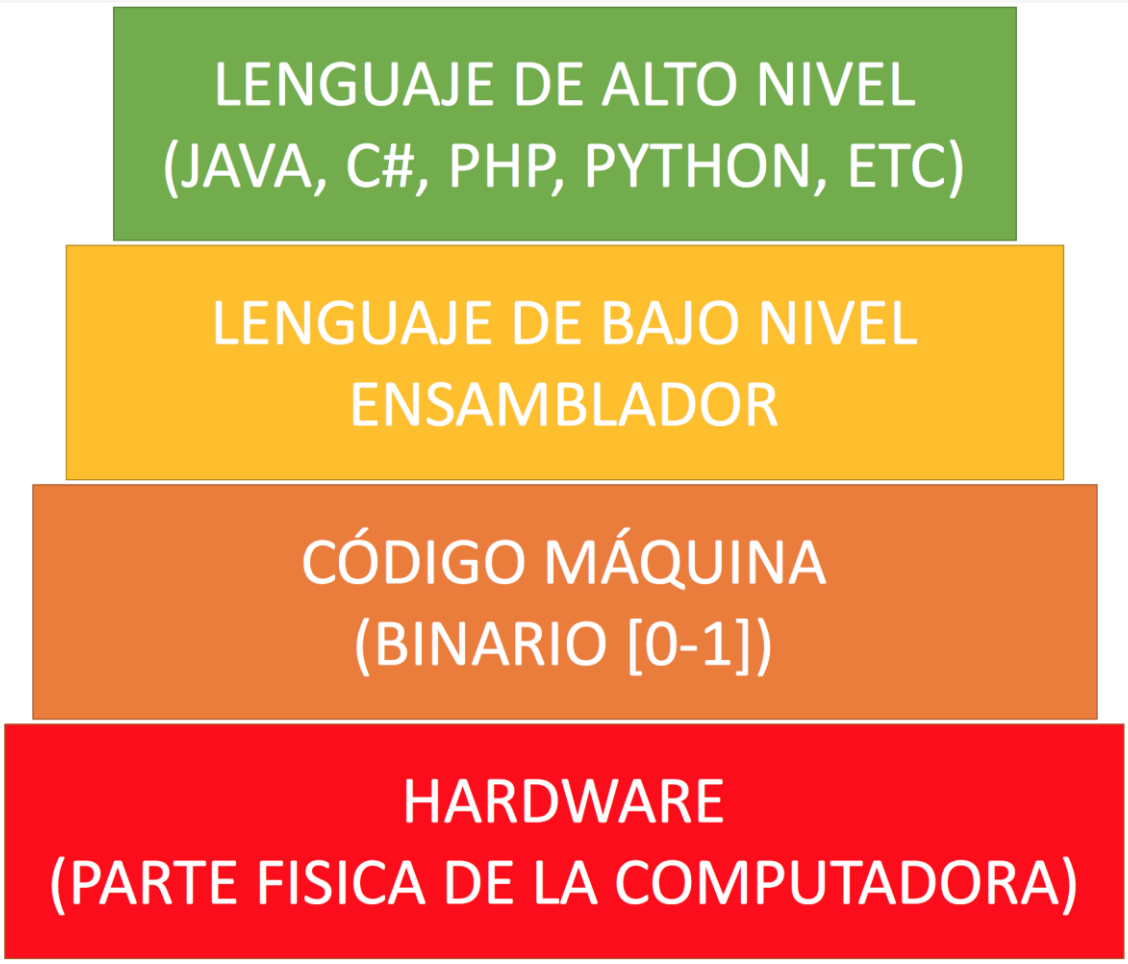


Figura 2.2.2.b Alto nivel – bajo nivel – codigo máquina. Fuente: conogasi.com

Existen muchos lenguajes de programación, pero destacaremos los más importantes de acuerdo con el índice TIOBE (estudio realizado por una empresa holandesa que analiza los lenguajes de programación más usados a nivel mundial). En el Top 6 de esa lista están:

Lenguaje de programación	Descripción
Java	Es el más utilizado actualmente y se debe a su simplicidad y legibilidad. Usado por más de 9 millones de usuarios, presente en miles de millones de dispositivos, ocupa el 15% del global mundial en cuanto a utilización. Se ha mantenido en las primeras posiciones durante el siglo XXI.
C	Surgió en los años 70 con el nombre de lenguaje "B". Sería el segundo lenguaje más usado a nivel mundial y se puede ejecutar en la mayoría de los sistemas operativos. Comunmente utilizado en las aplicaciones de escritorio.
Python	Un lenguaje de programación multiplataforma y multiparadigma, que también tiene un propósito general. Esto significa que soporta la orientación a objetos, la programación imperativa y funcional. Su sencilles, legibilidad y similitud con el idioma inglés lo convierten en un gran lenguaje, ideal para principiantes.
C++	Es una evolución del Lenguaje C. Está enfocado al desarrollo de aplicaciones y software más complejos que exigen, por ejemplo, una intervención visual. Los programas de diseño gráfico son un ejemplo del uso de este lenguaje de programación.
C#	Conocido también como "C Sharp" entre los programadores, también es una evolución de C y C++. Es un lenguaje de programación orientado a objetos. Creado en el año 2000, se destaca por su simplicidad, trabaja con aplicaciones framework .net como el visual studio de Windows. Cuenta con un 7% de uso a nivel mundial.
Visual Basic .NET	Ha tenido una rápida evolución en número de usuarios en los últimos años. Conocido por ser una herramienta mucho más amigable, que no exige tanto conocimiento como, por ejemplo, el C#.

Tabla 2.2.2.c Clasificación de Lenguajes de Programación según lista TIOBE (Top 6).

Les siguen:

7.	SQL	**8.**	PHP
9.	Ruby	**10.**	Rust
11.	TypeScript	**12.**	Swift
13.	Perl	**14.**	Go
15.	Kotlin	**16.**	Scheme
17.	Erlang	**18.**	Elixir
19.	Pascal	**20.**	Scala

Tabla 2.2.2.d Clasificación de Lenguajes de Programación según lista TIOBE (Top 7-20).

Los tipos de lenguaje de programación son vitales para el funcionamiento de todo lo que conocemos y que rodea internet en la actualidad.

2.2.3 Diseño de Aplicaciones

El proceso de diseño y desarrollo de una aplicación abarca desde la concepción de la idea hasta el análisis posterior a su publicación en las tiendas. Durante las diferentes etapas, diseñadores y desarrolladores trabajan de manera simultánea y coordinada.

En ocasiones, el diseño de una aplicación parte de una web previamente existente. En estos casos, la app tiene que tomar las funciones y contenidos que se han pensado para la web y adaptarlos a la pantalla y la interacción de un móvil. En otros casos, el diseño comienza desde cero y hay que decidirse por cuál de ellas empezar. Es aquí donde adquiere trascendencia el concepto de mobile First, que consiste en plantear el proceso de diseño teniendo en cuenta el móvil en primer lugar. Esto obliga a concentrase en lo esencial de un producto y poner el foco solo en lo que tiene sentido para este dispositivo.

Una vez diseñada la aplicación para el móvil, cabría plantearse cuál es la mejor forma de llevar lo hecho para el móvil a una pantalla de ordenador o cualquier otro dispositivo, que tienen usos diferentes y en el momento de adaptar el diseño habrá que tener en cuenta las características particulares de cada uno de ellos.

Además de las fases de diseño y desarrollo que se resumen en la figura inferior, habría que tener en cuenta los roles de coordinación entre las distintas fases, la participación del cliente y de los accionistas de la empresa.

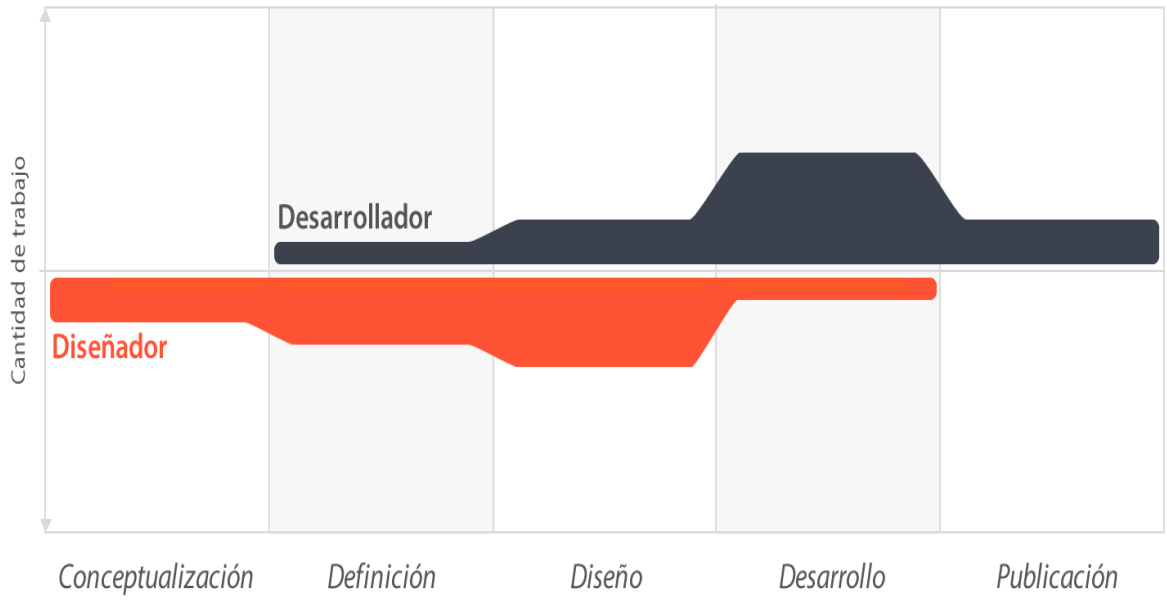


Figura 2.2.3 El proceso de diseño. Fuente: fiverr.com

Etapa del diseño	Observaciones
1. Conceptualización	El resultado de esta etapa es una idea de aplicación que tiene en cuenta las necesidades y problemas de los usuarios. La idea responde a una investigación preliminar y a la posterior comprobación de la viabilidad del concepto: • Ideación • Investigación • Formalización de la idea
2. Definición	En este paso el proceso se describe con detalle a los usuarios para quienes se diseñará la aplicación, usando metodologías como "personas" y "viaje del usuario". También aquí se sientan las bases de la funcionalidad, lo cual determinará el alcance del proyecto y la complejidad de diseño y programación de la app. • Definición de usuarios • Definición funcional
3. Diseño	En la etapa de diseño se llevan a un plano tangible los conceptos y definiciones anteriores, primero en forma de wireframes, que permiten crear los primeros prototipos para ser probados con usuarios, y posteriormente, en un diseño visual acabado que será provisto al desarrollador. • Wireframes. • Prototipos • Test con usuarios • Diseño visual
4. Desarrollo	El programador se encarga de dar vida a los diseños y crear la estructura sobre la cual se apoyará el funcionamiento de la aplicación. A partir de la versión inicial, dedica gran parte del tiempo a corregir errores funcionales para asegurar el correcto desempeño de la app y la prepara para su aprobación en las tiendas. • Programación del código • Corrección de bugs
5. Publicación	La aplicación, finalmente y como paso trascendental, es puesta a disposición de los usuarios en las tiendas. Luego se realiza un seguimiento a través de analíticas, estadísticas y comentarios de usuarios, para evaluar el comportamiento y desempeño de la app, corregir errores, realizar mejoras y actualizarla en futuras versiones. • Lanzamiento • Seguimiento • Actualización

Tabla 2.2.3.a Clasificación de Lenguajes de Programación según lista TIOBE (Top 7-20)

3 MATERIALES Y MÉTODOS

"Por norma, los hombres se preocupan más de lo que no pueden ver que de lo que pueden"

- Julio César -

Para el desarrollo completo de una aplicación, o lo que se conoce como "Full Stack App Development", debemos tener claro los siguientes aspectos: el diseño que va a adquirir ésta (basándonos normalmente en colores, fuentes, logos e imágenes corporativas), el tipo de desarrollo (híbrido o nativo) y por último en los lenguajes de programación que utilizaremos para front-end, back-end y BBDD.

Cabe destacar que gracias al enorme avance tecnológico que se da hoy en día, las posibilidades de elección en todos los bloques comentados anteriormente son prácticamente infinitas, por eso, para cada caso de uso, conviene hacer un buen estudio de mercado y decidir que tecnología viene mejor en cada trabajo.

3.1 Diseño Global

Este apartado se basará en una imagen que llamaremos "Maqueta raiz" y sobre la que explicaremos cada una de las elecciones que se han hecho en cada uno de los aspectos mencionados en la introducción de este bloque.

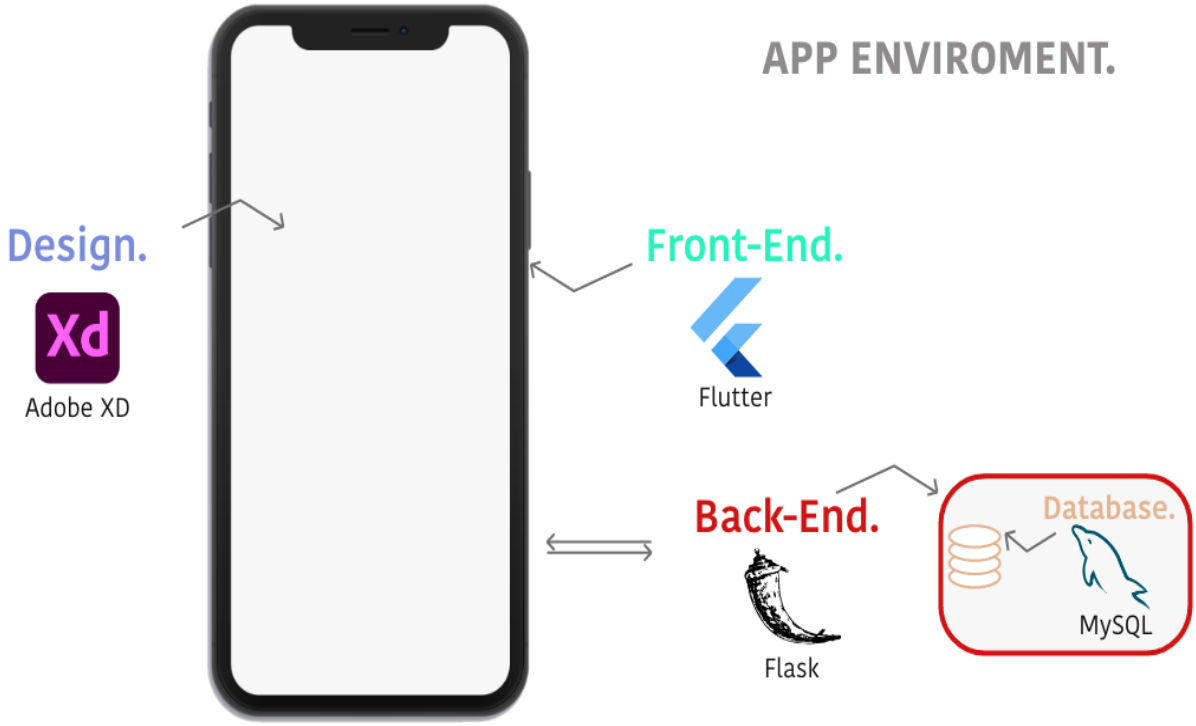


Figura 3.1.a Maqueta raiz de diseño global de la aplicación

Siguiendo el orden de la introducción del apartado empezaremos centrándonos en el software de diseño empleado para esta aplicación. Primero haremos una aproximación teórico-gráfica acerca de éste para posteriormente mostrar toda la evolución en lo que al diseño se refiere.

Adobe XD es uno de los softwares para maquetación y diseño más utilizados y con mayor comunidad que existe en el sector. Se trata de un editor de gráficos vectorizados desarrollado y publicado por Adobe Inc para diseñar y crear un prototipo de la experiencia del usuario para páginas web y apps móviles. Posee un software disponible tanto para Windows como para MacOS lo que le hace una alternativa bastante competitiva.

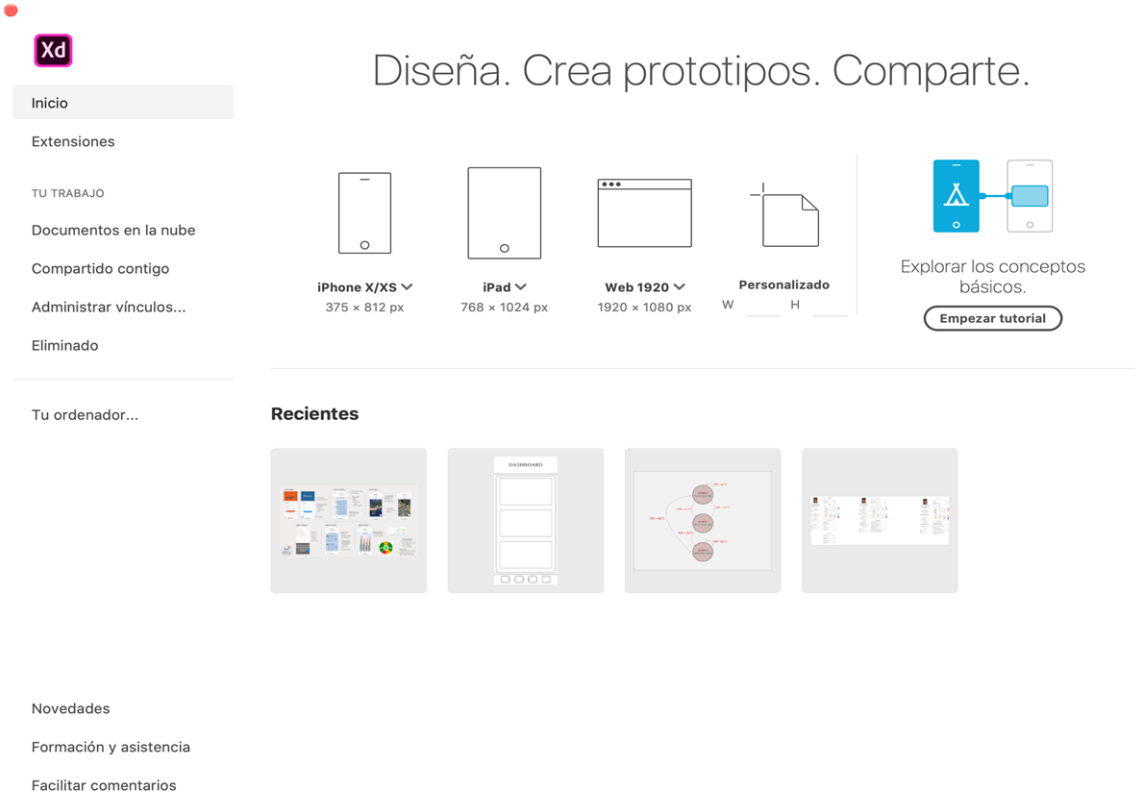


Figura 3.1.b Pantalla principal Adobe XD

Como podemos ver en la figura 3.1.b y ya hemos mencionado antes, Adobe XD nos aporta todo lo necesario para la maquetación en cualquier tipo de plataforma, además, mantiene siempre una versión en la nube de nuestros diseños y podemos acceder a ellos desde cualquier dispositvo.

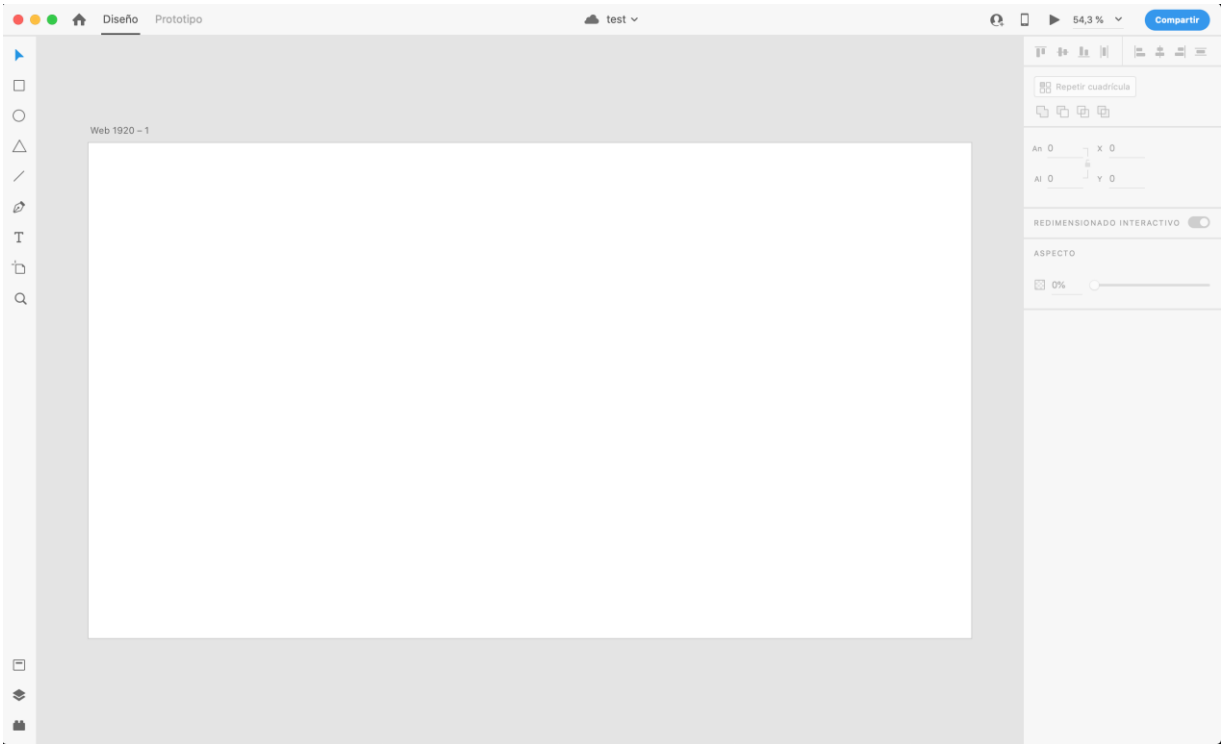


Figura 3.1.c Pantalla diseño Adobe XD

En la figura 3.1.c, vemos lo que sería el dashboard de nuestro programa donde se encuentran todas las herramientas de diseño necesarias para la tarea.

Cabe mencionar que dispone de una app para IOS y Android con la que se pueden ver en tiempo real todos nuestros proyectos sin necesidad de código lo que facilita la tarea enormemente tanto para el desarrollador como para el cliente.

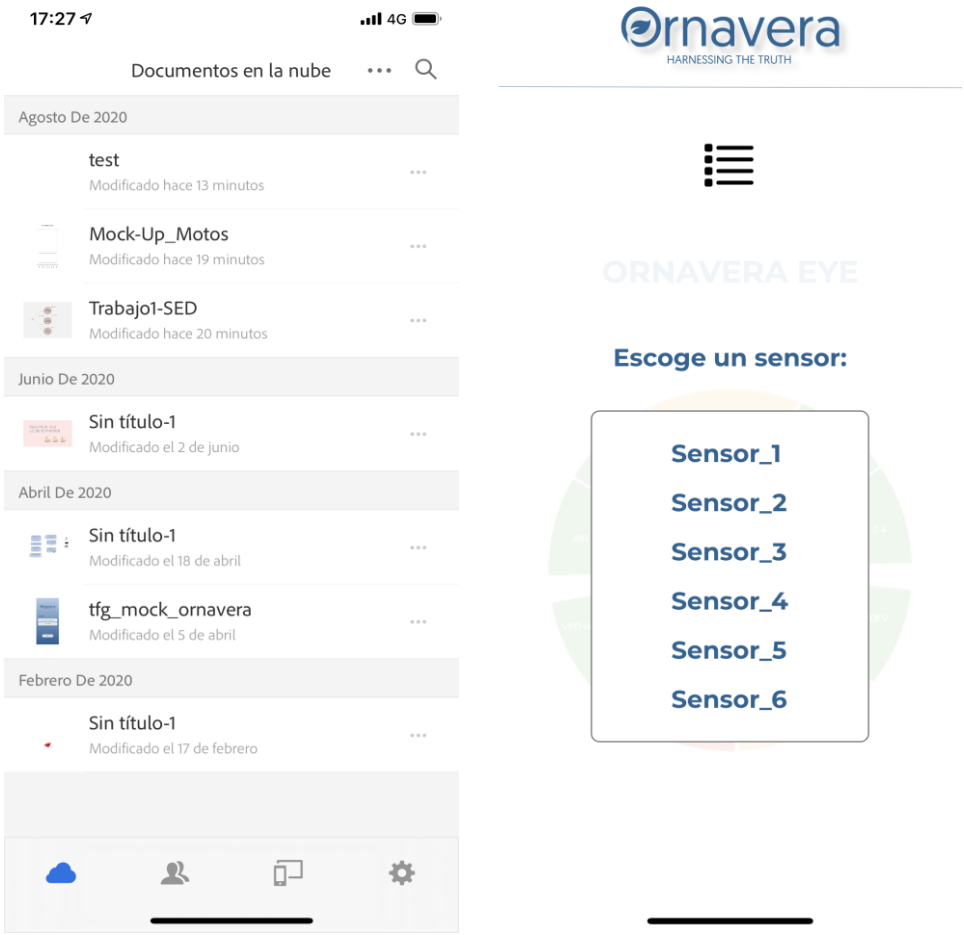


Figura 3.1.d App Adobe XD

3.1.1 Diseño Específico

Las figuradas mostradas a continuación, presentan una vista global del diseño completo de nuestra aplicación. Las estrategias de diseño siguen firmemente lo expuesto en el apartado 2.2.3 de este trabajo. Tras varias versiones de éste se optó por la marcada como "vfinal" en la figura 3.1.1.a que fue la que más satisfacía las necesidades tanto personales como de diseño de la app.

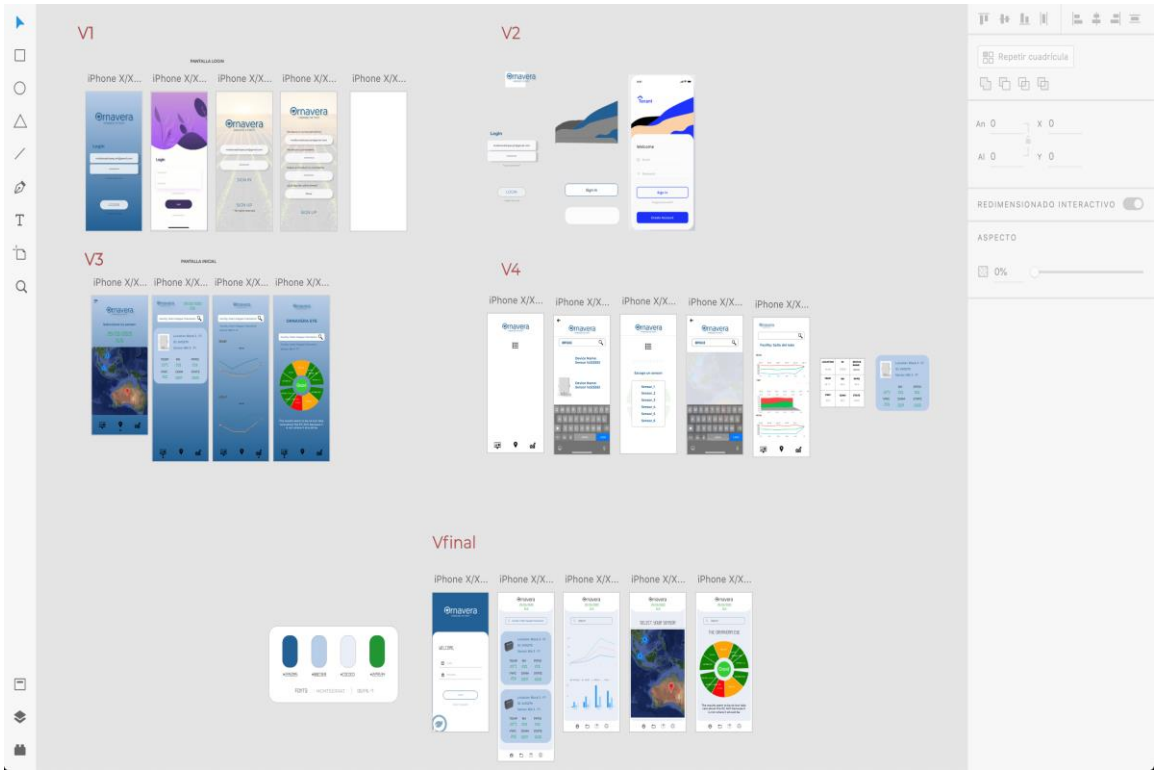


Figura 3.1.1.a Pantalla diseño App (I)

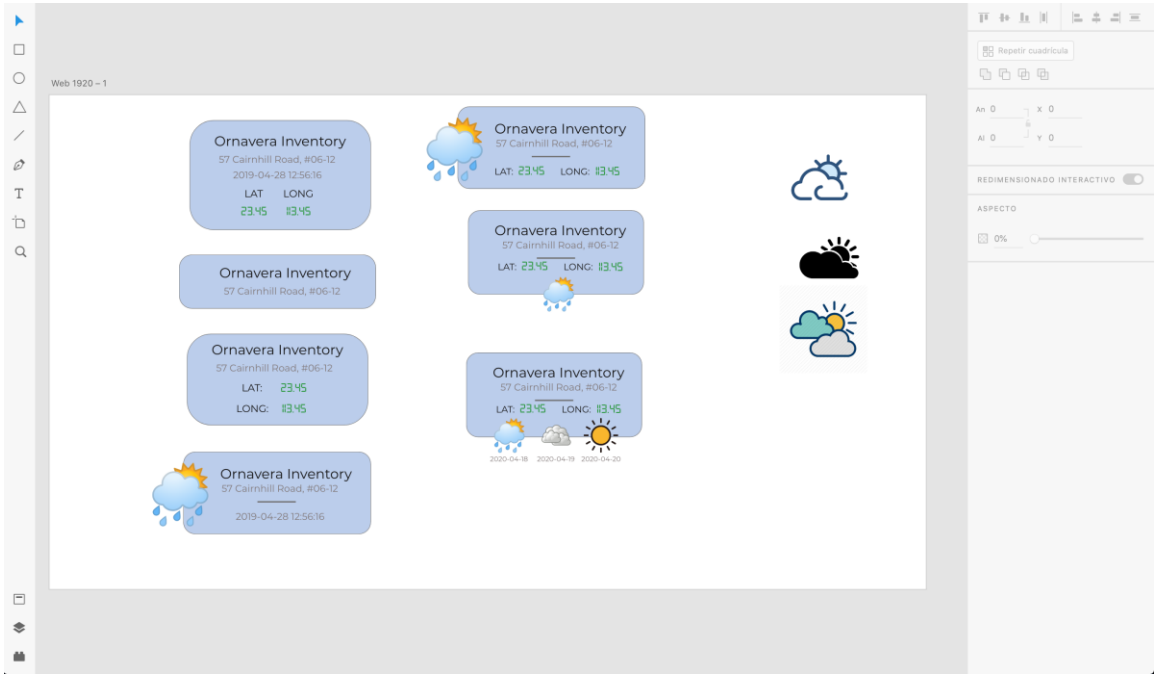


Figura 3.1.1.b Pantalla diseño App (II)

3.2 Lenguajes de Programación

3.2.1 Front-End

Vamos a entrar en la temática del entorno de desarrollo del front-end. Para este trabajo se ha elegido un entorno híbrido. En concreto se va a utilizar Flutter. Flutter es un SDK móvil *open source* que se utiliza para crear aplicaciones tanto de Android como de IOS de apariencia nativa a partir del mismo código. Se lanzó en Diciembre de 2018 pero ha existido en su fase beta desde 2015 y es una tecnología desarrollada por Google.

Actualmente Flutter [3] se encuentra en el top 10 de repositorios de software basados en estrellas de GitHub y además ya hay cientos de aplicaciones publicadas en ambos mercados con descargas de hasta 50 millones de usuarios (como el caso de Xianyu creada por el equipo de Alibaba).

Entrar en la explicación de la totalidad del código de Flutter relacionado con la aplicación sería una tarea ardua y poco aclaratoria, por lo que voy a contextualizar como funciona éste de manera global y pondré algún ejemplo referenciado en el apéndice.

La idea central de Flutter es el uso de *Widgets.* Mediante la combinación de diferentes widgets se puede llegar a crear la interfaz de usuario completa (botones y menús, fuentes, imágenes, colores y fondos con relleno o animados entre muchos otros). Flutter no utiliza widgets *OEM*, es decir, utiliza desarrolladores nativos como Material Design o Cupertino, sin embargo también es posible crear nuestros propios widgets.

Para evitar problemas de rendimiento debidos al uso de un lenguaje compilado, Flutter ha desarrollado su propio lenguaje conocido como Dart. Éste es compilado antes de tiempo en el código nativo. Gracias a ésto, Flutter es el único framework no nativo que ofrece vistas reactivas en tiempo real sin necesidad de un puente con el lenguaje *JavaScript*.

Una de las grandes ventajas de Flutter es su lenguaje, Dart, debido a que Flutter actualiza el árbol de vistas para cada nuevo macro, esto significa que se crean objetos que no duran más de un fotograma, por lo que Dart gracias a su "recolección y actualización de basura" ha demostrado ser muy eficiente. Además del compilador que tiene, con el que se pueden actualizar solo y exclusivamente las partes de código (los widgets) que se necesiten.

Como último detalle acerca de este lenguaje cabe decir que posee un repositorio enorme de paquetes precompilados y listos para su uso en nuestros proyectos, con una amplia guía de instalación que crece día a día.

3.2.2 Beneficios de Flutter

1. Ahorra tiempo y dinero.

Claramente debido a que se trata de un framework multiplataforma con el que puedes crear aplicaciones para las dos principales plataformas móviles con la misma base de código lo que ahorra mucho tiempo y recursos.

2. Actuación sobresaliente.

Primero por su precompilación en el código nativo y segundo por la posibilidad de uso de widgets propios, lo que implica menos comunicación entre aplicación y plataforma y tiempos de inicio rápidos y eficientes.

3. Recarga de código en tiempo real. Hot restart/Hot reload.

Es una de las características que más está llamando la atención de los

desarrolladores, la capacidad de refrescar en tiempo real nuestro código permite un feedback instantáneo al usuario y por lo tanto un mayor ahorro de tiempo a la hora de diseñar nuestras interfaces.

4. Compatibilidad:

Gracias al open source y a la capacidad de desarrollo de widgets propios, existe una amplia comunidad de desarrolladores que han creado sus *packages* para sus proyectos y si atraen nuestra atención podemos incluirlos en el nuestro.

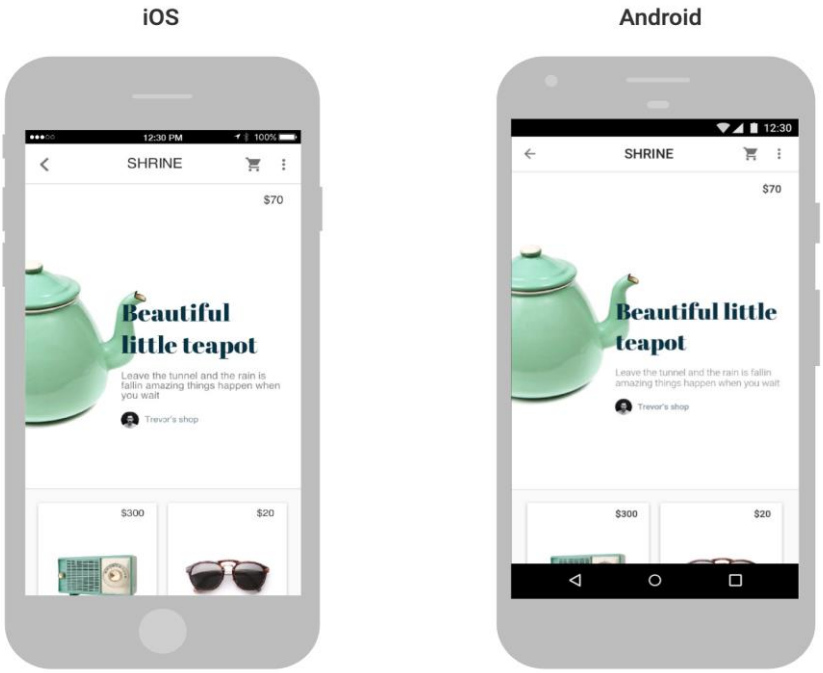


Figura 3.2.2.a Ejemplo Flutter iOS/Android (mismo código fuente). Fuente: flutter.dev/docs

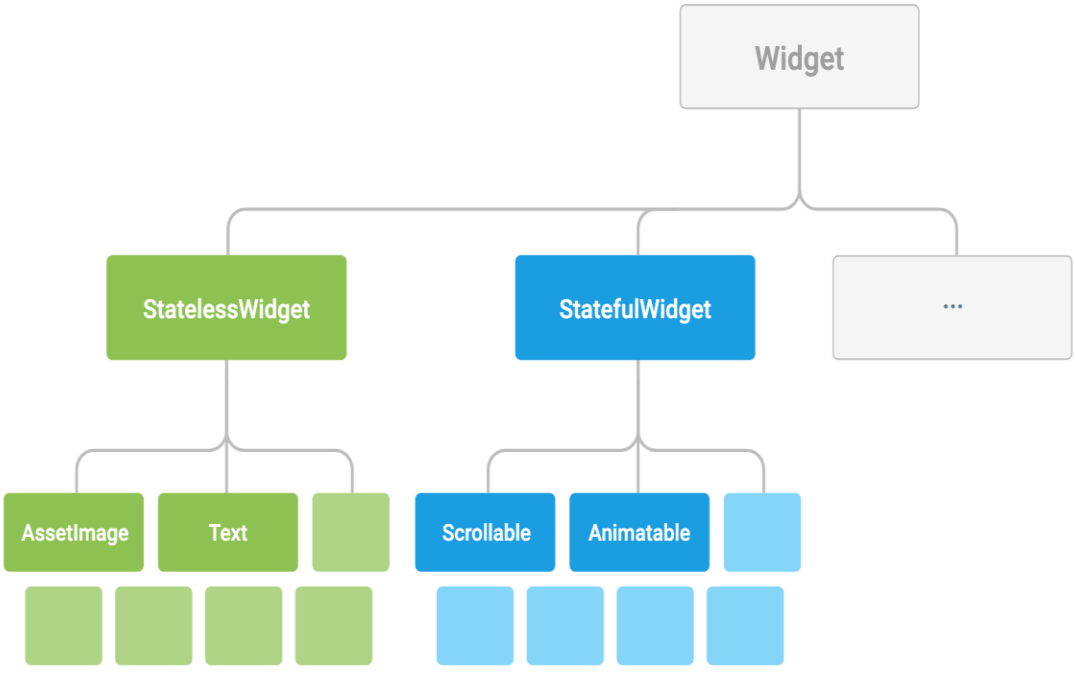


Figura 3.2.2.b Estructura en árbol de Widgets en Flutter. Fuente: flutter.dev/docs

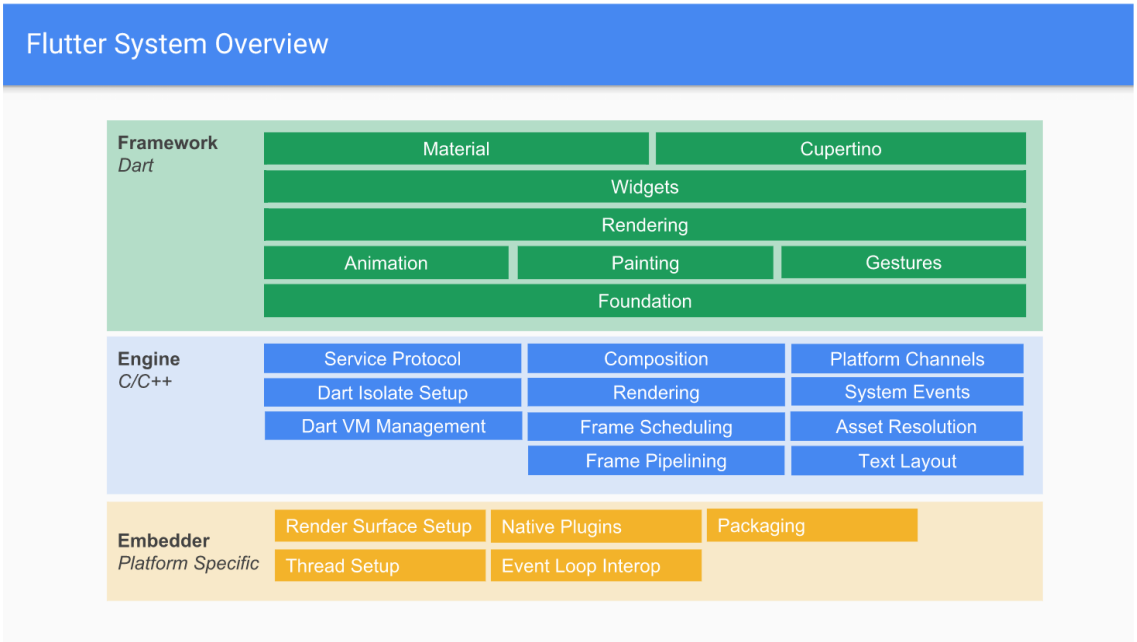


Figura 3.2.2.c Capa de widgets nativos. Fuente: flutter.dev/docs

3.2.3 Back-End

Entramos en el entorno de desarrollo de back-end. En este caso hemos escogido un microframework llamado Flask [4], basado en el lenguaje de programación Python.

Flask es la versión light-weight de Django, otro framework escrito también en Python. En nuestro caso y como el desarrollo del back-end es exclusivamente para la API con la que se comunicará nuestra aplicación, convenia mucho más Flask porque ahorra más recursos que Django, aunque sea un poco más limitado.

Flask nos ayuda a organizar el código, lógica, diseño y base de datos [5] en archivos separados. Además, nos permite enfocarnos en lo que los usuarios están solicitando y en qué tipo de respuesta devolver, lo que suponía la mejor elección para nuestra aplicación.

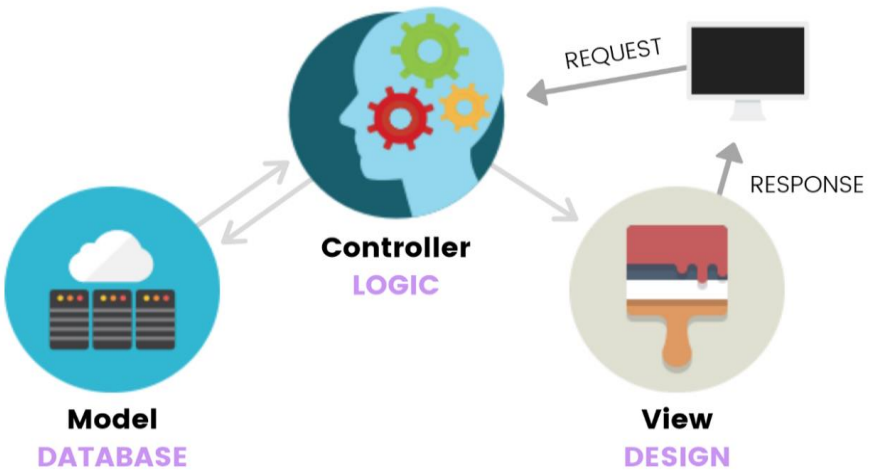


Figura 3.2.3.a Lógica de Flask (1). Fuente: medium.com

Para la utilización de este microframework lo primero que tendremos que hacer es crear un entorno virtual o *venv* en nuestro servidor. Este entorno es una herramienta que ayuda a mantener separadas las dependencias requeridas por los diferentes proyectos de python que contenga nuestro servidor y no estar continuamente actualizando de forma global la versión de este lenguaje y, por tanto, modificando y actualizando el código de cada proyecto.

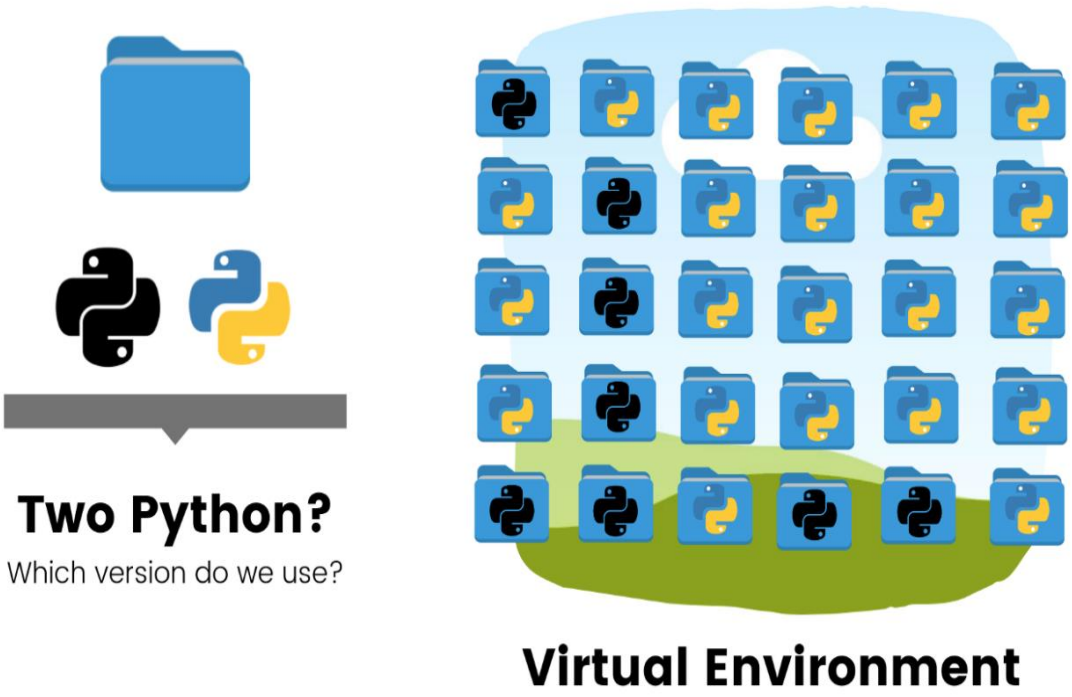


Figura 3.2.3.b Entornos virtuales (2). Fuente: medium.com

Una vez creado nuestro entorno virtual e instalado Flask pasaremos al código. Básicamente la herramienta va a leer de un archivo principal, normalmente *nombre_servicio.py*, desde donde:

1. **Importación de módulos.**

   Importaremos el módulo Flask y crearemos un servidor desde éste.

2. **Definición de endpoints.**

   Describiremos todas las rutas (“endpoints”) a las que el usuario puede acceder y mediante las cuales puede obtener información de la base de datos. Estas están recogidas en la figura 3.2.3.c.

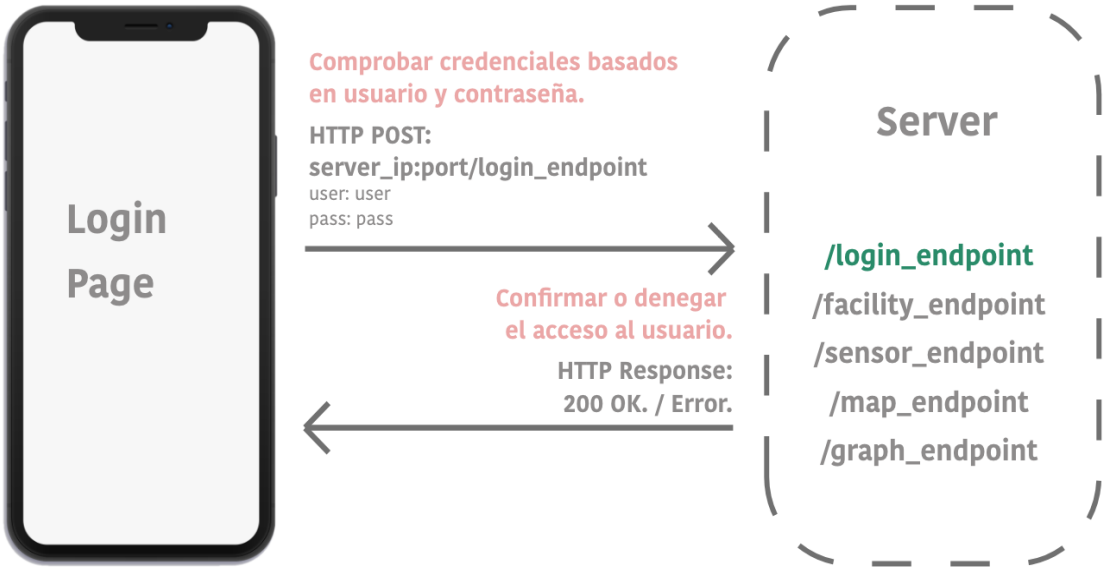


Figura 3.2.3.c Ejemplo de interacción con back-end en la pantalla de Login [6]

Finalmente, necesitamos que el módulo de python sobre el que se ejecuta Flask [7] esté siempre activo y evitar compilarlo de forma manual. Automatizaremos el servicio mediante systemctl desde nuestro servidor [8]. Básicamente un servicio es un programa que se ejecuta, o está esperando a ser ejecutado, en segundo plano.

Para ello, una vez implementado el servicio, desde la terminal de nuestro servidor lo único que tendremos que hacer es:

Para iniciar el servicio y dejarlo ejecutando en segundo plano:

```
systemctl start nombre_del_servicio
```

Para parar el servicio:

```
systemctl stop nombre_del_servicio
```

3.3 Bases de Datos

El entorno de la base de datos es MySQL que es un sistema de gestión de bases de datos relacional desarrollado por Oracle y es considerado como la base de datos de código abierto más popular del mundo. En el caso de nuestra app la base de datos ya venía proporcionada por lo que no se ha hecho un previo estudio de las posibilidades que había como en los apartados anteriores.

Como introducción de este tipo de tecnología cabe hacer una distinción entre bases de datos relacionales y no relacionales.

1. Bases de datos relacionales SQL.

El principio de este tipo de bases de datos se basa en la organización de la información en pequeñas porciones relacionadas entre sí mediante la relación entre identificadores.

En este ámbito se conocen las siglas ACID que vienen de: atomicidad, consistencia, aislamiento y durabilidad. Que son precisamente características que las bases de datos relacionales aportan a los sistemas que las usan y les permiten ser menos vulnerables frente a fallos y más robustas.

2. Bases de datos no relacionales No-SQL.

A diferencia de las relacionales y como su propio nombre indica, las bases de datos no relaciones no tienen un identificador que las relacione entre ellas. La información en este caso se organiza en documentos y es bastante útil cuando, a priori, no se tiene un esquema claro de lo que se va a desarrollar.

La principal tecnología usada con este tipo de bases de datos es MongoDB.

Tras introducir los dos grandes tipos de bases de datos y una vez que sabemos que en nuestra aplicación vamos a utilizar las bases de datos relacionales vamos a pasar a hacer un desglose de cada una de las tablas que utilizaremos y las respectivas columnas que las componen.

Vamos a utilizar dos “databases” dentro de nuestro servidor. La primera “user” será la encargada de proporcionarnos todo lo relacionado con el usuario, las facilities que posee y los sensores que se encuentran en cada una de esas facilities y por otro lado tenemos la tabla “meas” sobre la que obtendremos las medidas realizadas por los sensores.

Tablas	Contenido	Número de columnas
`facility`	Información relativa a las facilities	12
`location`	Localización de facilities a partir de su identificador	23
`rnode`	Información relativa a los sensores	15
`rnodelocation`	Localización de los sensores a partir de su identificador	9
`user`	Información acerca del usuario	17
`user_facility`	Relación entre usuario y facilities asociadas	2

Tabla 3.3.a. Base de datos user

Tablas	Contenido	Número de columnas
stat	Histórico de máximos, mínimos y medios de cada una de las medidas realizadas por los sensores	94
measv4	Medidas realizadas por los sensores	43
event	Acciones a realizar asociadas a facilities	15

Tabla 3.3.b. Base de datos meas

Campo	Definición
fid	Identificador de facility
customerid	Identificador de cliente
facilityname	Nombre de la facility
datecreated	Fecha de creación
datemodified	Fecha de modificación
address	Dirección
lat	Coordenada de latitud
lng	Coordenada de longitud
datatableid	Identificador de tabla asociada
tzid	Identificador de time zone asociado
isdeleted	Estado
soilid	Identificador de tipo de suelo

Tabla 3.3.c. Base de datos user. Contenido de la tabla facility

Campo	Definición
userid	Identificador de usuario
email	Email de usuario
profileid	Identificador de tipo de perfil
password	Contraseña de usuario
token	Token de sesión de usuario
customerid	Identificador de cliente
defaultscreenid	Identificador de pantalla por defecto asociada al usuario
weightunitid	Identificador de unidad de pesaje
lengthunitid	Identificador de unidad de estatura
tempunitid	Identificador de unidad de temperatura
isdeleted	Estado de uso
isactive	Estado actual

Tabla 3.3.d. Base de datos user. Contenido de la tabla user

Nombre	Abreviatura	Definición
Temperatura	**t**	Temperatura recogida por el sensor
Humedad relativa	**rh**	Relación entre presión de vapor de agua y vapor en equilibrio a temperatura dada
Déficit de Vapor	**vpd**	Cantidad de vapor para saturar la atmósfera a temperatura dada
Grados-día	**dd**	Temperatura acumulada
Luminosidad	**lux**	Unidad de luminancia
Temperatura de suelo	**st**	Temperatura del suelo
Electro-Conductividad	**ec**	Capacidad de conducción de corriente eléctrica del suelo
Contenido volumétrico en agua	**vwc**	Volumen de agua entre volumen total del suelo
Diámetro del tronco	**diam**	Diámetro del árbol
Radiación fotosintéticamente activa	**PAR**	Cantidad de radiación integrada dentro del rango de longitudes de onda que son capaces de producir actividad fotosintética en plantas y otros organismos
Integral de luz diaria	**DLI**	Describe el número de fotones fotosintéticamente activos (PAR) que se entregan a un área específica en un período de 24 horas

Tabla 3.3.e. Medidas tomadas por los sensores usadas en las tablas measv4 y stat

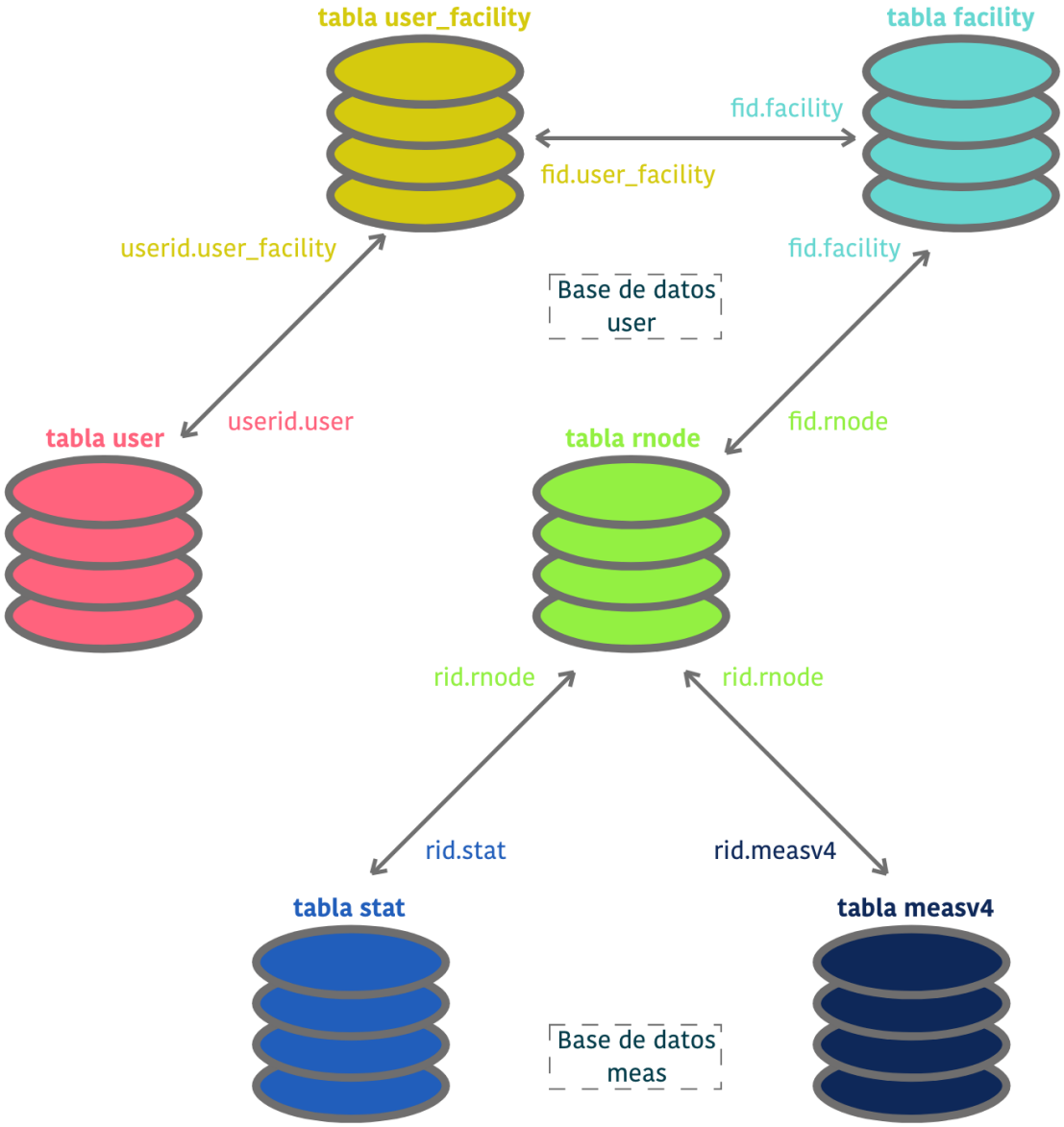


Figura 3.3.a Relaciones entre bases de datos

4 DISEÑO EXPERIMENTAL Y RESULTADOS

"Escoge un trabajo que te guste y nunca tendrás que trabajar ni un solo día de tu vida"

- Confucio -

Una vez realizada la elección de framework y tras conocer todo lo que implica el desarrollo y diseño de una aplicación, estamos preparados para iniciar nuestro proyecto, no sin antes, y como último matiz, hacer una elección del tipo de arquitectura que seguirá éste. En este apartado nos referiremos al proceso de selección de esta arquitectura y en concreto, una vez elegida ésta, como la hemos tratado en nuestra app.

4.1 Arquitectura de la Aplicación

Como parte fundamental de la creación de una aplicación es necesario definir una arquitectura previa a su desarrollo que nos permita, entre otros factores hacer nuestro código modular y aumentar la eficiencia de éste. Conocemos muchos patrones de desarrollo como MVC, MVMC, BLoC… pero en este caso se ha empleado un modelo conocido como ScopedModel.

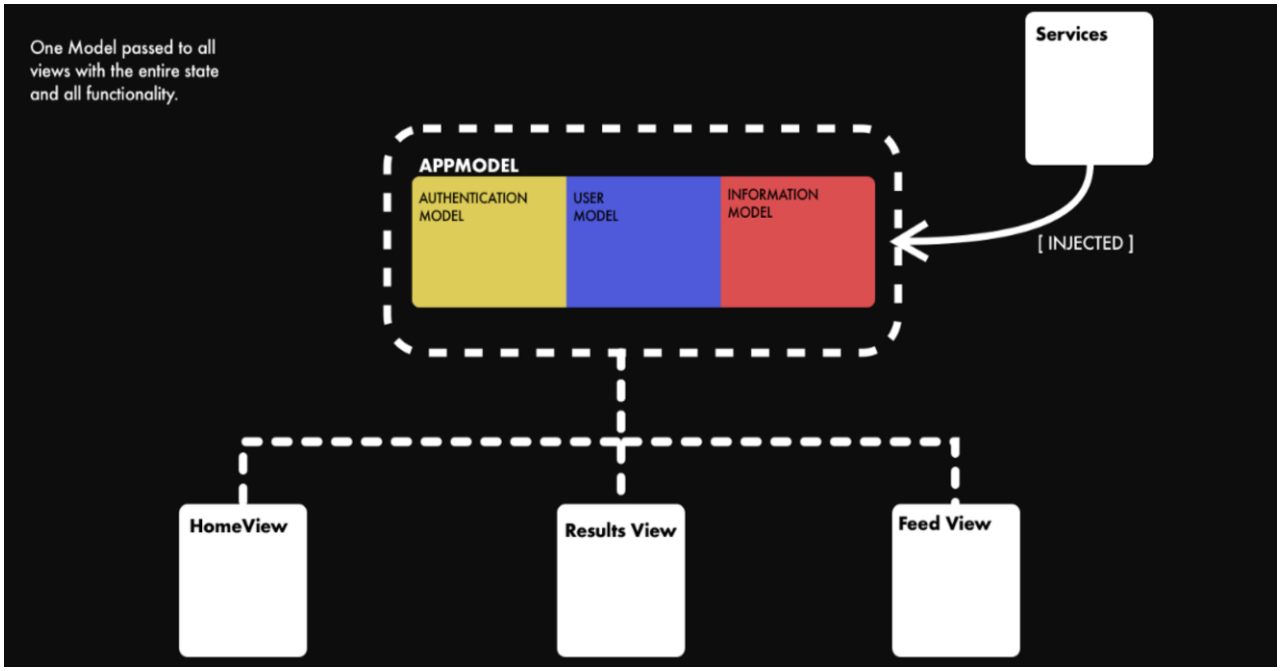


Figura 4.1.a Non Scoped Model Flutter. Fuente: medium.com/arquitectura-flutter

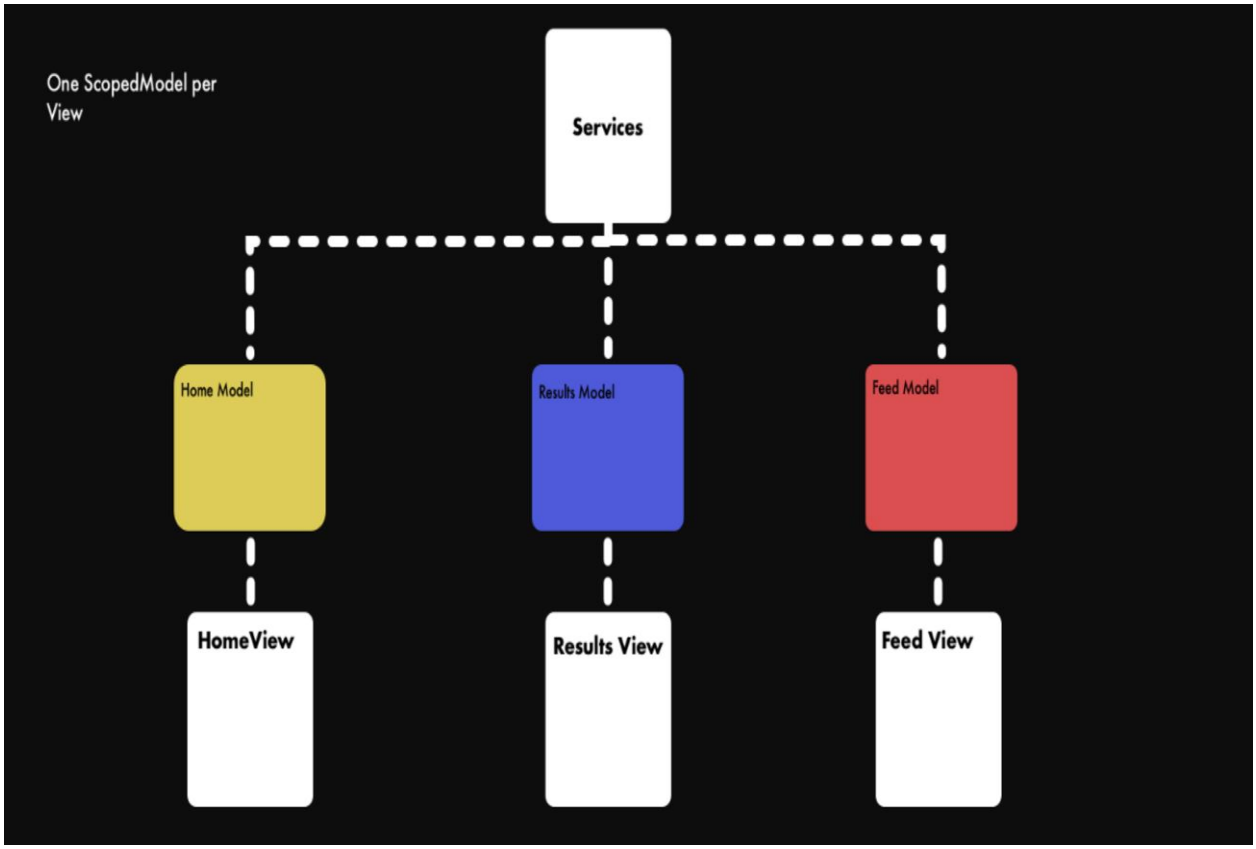


Figura 4.1.b Scoped Model Flutter. Fuente: medium.com/arquitectura-flutter

Como podemos observar el Scopped Model, al contrario que el Non Scopped Model y como su traducción vendría a indicar (*"modelo centrado"*), dedica a cada interfaz de la aplicación, es decir, a cada una de las pantallas que la componen, un modelo de datos asociado a cada una de ellas, que se nutren de las llamadas a la API del servidor.

Esta sería la idea principal de estructura de la aplicación que se ha seguido para su desarrollo, ahora pasaremos a ejemplificar la totalidad de ésta.

4.1.1 Arquitectura Principal

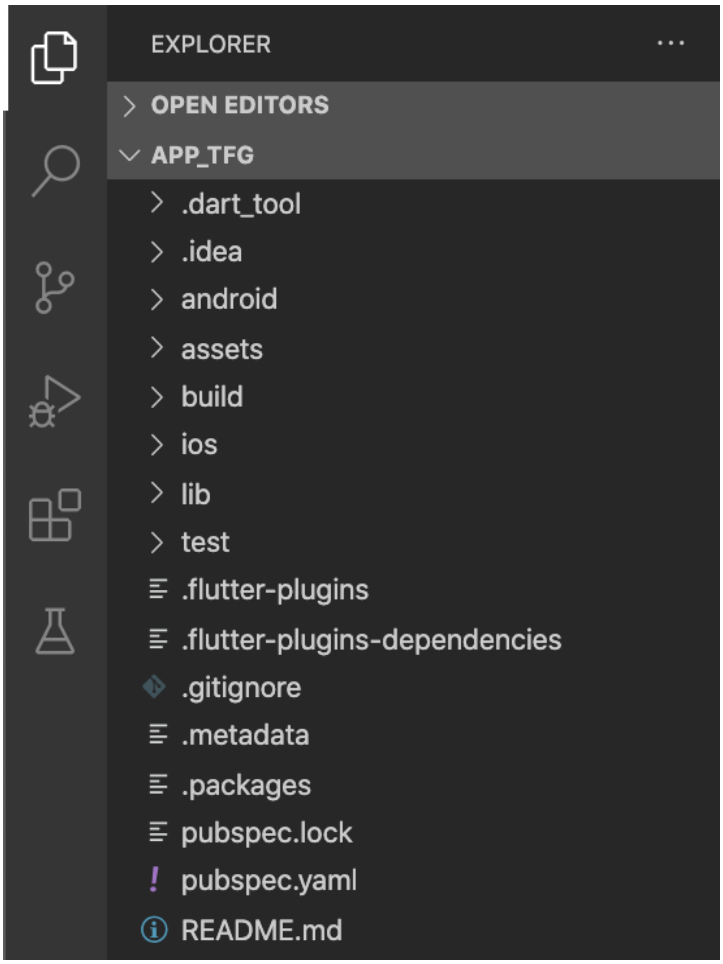


Figura 4.1.1.a Arquitectura principal de la app

Antes de introducir la arquitectura ScoppedModel, debemos explicar la arquitectura del framework Flutter. Como se puede ver la figura 4.1.1.b, en Flutter se dispone de dos menús llamados Android e IOS, que son los menús en los que se almacena la información relacionada con cada uno de estos frameworks nativos, y a los que hay que acudir cuando necesitamos dar permisos de algún tipo en las respectivas plataformas para uso de algún paquete en especial (como por ejemplo la clave de la API de Google Maps).

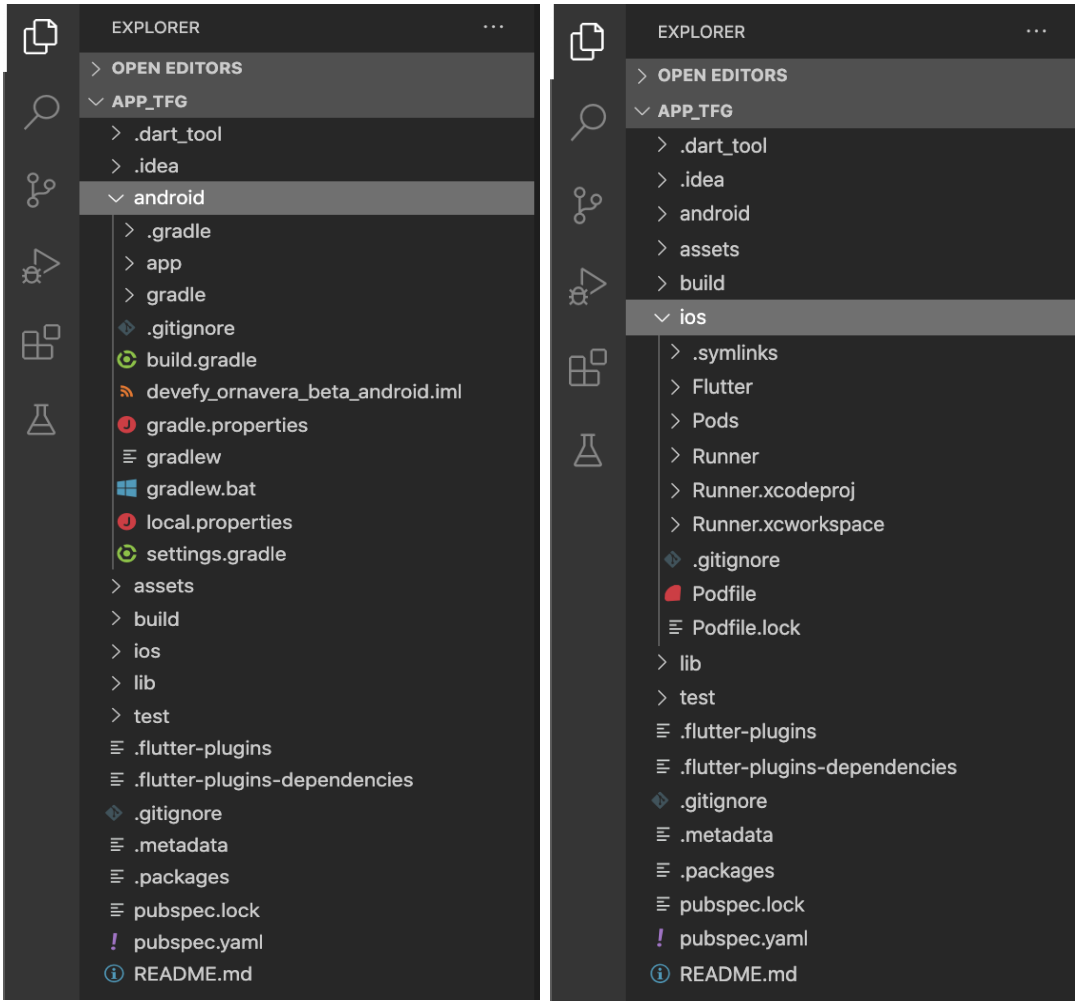


Figura 4.1.1.b Menú de frameworks nativos

Para todo lo relacionado con el “material” de la aplicación, encontramos la carpeta assets, en ella emplazaremos todo lo relativo a fuentes, imágenes, iconos, animaciones de los que dispondrá la aplicación y de ella se nutrirá tanto Android como IOS, es decir, si queremos definir el icono de la aplicación, éste debe situarse en la carpeta assets para que cada plataforma sea capaz de leerlo a su manera, pero desde la misma fuente.

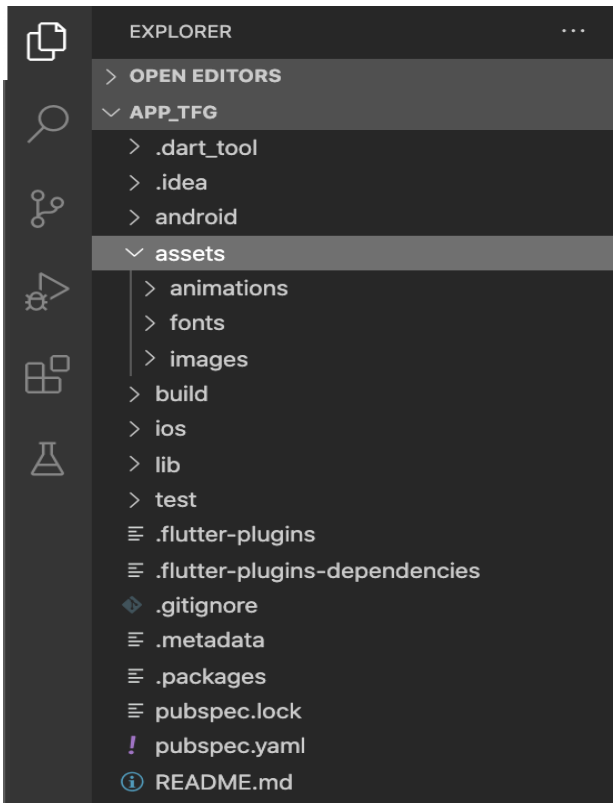


Figura 4.1.1.c Carpeta assets

4.1.2 Scopped Model en Flutter

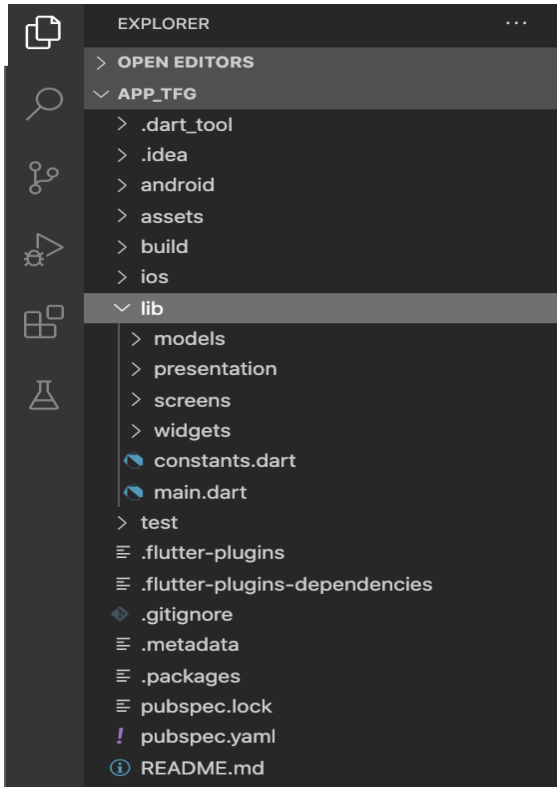


Figura 4.1.2.a Carpeta lib

Ahora sí pasamos a explicar el modelo Scopped Model dentro de nuestra aplicación. Todo lo relacionado con éste estará dentro de la carpeta lib. Como podemos observar en la figura 4.1.2.b, dentro de lib tenemos *models, screen y widgets*,

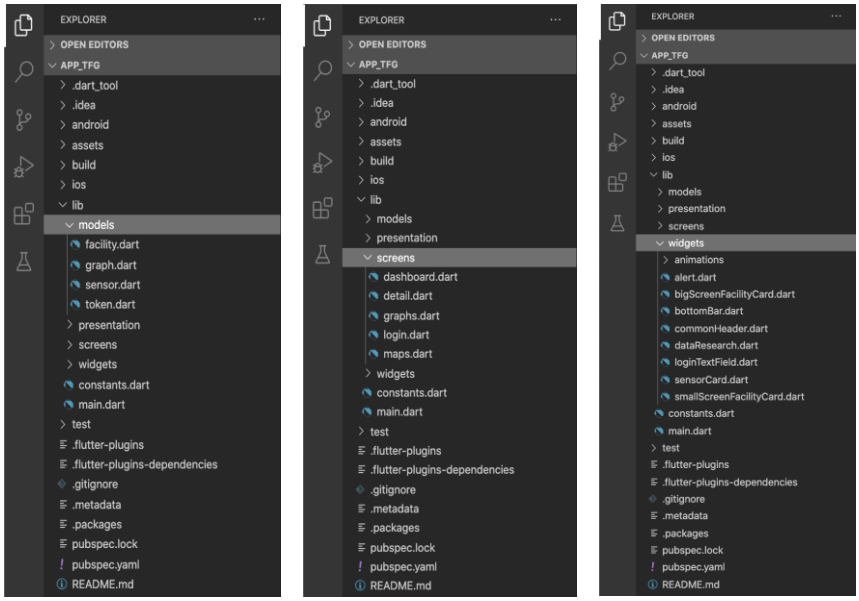


Figura 4.1.2.b Carpetas models, screens y widgets, respectivamente

Cumpliendo con el requerimiento de este modelo, para cada interfaz de la aplicación, tenemos un modelo de datos que se nutre de una petición al servidor, esto es, para la pantalla de facilities (*screens/dashboard.dart*) tenemos un modelo (*models/facility.dart*) asociado para la obtención de la información y la maquetación de los datos.

Finalmente, atendiendo a la estructura de un proyecto en Flutter, se debe hacer mención del archivo pubspec.yaml, Flutter utiliza este archivo, ubicado en la raíz del proyecto, para identificar los recursos requeridos por la aplicación, es decir, cualquier paquete externo que se importe se debe incluir aquí para su correcto uso.

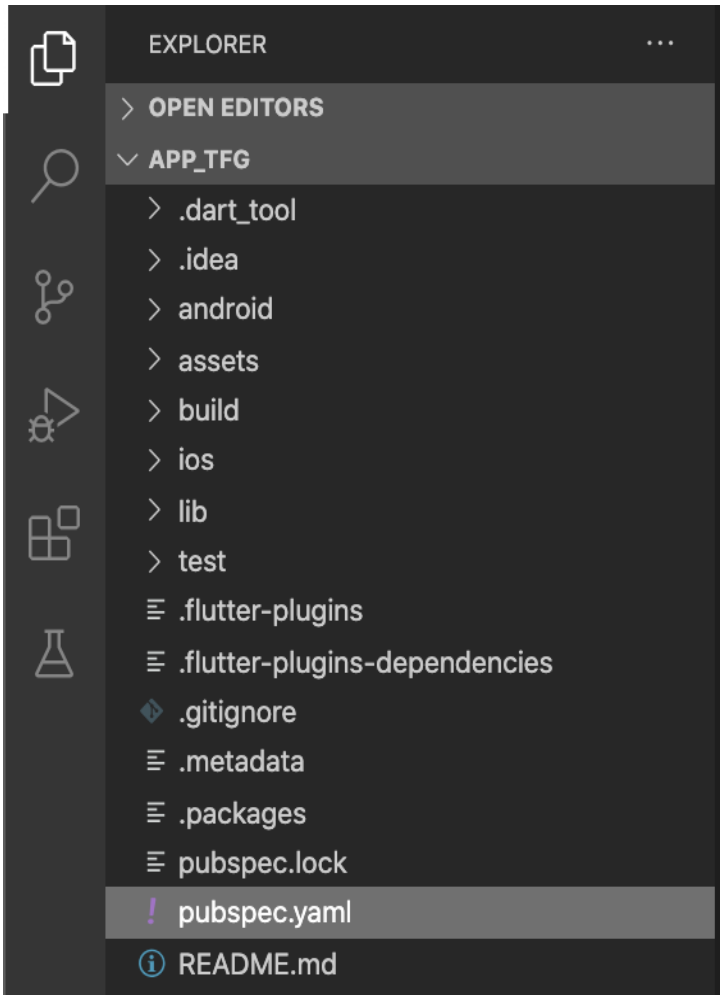


Figura 4.1.2.c Ubicación de pubspec.yaml

pubspec.yaml

```yaml
description: A new Flutter project.
version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^0.1.2
  flutter_screenutil: ^1.1.0
  http: ^0.12.0
  intl: ^0.16.1
  flappy_search_bar: ^1.7.2
  path_provider: ^1.6.5
  flare_flutter: ^1.5.0
  charts_flutter: ^0.9.0
  fcharts: ^0.0.11
  auto_size_text: ^2.1.0
  google_maps_flutter: ^0.5.24+1
  shared_preferences: ^0.4.3
  flutter_phoenix: ^0.1.0


dev_dependencies:
  flutter_test:
    sdk: flutter


flutter:

  uses-material-design: true

  #To add assets to your application, add an assets section, like this:
  assets:
   - assets/images/
   - assets/images/weather_icons/
   - assets/animations/

  fonts:
    - family: Montserrat
      fonts:
        - asset: assets/fonts/Montserrat-Light.ttf
        - asset: assets/fonts/Montserrat-Bold.ttf
    - family: Digital7
      fonts:
        - asset: assets/fonts/digital-7.ttf
    - family: CustomIcons
      fonts:
        - asset: assets/fonts/CustomIcons.ttf
```

Figura 4.1.2.d Contenido de pubspec.yaml

4.2 Arquitectura: Resultados y Capturas de Pantalla

Para una mejor comprensión de lo que sería el resultado final de la aplicación vamos a realizar una descripción detallada de cada una de las pantallas que la componen.

4.2.1 Pantallas: Login

Esta pantalla, en lo referente al diseño, como todas las desarrolladas para esta aplicación tiene un diseño propio realizado con Adobe XD, se ha obtado por elegir dos de los logos corporativos para dar un tono de seriedad a la pantalla con la que se abre la aplicación.

El funcionamiento de esta pantalla es el siguiente:

1. **Recogida de datos.**

   El usuario introduce sus datos (usuario y contraseña) [9], éstos se recogen de teclado mediante widgets asociados a los campos de email y password que actúan de controladores y detectan cualquier cambio en ellos.

2. **Envío de datos al servidor.**

   Los datos son mandados mediante una petición POST hacia el servidor.

3. **Generación de token de inicio de sesión.**

   En el servidor, se comprueba la veracidad de éstos y, si efectivamente los datos son correctos, se genera un token de inicio de sesión.

4. **Shared preferences.**

   El token generado se devuelve a la aplicación y se almacena en las Shared Preferences, que permiten a las aplicaciones guardar configuraciones, propiedades o datos para su posterior reutilización, ya que gracias a este token, se podrá acceder a las demás pantallas tras checkear su validez.

5. **Comenzamos.**

   Una vez realizados todos los pasos anteriores, accedemos a la aplicación con nuestro token asociado.

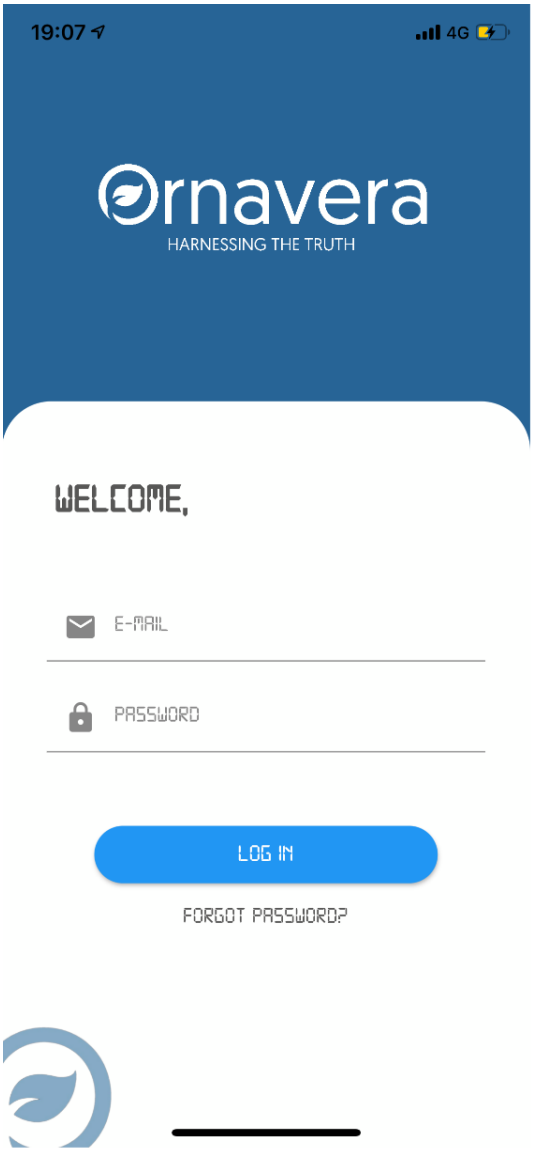


Figura 4.2.1 Pantalla de login

4.2.2 Pantallas: Facilities (Dashboard)

En lo referente al diseño en esta pantalla se ha optado por lo que se conoce como *tarjetas* que en un simple vistazo muestran información relacionada con cada una de las facilities, en concreto muestran:

i. El tiempo que hace actualmente basado en la API de accuweather.

ii. El nombre de la facility.

iii. La dirección de la facility.

iv. Las coordenadas de latitud y longitud de la facility.

El funcionamiento de esta pantalla es el siguiente:

1. **Obtención del token.**

   Se accede a las Shared Preferences para obtener el token generado en la pantalla del login.

2. **Consulta al servidor.**

   Una vez tenemos el token, pasamos a hacer una petición al "endpoint" del servidor asociado con las facilities. Éste nos va a devolver en formato json todos los campos mencionados en el punto anterior.

   Vamos a guardar la información de las facilities (nombre e id) en las Shared Preferences como hicimos con el token.

3. **Maquetación de datos dentro de la app.**

   Tras obtener los datos, lo que vamos a hacer es crear una lista (*listview*) en función del número de facilities que tenga ese usuario, es decir, una lista dinámica y para cada "tarjeta" de esta lista añadiremos con el formato gráfico que podemos ver en la imagen los datos.

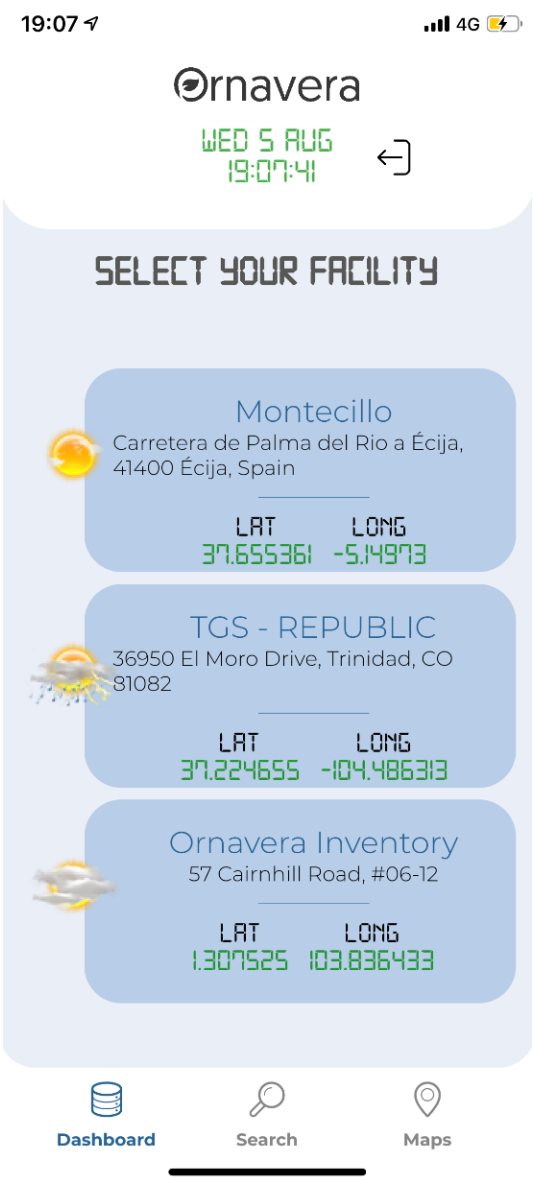


Figura 4.2.2 Pantalla de facilities

4.2.3 Pantallas: Sensores

En lo referente al diseño en esta pantalla se ha vuelto a optar por *tarjetas* que muestran información relacionada con cada una de los sensores:

i. El nombre del sensor.

ii. La fecha de la última medida tomada.

iii. La temperatura asociada a la fecha de la última medida tomada.

iv. La humedad relativa asociada a la fecha de la última medida tomada.

v. El déficit de vapor asociado a la fecha de la última medida tomada.

vi. El contenido volumétrico en agua asociado a la fecha de la última medida tomada.

vii. La integral de luz diaria asociada a la fecha de la última medida tomada.

viii. El estado del sensor (activado/desactivado).

El funcionamiento de esta pantalla es el siguiente:

1. **Obtención de los datos.**

   La obtención de los datos expuestos anteriormente va a depender de la facility seleccionada en la pantalla de facilities. Cuando pulsemos en una de las tarjetas de facility, se hará una petición de los datos asociados con el número de facility pulsado (según la numeración de la base de datos, no de la vista de la app), en este caso vemos que el uso del token no es necesario porque ya lo hemos usado para hacer la petición de las facilities.

2. **Consulta al servidor.**

   Una vez tenemos el número de facilty, pasamos a hacer una petición al "endpoint" del servidor asociado con los sensores. Éste nos va a devolver en formato json todos los campos mencionados en el punto anterior.

3. **Maquetación de datos dentro de la app.**

   Volvemos a crear una lista (*listview*) en función del número de sensores que tenga esa facility, para cada "tarjeta" de esta lista añadiremos con el formato gráfico que podemos ver en la imagen X, los datos de cada sensor.

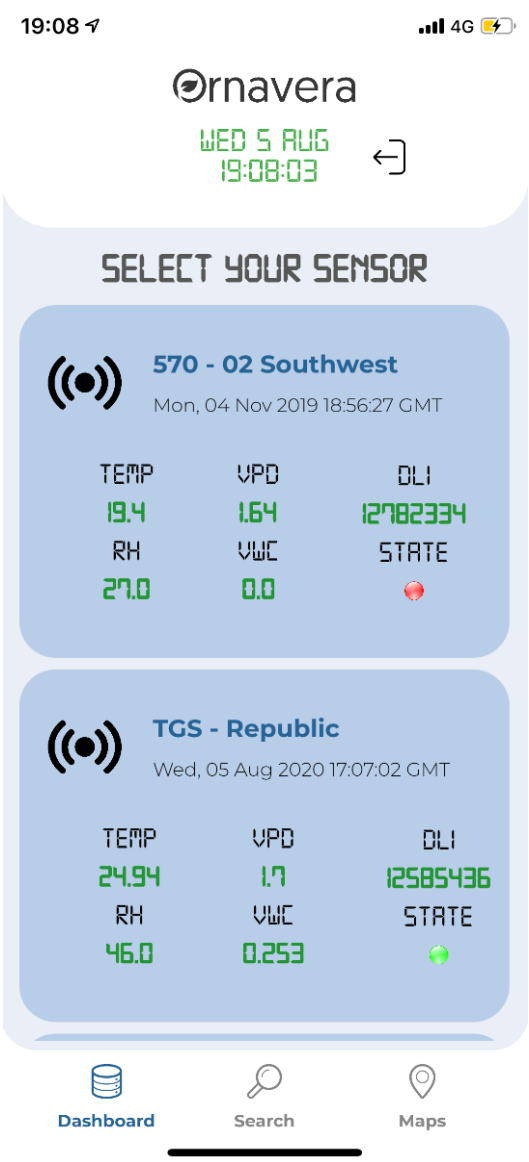


Figura 4.2.3 Pantalla de sensores

4.2.4 Pantallas: Gráficas

Como propósito de este trabajo de fin de grado se ha optado por implementar una gráfica de tipo barras [10] para una medida en concreto (temperatura), pero como bien se especificará en el apartado de líneas futuras, para una posible expansión de la aplicación se podrían desarrollar otro tipo de gráficas (lineales, donut…) y con una selección de medida.

El funcionamiento de esta pantalla es:

1. **Selección de medida en la pantalla de sensores.**

   En este caso, y como ya se ha especificado en la introducción de este apartado, seleccionaremos la medida de temperatura asociada a cada sensor.

2. **Obtención de los datos.**

   Mediante una petición hacia el endpoint del servidor asociado con las gráficas obtendremos en formato json los datos relacionados con temperatura máxima, mínima y media de cada uno de los sensores.

3. **Maquetación dentro de la app.**

   Tras obtener los datos, que ya vienen formateados gracias a la labor del servidor, apoyándonos en una librería de flutter conocida como charts_flutter, lo que tendríamos que hacer es asociar a cada gráfica de barras una lista de datos (temperatura máxima, media, mínima y fecha en la que fue tomada la medida), y asignarle un color intuitivo para hacerlo fácil hacia el usuario.

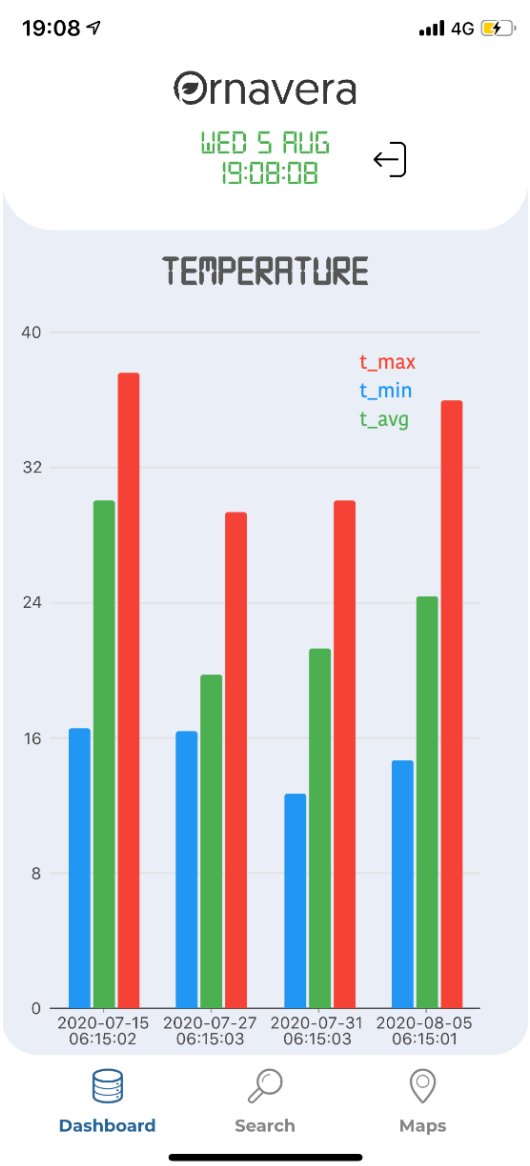


Figura 4.2.4 Pantalla de gráficas

4.2.5 Pantallas: Búsqueda

El funcionamiento de esta pantalla es elemental [12]:

1. **Búsqueda en base a cadena de texto.**

   La primera opción es introducir una cadena de texto en la que especifiquemos el nombre de alguna de nuestras facilities, la aplicación buscará coincidencias según la base de datos y mostrará los resultados.

2. **Nueva vista.**

   Una vez obtenidos los resultados, al seleccionar uno, accederemos a la pantalla de sensores relacionados con la facility, de la misma forma en la que se especifica en el apartado 4.2.2.

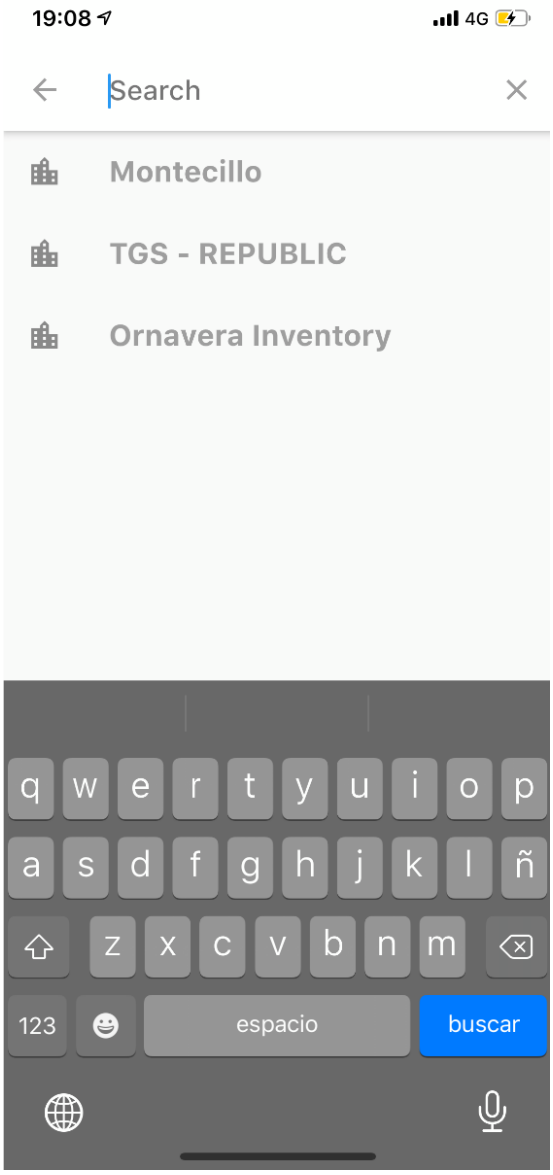


Figura 4.2.5 Pantalla de búsqueda

4.2.6 Pantallas: Mapas

Finalmente, tenemos la pantalla de mapas [13]. Una pantalla clave para la geolocalización de las facilities y en un futuro la de los sensores si se requiere.

El funcionamiento de esta pantalla es:

1. **Shared preferences.**

   Consultamos las Shared Preferences para obtener el id de las facilities relacionado con el usuario que esté usando la app.

2. **Petición al servidor.**

   Una vez obtenidos los id, realizamos una petición al endpoint de mapas que nos devolverá las coordenadas relacionadas con cada una de las facilities.

3. **Google Maps.**

   Gracias al servicio que ofrece google maps, y una vez creada una cuenta para tener acceso al uso de su API, importamos el módulo en flutter, *google_maps_flutter*, y lo que nos queda es pasar como parámetro al constructor del mapa las coordenadas de latitud y longitud recibidas por el servidor, para que éste y utilizando marcadores los geolocalice. En la figura 4.2.6 se puede ver un ejemplo de uso.

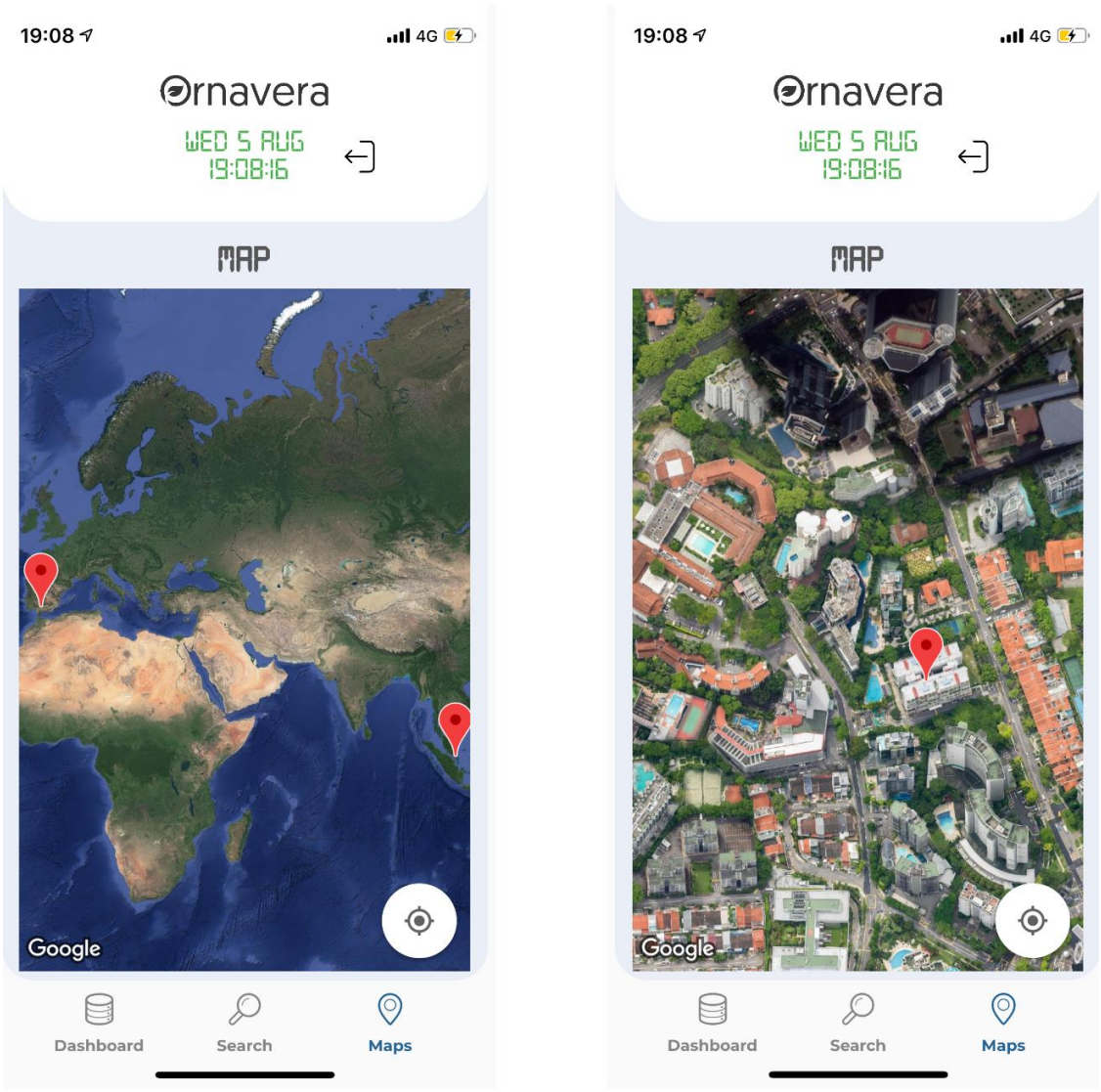


Figura 4.2.6 Pantalla de mapas

5 CONCLUSIONES Y LÍNEAS FUTURAS

"Las personas que están lo suficientemente locas para pensar que pueden cambiar el mundo, son las que lo cambian"

- Steve Jobs -

Con lo expuesto en el marco teórico, vemos que el desarrollo completo de una app va de la mano de muchísimos factores. Y ya no solo son técnicos como lenguajes de programación, ni de diseño para poder maquetar la aplicación desde un primer momento, sino factores de negocio, que hacen un estudio de mercado y ven que posibilidad tiene más opciones de triunfar. Es importante crear un ecosistema en el que todos estos factores vayan a la par, además de realizar previsiones de mejora para nuestra app.

5.1 Conclusiones

Hemos aprendido que la realización de una app se basa, a grandes rasgos, en cuatro bloques bien diferenciados. El primer bloque trata del diseño, será de vital importancia disponer desde primera hora de una aproximación del producto final sin tener que implementar ni una línea de código. Esto hace la tarea más fácil a todas las partes, ya que gracias a los tipos de software que existen hoy día, como Adobe XD en nuestro caso, podemos ir al cliente con una maqueta de la app en cuestión de pocas horas, idea que atrae bastante a la hora de vender nuestro producto.

En segundo lugar, ya sabemos que hay que elegir entre tecnologías nativas o híbridas y que el desarrollo de una app en cada una puede cambiar por completo. En nuestro caso, la alternativa clara era una tecnología híbrida para desarrollar una aplicación que funcionara en todo tipo de dispositvos con el mismo código a modo de TFG, y bajo mi punto de vista creo que el futuro irá encaminado por este sendero, ya que reduce horas de trabajo, coste y recursos, obteniendo un resultado que se adapta perfectamente a los requerimientos.

En tercer lugar, tenemos la elección de la tecnología que dará respaldo a nuestra aplicación además de a nuestros datos. En nuestro caso Flask con Python, que actuará de API para la conexión de cada una de las interfaces de nuestra app con el servidor.

Y, por último, pero no menos importante, está la elección de la base de datos. En el caso de nuestra app es una decisión que no se ha tomado porque ya venía dada, pero en general y en función de los propósitos que tengamos, hemos visto que hay dos tipos, una posee una estructura rígida y organizada basada en ACID, mientras que otra ofrece una estructura libre y flexible que permite modificarse con el tiempo. Ambas, hoy en día, son una buena opción en función de nuestros intereses.

5.2 Líneas Futuras

Nos centramos ahora en las líneas futuras de mejora de la aplicación, sobre lo que denominaremos la versión beta, entendiendo como versión beta el estado actual de la aplicación.

La característica principal del código realizado en Flutter es la **Modularidad**. Esta característica resultará clave para las mejoras a incorporar en la aplicación en cualquier momento ya que el código soporta la inclusión de nuevos desarrollos de una manera eficiente comparativamente con otros entornos.

Teniendo en cuenta este aspecto clave, expondré a continuación posibles mejoras que dotarán a la aplicación un mayor valor añadido, una mejor experiencia de usuario y, por tanto, un mayor potencial de comercialización.

5.2.1 Modificación de Idioma

Como primera mejora y básica la opción de cambiar el idioma de la app en función de la localización del usuario que la utilice. Esta mejora básica, pasará a ser una mejora necesaria si queremos que la aplicación tenga uso internacional.

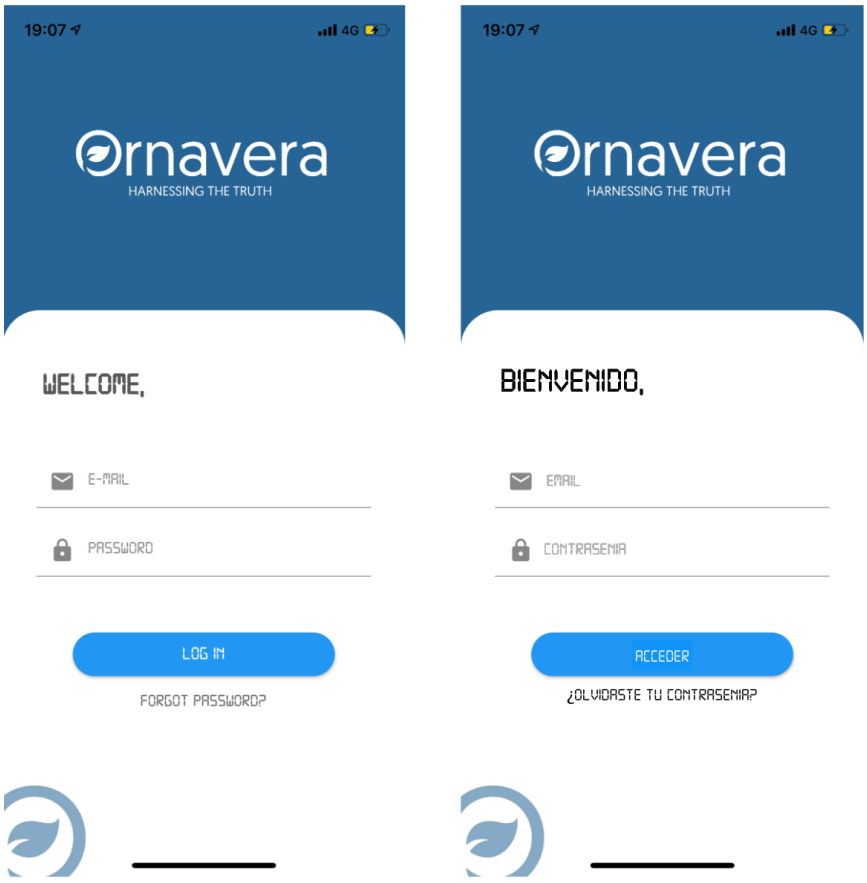


Figura 5.2.1 Nueva pantalla de modificación de idioma

5.2.2 Nueva Pantalla Eventos

Se propone una nueva pantalla asociada a cada facility que se denominará Pantalla de eventos y, tal como su nombre indica, a través de esta pantalla será posible asociar eventos relacionados con la facility, con un calendario para marcar la fecha y un histórico para ir registrando los eventos completados y poder diferencias entre éstos y los programados.

Figura 5.2.2 Nueva pantalla de eventos

5.2.3 Selector de Variables en la Pantalla Sensores

Como ya se ha indicado y con propósitos exclusivos de TFG, actualmente solo hay una variable que se grafica y es la temperatura, en un futuro sería muy interesante añadir un selector de variables como paso previo a la conexión con las pantallas gráficas que permitirán al usuario la posibilidad de acceder a la representación gráfica de cualquiera de las variables que gestiona la aplicación.

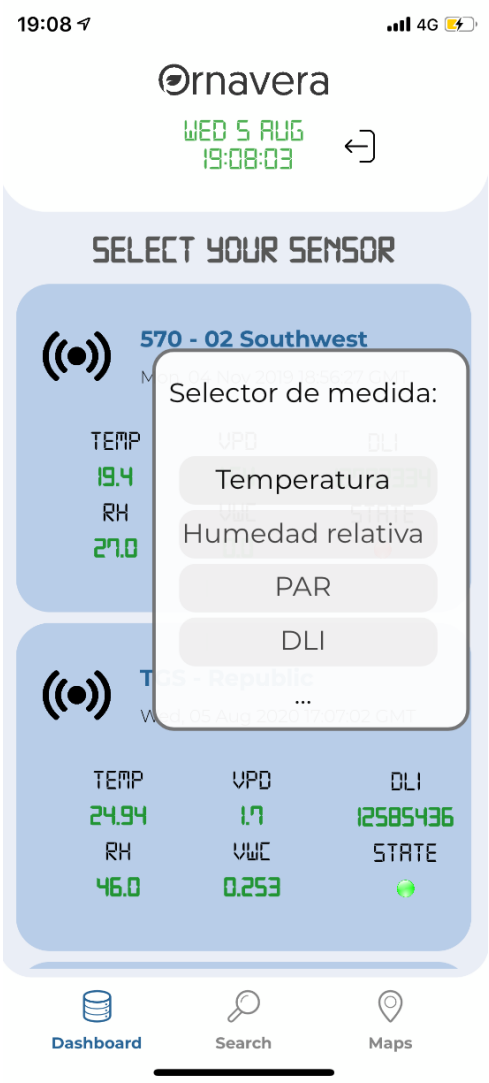


Figura 5.2.3 Nueva pantalla de selector de variables

5.2.4 Nuevos Tipos de Gráficas

Por último, y como podemos ver en la figura 5.2.4, la proposición sería el desarrollo de nuevos tipos de gráficas [13] que pudieran hacerle la tarea más fácil al usuario a la hora de consultar cualquier tipo de medida, entre ellas están:

1. **Gráficas temporales.**

   Éstas permitirían al usuario ver la evolución semanal (o de cualquier otro período de tiempo) de la variable que haya seleccionado.

2. **Gráficas circulares/sectoriales para previsiones.**

   Gráficas que, en base a ciertas medidas, es capaz de generarte un estado global de como está funcionando el cultivo.

3. **Gráficas de barras en bloques.**

   Para comparaciones de máximos, mínimos y medias de medidas seleccionadas por el usuario y que pueden ser útiles su estudio en conjunto.

Y, en general, cualquier otro tipo de gráfica que se pueda llegar a implementar con Flutter, ya que éstas serían clave en la aplicación porque se tendría información útil al alcance de la mano.

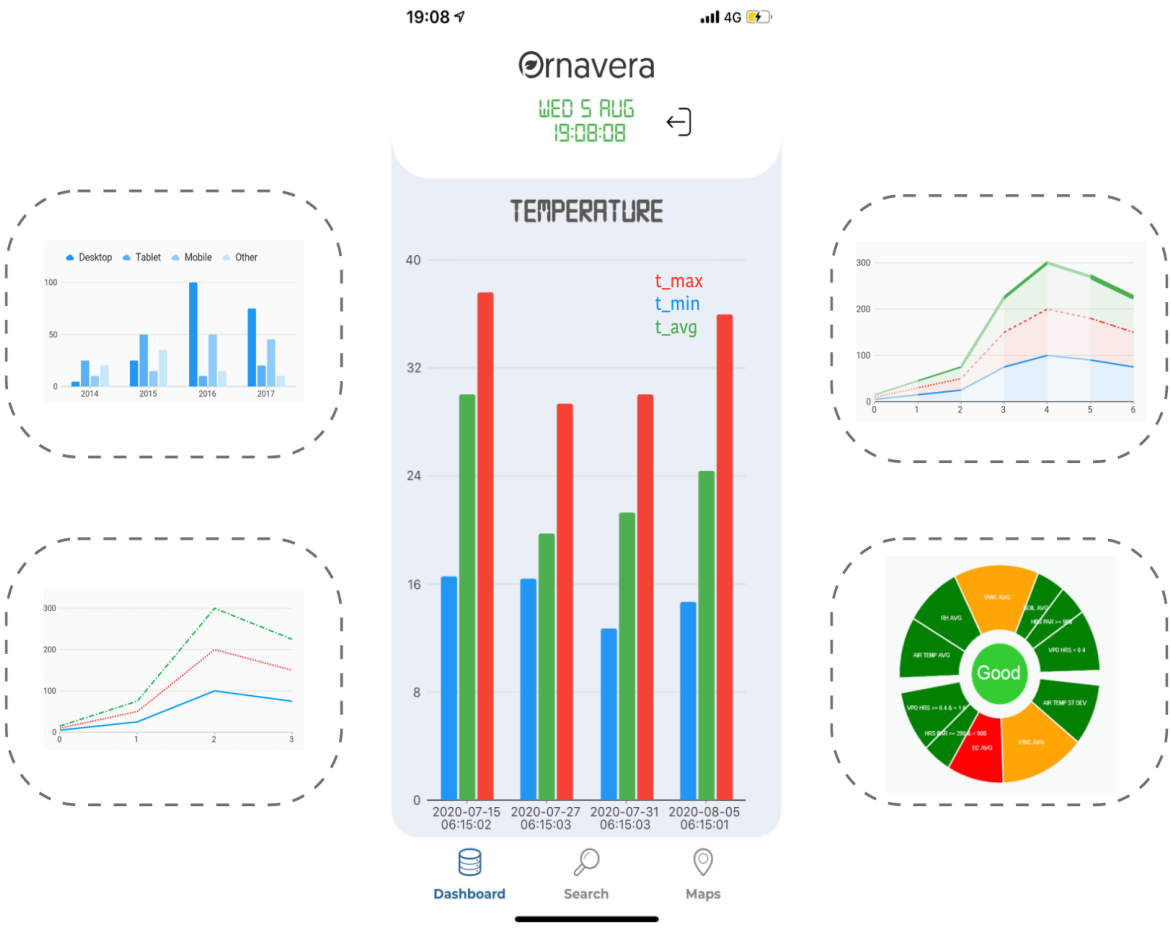


Figura 5.2.4 Nuevas pantallas con nuevos tipos de gráficas

INDICE DE FIGURAS

INDICE DE TABLAS

BIBLIOGRAFÍA

[1] N, Rafael, "*https://puentesdigitales.com/2018/03/28/el-iot-en-los-objetivos-de-desarrollo sostenible/*", puentesdigitales.com, 28 March 2018

[2] IEEE.org, "*https://standards.ieee.org/standard/2413-2019.html*", standards.iee.org

[3] Yu, Angela, *"The complete Flutter Development Bootcamp",* appbrewery.co, February 2020

[4] "*Flask for Beginners Tutorial — Learn Flask in 40 minutes (2019)*", Pretty Printed, Youtube, 21 March 2019

[5] "*Python Flask Tutorial For Beginners*", Edureka, Youtube, 18 December 2018

[6] "*Web Programming with Flask — Intro to Computer Science — Harvard's CS50 (2018)"* freeCodeCamp.org, Youtube, 29 January 2019

[7] Villalon, Salvador, *"How to build a web application using Flask and deploy it to the cloud",* FreeCodeCamp, 28 August 2018

[8] A, Damian, "*systemctl, trabaja con los servicios desde la terminal*", ubunlog.com, 24 March 2017

[9] Agapov, Oleg, "*JWT authorization in Flask*", Medium, 21 November 2017

[10] Thomsen, Michael, "*Beautiful, animated charts for Flutter*", Medium, 23 March 2018

[11] Ecalle, Thomas, "*An automatic Search Bar in Flutter*", Medium, 25 October 2019

[12] CQ, Eduardo, "*Google Maps en Flutter*", Medium, 20 April 2019

[13] G, Jayavigneshwaran, "*Introducing Data Visualization Widgets for Flutter*", syncfusion.com, 31 July 2019