# **2. Estado del arte/Estudio teórico**

El sistema planteado para este proyecto se compondrá de tres tecnologías de software: una aplicación para smartphone, una base de datos y una aplicación web. La finalidad de este capítulo es hacer un estudio de estas tecnologías, pasando por su evolución histórica y estudiando después alguna tecnología en particular. Esto servirá como base para justificar la elección de un stack más adelante.

**2.1. Smartphones**

## **2.1.1. Evolución histórica de aplicaciones para smartphones**

La evolución de las aplicaciones móviles ha estado estrechamente ligada al desarrollo del hardware de los teléfonos inteligentes y a la consolidación de sus sistemas operativos. Desde sus inicios hasta la actualidad, el desarrollo de software para dispositivos móviles ha pasado de ser una funcionalidad limitada a convertirse en un elemento central de la vida digital contemporánea, con millones de aplicaciones activas en distintas plataformas.

Los primeros antecedentes del software móvil pueden encontrarse en las PDAs (Personal Digital Assistants) de los años 90, como la Palm Pilot o la serie Psion, que permitían ejecutar aplicaciones básicas de calendario, notas o gestión de contactos. Estas aplicaciones eran estáticas, sin conectividad a Internet, y con interfaces muy simples. Posteriormente, algunos teléfonos móviles de gama alta comenzaron a incorporar funciones avanzadas y rudimentarias tiendas de aplicaciones, como el Nokia Ovi Store, que ofrecían juegos y utilidades sencillas.

Pero el verdadero punto de inflexión llegó en 2007, con la aparición del iPhone y el lanzamiento de iOS (en ese momento llamado iPhone OS). Apple introdujo por primera vez una interfaz multitáctil con una experiencia de usuario optimizada, y, en 2008, lanzó la App Store, que significó un gran cambio que lleva siendo práctico hasta hoy en día, permitiendo a desarrolladores externos distribuir aplicaciones de forma centralizada y sencilla.

Ese mismo año, Google lanzó la primera versión de Android, un sistema operativo móvil basado en Linux, y, al igual que iOS, lanzó también una tienda oficial: Google Play (inicialmente Android Market). Android se caracterizó desde el inicio por su naturaleza de código abierto y su disponibilidad en múltiples dispositivos fabricados por distintas marcas, y esto favoreció una rápida expansión global. Esta apertura dio lugar a una gran diversidad de dispositivos, versiones y configuraciones, lo que supuso un reto añadido para los desarrolladores, pero también una gran oportunidad de alcance.

Gracias a la existencia de estas tiendas oficiales que abrían la posibilidad a cualquier desarrollador a tener su aplicación propia en un mercado de grandes compañías, durante la década de 2010, el desarrollo de aplicaciones móviles se consolidó como una disciplina propia dentro del ámbito del desarrollo de software. Entonces surgieron entornos de desarrollo integrados (IDE) específicos como Xcode (para iOS) y Android Studio (para Android), y se introdujeron lenguajes de programación modernos como Swift (Apple, 2014) y Kotlin (JetBrains, adoptado oficialmente por Google en 2017), que simplificaron la creación de aplicaciones robustas y seguras.

Al mismo tiempo, se popularizaron nuevas arquitecturas de diseño orientadas a la experiencia de usuario (UX/UI), la modularidad y la mantenibilidad del código. También, con el paso de los años y la gran acogida que fue teniendo esto a nivel internacional, tanto en compañías como a nivel personal, se hizo una práctica muy habitual la integración con servicios en la nube, bases de datos remotas, APIs externas y funcionalidades avanzadas como geolocalización, reconocimiento facial, almacenamiento local, sensores, notificaciones push y machine learning, convirtiendo el desarrollo de aplicaciones en dispositivos móviles en un mercado gigante con infinitas posibilidades y adaptado a las necesidades concretas de cada usuario.

Es por esto que cada año más compañías y desarrolladores individuales se unen a la creación de aplicaciones, y los mercados como App Store y Google Play Store se convierten en una base de datos enorme de aplicaciones con funcionalidades tan distintas como intereses individuales existan. Cada día salen nuevos desafíos y tendencias, se pretende mejorar el rendimiento, mejorar de manera continua la experiencia de usuario, así como la seguridad de los datos o la integración con distintas nuevas tecnologías, como la IA, la realidad aumentada o los dispositivos wearables que hoy en día se pueden encontrar en muchos hogares.

Por último, en los últimos años, el avance tecnológico ha impulsado también el desarrollo de herramientas multiplataforma, como Flutter, React Native, Xamarin o Ionic, que permiten escribir una única base de código para generar aplicaciones tanto para Android como para iOS. Aunque estas soluciones ofrecen ventajas en cuanto a velocidad y coste de desarrollo, en muchos casos no alcanzan el mismo nivel de rendimiento o integración que las aplicaciones nativas, especialmente en lo relativo al acceso a funcionalidades específicas del dispositivo.

A continuación pasamos a ver más en detalle las características y algunos datos técnicos de uno de los gigantes actuales, Android.

**2.1.2. Android**

Android es un sistema operativo orientado a Smartphones que fue desarrollado inicialmente por Android Inc. y que posteriormente y hasta hoy en día forma parte de la compañía Google, que publicó su primera versión definitiva en 2008, aunque no fue hasta 2010, con el crecimiento en el uso de este tipo de dispositivos, que Android se convirtió en uno de los líderes mundiales en este sector.

Uno de los motivos que hacen de Android un sistema operativo líder es su naturaleza de código abierto, basada en el proyecto AOSP (Android Open Source Project). Esto facilita que cualquier compañía de fabricantes de smartphones (Xiaomi, Samsung…) pueda personalizar a su gusto cualquier parte del sistema sin la necesidad de pagar por licencias de uso. Esto ha facilitado que se puedan fabricar dispositivos Android en una variedad de gamas muy grande, desde dispositivos con funcionalidades básicas hasta smartphones muy potentes, lo que ha ayudado a que su comercialización sea internacional, al tener un abanico de precios tan grande.

Otra característica destacable de Android es su gran capacidad de escalabilidad y flexibilidad, siendo compatible con una gran variedad de dispositivos. a lo largo de los años Android ha sido utilizado en todo tipo de dispositivos emergentes que han salido al mercado como relojes inteligentes (Android Wear), televisores (Android TV), automóviles (Android Auto) e incluso dispositivos IoT.

**2.1.2.1. Estructura de Android**

El núcleo de Android está basado en Linux, con lo que se tienen servicios de seguridad, acceso a memoria, multiproceso o drivers de dispositivos.

Todas las capas subyacentes del sistema se comunican a través de este núcleo. Para garantizar el correcto funcionamiento del sistema operativo, se emplean entornos de ejecución basados en el lenguaje Java, como es el caso de Dalvik (y actualmente ART en versiones modernas), además de incorporar diversas bibliotecas desarrolladas en otros lenguajes como C y C++. Entre estas librerías destaca SQLite, un motor de bases de datos ligero ampliamente utilizado en el entorno móvil por su eficiencia y portabilidad.

En los niveles superiores del sistema se encuentran los entornos de aplicación, que constituyen la plataforma de acceso a las distintas funcionalidades del dispositivo. Estas capas superiores permiten el desarrollo de aplicaciones de manera libre y flexible. El sistema fue diseñado desde su origen para fomentar la reutilización de componentes, de esta forma distintas aplicaciones pueden beneficiarse e interactuar entre sí, aprovechando funcionalidades ya existentes y optimizando el desarrollo.

**2.1.2.2. Actualizaciones Android**

Android ha tenido una evolución constante desde su lanzamiento, traduciéndose en actualizaciones periódicas que mejoran el rendimiento y añaden funcionalidades, así como avances en seguridad e interfaz de usuario.

En el contexto del desarrollo en Android, es importante garantizar la compatibilidad entre versiones del sistema, evitando inestabilidad en distintos dispositivos.

En la siguiente tabla se muestran todas las versiones existentes a día de hoy, con su fecha de lanzamiento.

|  |  |  |
| --- | --- | --- |
| **Nombre de versión** | **Nº de versión** | **Fecha de lanzamiento** |
| [Apple Pie](https://es.wikipedia.org/wiki/Android#Android_1.0_(API_1))[[59]](https://es.wikipedia.org/wiki/Android#cite_note-nombres_oficiales_y_no_oficiales-59)​ | 1.0 | 23 de septiembre de 2008 |
| [Banana Bread](https://es.wikipedia.org/wiki/Android#Android_1.1_(API_2))[[59]](https://es.wikipedia.org/wiki/Android#cite_note-nombres_oficiales_y_no_oficiales-59)​ | 1.1 | 9 de febrero de 2009 |
| [Cupcake](https://es.wikipedia.org/wiki/Android_Cupcake) | 1.5 | 25 de abril de 2009 |
| [Donut](https://es.wikipedia.org/wiki/Android_Donut) | 1.6 | 15 de septiembre de 2009 |
| [Eclair](https://es.wikipedia.org/wiki/Android_Eclair) | 2.0 – 2.1 | 26 de octubre de 2009 |
| [Froyo](https://es.wikipedia.org/wiki/Android_Froyo) | 2.2 – 2.2.3 | 20 de mayo de 2010 |
| [Gingerbread](https://es.wikipedia.org/wiki/Android_Gingerbread) | 2.3 – 2.3.7 | 6 de diciembre de 2010 |
| [Honeycomb](https://es.wikipedia.org/wiki/Android_Honeycomb)[[60]](https://es.wikipedia.org/wiki/Android#cite_note-60)​ | 3.0 – 3.2.6 | 22 de febrero de 2011 |
| [Ice Cream Sandwich](https://es.wikipedia.org/wiki/Android_Ice_Cream_Sandwich) | 4.0 – 4.0.5 | 18 de octubre de 2011 |
| [Jelly Bean](https://es.wikipedia.org/wiki/Android_Jelly_Bean) | 4.1 – 4.3.1 | 9 de julio de 2012 |
| [KitKat](https://es.wikipedia.org/wiki/Android_KitKat) | 4.4 – 4.4.4 | 31 de octubre de 2013 |
| [Lollipop](https://es.wikipedia.org/wiki/Android_Lollipop) | 5.0 – 5.1.1 | 4 de noviembre de 2014 |
| [Marshmallow](https://es.wikipedia.org/wiki/Android_Marshmallow) | 6.0 – 6.0.1 | 29 de septiembre de 2015 |
| [Nougat](https://es.wikipedia.org/wiki/Android_Nougat) | 7.0 – 7.1.2 | 15 de junio de 2016 |
| [Oreo](https://es.wikipedia.org/wiki/Android_Oreo) | 8.0 – 8.1 | 21 de agosto de 2017 |
| [Pie](https://es.wikipedia.org/wiki/Android_Pie) | 9.0 | 6 de agosto de 2018 |
| [10](https://es.wikipedia.org/wiki/Android_10) | 10.0 | 3 de septiembre de 2019 |
| [11](https://es.wikipedia.org/wiki/Android_11) | 11.0 | 8 de septiembre de 2020 |
| [12](https://es.wikipedia.org/wiki/Android_12) | 12.0 - 12L | 4 de octubre de 2021 |
| [13](https://es.wikipedia.org/wiki/Android_13) | 13.0 | 15 de agosto de 2022 |
| [14](https://es.wikipedia.org/wiki/Android_14) | 14.0 | 4 de octubre de 2023 |
| [15](https://es.wikipedia.org/wiki/Android_15) | 15.0 | 15 de octubre de 2024 |
| [Baklava](https://es.wikipedia.org/wiki/Android_16)[[61]](https://es.wikipedia.org/wiki/Android#cite_note-61)​ | 16.0 | 18 de noviembre de 2024 |

Tabla 1: Versiones Android y fecha de lanzamiento

Es recomentable, también, a la hora del desarrollo, conocer qué versiones de Android son más utilizadas en los dispositivos que hay en el mercado. En la siguiente tabla se muestra la cuota de mercado de las distintas versiones hasta agosto de 2024.

|  |  |
| --- | --- |
| Versión | Cuota de mercado (%) |
| Android 14 | 30.84 |
| Android 13 | 20.57 |
| Android 12 | 15.09 |
| Android 11 | 13.32 |
| Android 10 | 7.23 |
| Android 9 | 4.63 |
| Android 8 | 2.65 |
| Android 7 | 1.21 |
| Otros | 3.54 |

Tabla 2: Cuota de mercado de las versiones Android

**2.1.2.3. Estructura fundamental**

**2.1.2.3.1. Actividad**

Para entender la estructura fundamental de una aplicación Android es fundamental introducir el concepto de Actividad.

Una Actividad (Activity) es uno de los componentes principales del ciclo de vida de cualquier aplicación Android. La finalidad de este componente es proporcionar una pantalla para realizar la interacción con el usuario. Se podría entender como cualquier parte visual del sistema, ya sea un formulario, una pantalla de inicio, o cualquier acción, como tomar una foto o enviar un correo electrónico.

El cambio de ventanas se maneja mediante Intents (que se explicarán más adelante), los cuales se almacenan en una pila. Cada actividad que se abre se pausa a la espera de ser retomada, lo que da lugar a la llamada "pila de retroceso".

Las actividades tienen cuatro posibles estados que están vinculados a esta pila:

* **Running (Activo):** La actividad está en primer plano, mostrándose al usuario y situada en la cima de la pila.
* **Paused (Pausado):** La actividad está en segundo plano porque otra actividad está en primer plano. Si otra actividad se vuelve activa, esta pasa al siguiente estado.
* **Stopped (Parada):** La actividad no es visible y su estado debe ser guardado para su futura recuperación.
* **Destroyed (Destruida):** La actividad ha sido eliminada definitivamente y no puede ser recuperada, ya que ha salido de la pila.

Para manejar estos estados, las actividades pueden definir las acciones a seguir mediante los siguientes métodos:

* **onCreate():** Se llama una vez al crear la actividad.
* **onRestart():** Se llama cuando la actividad vuelve al primer plano.
* **onStart():** Se llama cuando la actividad se vuelve visible al usuario.
* **onResume()**: Se llama cada vez que el usuario vuelve a interactuar con la actividad.
* **onPause():** Se llama cuando otra actividad pasa al primer plano.
* **onStop():** Se llama cuando la actividad deja de ser visible.
* **onDestroy():** Se llama cuando la actividad está finalizando, ya sea porque el método *onFinish()* fue llamado o para liberar memoria.

En el siguiente diagrama vemos una representación del ciclo de vida de una actividad en Android.

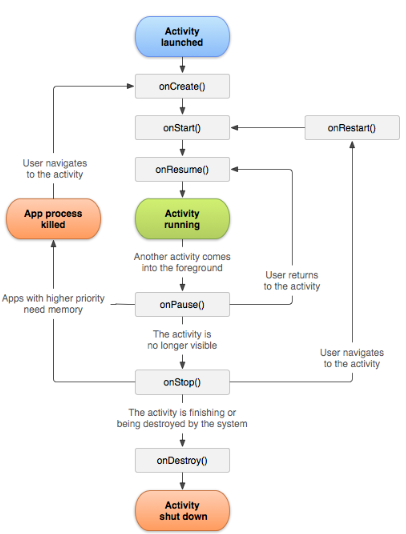


Figura 1: Ciclo de vida de una actividad Android

**2.1.2.3.2. Intents**

En Android, Intent es un mecanismo utilizado para la comunicación entre los distintos componentes del sistema. Se entiende como un mensaje asíncrono que permite solicitar una acción, que puede ser dentro de la aplicación o entre aplicaciones.

Existen dos tipos de intents:

* Explícitos: Utilizados para iniciar un componente específico, que se debe especificar al crearlo
* Implícitos: No se especifica ningún componente, sino que se llama a una acción general y será el propio sistema el encargado de resolverlo

**2.1.2.2 Comparativa con otras opciones**

Frente a otras alternativas vistas anteriormente, como iOS, Android presenta una curva de entrada más accesible. iOS requiere equipos Apple y la compatibilidad con otro tipo de dispositivos es muy reducida. Además, Android no necesita ninguna licencia para publicar aplicaciones en su tienda oficial (Google Play Store), como sí se necesitan en la tienda de Apple (App Store), además de tener un proceso de revisión más estricto y políticas de calidad exigentes.

Por otro lado, en cuanto a las tecnologías multiplataforma, se presentan como una gran alternativa al desarrollo nativo. Aún así, actualmente, desarrollar directamente en Android ofrece un rendimiento mayor y mejor eficiencia, ya que permite interactuar con el sistema operativo directamente y no utiliza una capa intermedia, aprovechando de esta manera mejor los recursos que ofrece el dispositivo. Además, se permite un control total sobre aspectos vistos en el anterior apartado como el ciclo de vida de la aplicación, el manejo de actividades, así como una integración más profunda con servicios de Google como Firebase o Google Maps

## **2.2. Evolución histórica de bases de datos**

Se pasa ahora a estudiar una parte fundamental de la informática actual, las bases de datos, que nacieron por la necesidad de tener un sistema que almacene los datos y facilite su acceso desde cualquier tipo de sistema.

El término de base de datos ha ido cambiando enormemente desde su origen en la segunda parte del siglo XX, manteniéndose siempre ajustado tanto a las demandas tecnológicas como empresariales de cada momento. El cambio ha estado caracterizado por el paso de modelos de almacenamiento sencillos y rígidos hasta que se convirtieran en cada vez más volátiles, distribuidos e inmersos en la nube, capaz de manejar enormes volúmenes de datos de modo eficaz y eficiente.

Los primeros modelos de gestión de bases de datos aparecen en los años 60, en un período en el que informática se hallaba en el comienzo de su proceso de desarrollo. En esta primera etapa se dominaban los modelos jerárquico y de red, que estructuraban la información según esquemas de relaciones padre-hijo o complejas redes de nodos interconectados. Estos modelos ofrecían cierta eficiencia en contextos muy controlados, pero contaban con un enorme problema de escalabilidad, de flexibilidad y de mantenimiento que hacía que no fueran útiles en proyectos diferentes.

El verdadero punto de inflexión se produjo en los años 70 con el invento del modelo relacional, ideado por Edgar F. Codd. Se basó en la representación de los datos en tablas (relaciones), su manipulación por medio de un lenguaje formal, el Structured Query Language (SQL). El modelo relacional simplificó de un modo notable cómo se interaccionaba con los datos y fomentó una separación entre estructura física y de carácter lógica. Gracias a este nuevo sistema, en las posteriores décadas nacieron importantes sistemas de gestión de bases de datos relacionales (RDBMS), como Oracle, IBM DB2, Microsoft SQLServer y PostgreSQL, que se convirtieron en columnas centrales de la informática empresarial, que a día de hoy siguen siendo importantes sistemas utilizados en todo el mundo para la gestión de datos.

Con el paso de los años y el crecimiento de Internet surgen nuevas demandas por escalabilidad, disponibilidad y elasticidad. En este ámbito, desde mediada la década de 2000 se iniciaron el desarrollo de las llamadas bases de datos NoSQL (“Not Only SQL”), con el fin de eliminar algunas de las restricciones que eran difíciles de solventar con los robustos sistemas de modelos relacionales cuando los datos que se tenían eran de naturaleza no estructurada. Base de datos como MongoDB, Cassandra o CouchDB nos permiten acceder a modelos de documentos, grafos o pares clave-valor, y se encuentran ampliamente adoptadas en las aplicaciones que procesan enormes volúmenes de datos o necesitan alta disponibilidad en tiempo real.

En paralelo con el crecimiento de los servicios en la nube se produjo un nuevo paradigma en el almacenamiento de datos y gestión de ellos: el Database-as-a-Service (DBaaS). Este, se trata de un modelo que permite que los desarrolladores puedan trabajar con bases de datos sin darle importancia a cómo se mantienen ni su infraestructura, delegando estos aspectos en proveedores especializados. Firebase Realtime Database, Amazon DynamoDB, Google Cloud Firestore o Supabase son ejemplos de esta nueva tendencia de soluciones, en las que se convierten en aspectos importantes la escalabilidad automática, el tiempo de respuesta en tiempo real y comunicación con aplicaciones web como aplicaciones móviles.

En el presente, las bases de datos cumplen con el mismo papel de almacenar información, pero también se integran con análisis, inteligencia artificial e automatización. También crece el interés por conceptos como el de seguridad, escalabilidad horizontal y eficiencia energética en el proceso de trabajo de los datos.

A continuación se va a ver más en detalle la base de datos Firebase, para hacer posteriormente una comparación con otra tecnología con un modelo diferente.

### **2.3. Firebase**

**Firebase** es una plataforma de desarrollo de aplicaciones creada por Firebase Inc. en 2011 y adquirida por Google en 2014. Desde entonces, se ha consolidado como una solución integral utilizada por una gran cantidad de desarrolladores, que ofrece un amplio conjunto de herramientas y servicios orientados, entre otras cosas, al desarrollo de aplicaciones web y móviles. Entre sus funcionalidades más destacadas se encuentran la autenticación de usuarios, el alojamiento web, el almacenamiento de archivos, las funciones en la nube y, especialmente, su sistema de bases de datos en tiempo real.

Firebase cuenta con dos sistemas principales de bases de datos: Firebase Realtime Database y Cloud Firestore. Ambas soluciones están orientadas al almacenamiento de datos en la nube y siguen un modelo NoSQL basado en documentos. A diferencia de las bases de datos relacionales tradicionales, Firebase organiza la información en colecciones y documentos jerárquicos, lo que permite una mayor flexibilidad y escalabilidad en aplicaciones que requieren datos dinámicos y estructuras cambiantes.

Una de las principales características que distingue a Firebase es su capacidad para ofrecer sincronización en tiempo real. Cuando un dato se actualiza en la base de datos, todos los clientes conectados reciben inmediatamente esa actualización, sin necesidad de realizar consultas adicionales. Esta funcionalidad resulta especialmente útil en aplicaciones que requieren interactividad continua entre usuarios o visualización en directo de los datos, como chats, paneles colaborativos, o sistemas de notificación.

Firebase también facilita la integración con otros servicios del ecosistema de Google, como Firebase Authentication (para el registro y gestión segura de usuarios), Firebase Hosting (para alojar aplicaciones web estáticas), Firebase Cloud Functions (para ejecutar lógica del lado del servidor sin necesidad de configurar servidores propios) y Firebase Analytics (para el análisis del comportamiento de los usuarios). Todo esto se encuentra dentro de una arquitectura serverless, en la que el desarrollador puede centrarse en la lógica de negocio sin necesidad de gestionar la infraestructura.

En el contexto de un trabajo que combine una aplicación web con una aplicación para smartphones, Firebase representa una elección especialmente adecuada por varias razones:

* **Compatibilidad multiplataforma**: Firebase proporciona SDKs específicos para **JavaScript**, **Android** e **iOS**, lo que permite compartir una única base de datos entre ambas aplicaciones (web y móvil) sin necesidad de crear APIs intermedias.
* **Desarrollo ágil y escalable**: Gracias a su enfoque gestionado (DBaaS), Firebase permite desplegar soluciones rápidamente, con un mantenimiento mínimo por parte del desarrollador. La plataforma se encarga automáticamente de la escalabilidad, el rendimiento y la disponibilidad, lo cual es especialmente valioso en proyectos de pequeña escala, que buscan una implementación funcional y eficiente.
* **Actualización y sincronización en tiempo real**: Esta funcionalidad garantiza que los datos visualizados por los usuarios en distintos dispositivos estén siempre sincronizados, lo cual mejora notablemente la experiencia del usuario. Por ejemplo, si un usuario modifica una información desde la app móvil, ese cambio se refleja instantáneamente en la interfaz web, y viceversa.
* **Seguridad y control de acceso**: Firebase permite definir reglas de seguridad basadas en la autenticación del usuario y en la estructura de los datos, lo que proporciona un control preciso sobre quién puede leer o modificar cada parte del contenido.
* **Integración con funcionalidades adicionales**: La plataforma ofrece extensiones como almacenamiento de archivos multimedia, notificaciones push, funciones programables o análisis de comportamiento, lo que facilita la creación de aplicaciones completas sin necesidad de depender de múltiples proveedores externos.

Es importante mencionar que Firebase cuenta con un plan gratuito (Spark) que es bastante completo para pruebas o aplicaciones pequeñas. Además, puede ser una gran utilidad en aplicaciones donde no se tenga claro si este modelo de base de datos es el más eficiente o casos donde una aplicación empiece siendo pequeña pero tenga expectativas de crecer as lo largo del tiempo, porque se puede pasar del plan gratuito al plan de pago, añadiendo recursos y mejorando la escalabilidad de manera notable.

Como vemos, Firebase se presenta como una solución moderna y robusta para el desarrollo de aplicaciones que requieran conectividad entre múltiples plataformas y especialmente eficaz para aplicaciones que no requieren una cantidad y complejidad de datos muy grande. Además, su facilidad de uso, integración con herramientas clave y capacidad para sincronizar datos en tiempo real la convierten en una elección idónea para proyectos que combinan aplicaciones web y móviles, como es el caso de este proyecto.

**2.3.1. Comparativa**

Como se ha visto, el desarrollo en sistemas SQL supone una gran opción por su robustez y tiene una gran capacidad para manejar estructuras de datos complejas.

No obstante, los sistemas NoSQL, como es Firebase, ofrecen claras ventajas para integraciones directas o agilidad de desarrollo. Estos sistemas no requieren esquemas fijos, lo que facilita adaptarse a cambios en la estructura de datos a lo largo del desarrollo. Además, al tratarse de un servicio completamente gestionado, elimina la necesidad de configurar un servidor backend o desarrollar una API intermediaria, algo que sí sería necesario al utilizar sistemas SQL, como puede ser PostgreSQL, que requiere una arquitectura más compleja para su integración con aplicaciones frontend.

## **2.1. Evolución histórica de desarrollos web**

El desarrollo web ha experimentado una gran evolución desde los sistemas más simples de los años 60 y 70 hasta los más actuales, que proporcionan diversas opciones y herramientas para la interactividad del usuario. Esta evolución ha estado estrechamente relacionada con los avances en tecnologías de frontend, backend, bases de datos, y más recientemente desde 2008, con la integración móvil.

Los primeros sistemas de comunicación en línea se remontan a ARPANET, Usenet y BBS, hasta que en 1989, Tim Berners-Lee inventó el HyperText Marup Language (HTML) y la World Wide Web (WWW). Pero no fue hasta un año después, en 1990, que no se desarrolló el primer sitio web. Esto ocurrió en el CERN, el Laboratorio Europeo de Física de Partículas. Por lo tanto, los inicios de la Web se encuentran entre 1990 y 1995, donde se observa un HTML estático, siendo únicamente documentos de este tipo de lenguaje simples, sin estilos y sin interactividad, sin bases de datos integradas ni programación del lado del servidor, por lo que eran sitios meramente informativos. En los inicios de la Web se usaban navegadores como Mosaic y Netscape.

En cuanto al diseño, las primeras páginas web eran totalmente estáticas, permitiendo entonces pocas posibilidades de interactividad y de estilos. En 1996 fue cuando empezaron a poder verse las primeras páginas webs parecidas a las que encontramos actualmente. En 2005 ganó popularidad CSS, que permite separar los elementos de contenido con los de diseño. En 2008 llegaron los smartphones, por lo que el diseño web tuvo que saber adaptarse a las pantallas móviles. Actualmente, el diseño web se centra en la experiencia del usuario (UX, UI)

En general, la evolución de la web comenzó con la Web 1.0 introduciendo al mundo el concepto de “en línea”, la Web 2.0 añadió la interactividad y las redes sociales, la Web 3.0 trajo consigo los datos inteligentes y personalizados y la Web 4.0 es la más actual y la que trabaja para revolucionar la experiencia web con avances como la inteligencia artificial.

La era del servidor y las primeras bases de datos, la Web 1.0, se puede situar entre 1995 y 2005, puesto que aparecieron diversos avances. Por un lado, lenguajes del lado del servidor tales como PHP, ASP de Microsoft, Perl, ColdFusion y conexiones a bases de datos como MySQL, PostgreSQL y SQL Server. Por otro lado, JavaScript salió a la luz permitiendo las primeras interacciones en el cliente y aparecieron también los primeros sistemas de gestión de contenido. En esta época se consolida el modelo cliente-servidor.

De 2005 a 2012 se da la llamada Web 2.0, donde el uso global de AJAX supuso la posibilidad de generar sitios web más dinámicos sin la necesidad de recargas páginas. En esta era nacieron plataformas conocidas actualmente como redes sociales: Facebook, YouTube, Twitter (ahora llamado X). También hubo un aumento en la frecuencia de uso del APIs REST para conectar servicios y se desarrollaron frameworks JavaScript (jQuery, Prototype.js) y avances en diseño web (CSS2/3, HTML5).

De 2012 a 2020, en la era de la Web 3.0, se puede observar la aparición de Single Page Applications (SPA), que permite a sus aplicaciones cargar una sola página HTML y actualizar su contenido dinámicamente. Los frameworks populares de esta era son Angular (Google), React (Facebook) y Vue.js. Backend con Node.js y arquitecturas como MVC y RESTful APIs. En esta parte del desarrollo web, hay un masivo de JSON para intercambio de datos, y bases de datos NoSQL: MongoDB, Firebase, CouchDB; contenedores (como Docker) y DevOps.

En cuanto al desarrollo web actual, este comprende la era de la Web 4.0, desde 2020 hasta el día de hoy, donde se pueden encontrar nuevos avances tales como el Jamstack, Serveless architecture o WebAssembly (WASM). Jamstack supone un enfoque moderno basado en JavaScript, APIs y Markup; Serverless architecture que permite que las funciones se ejecuten sin necesidad de gestionar un servidor; y WASM que permite la ejecución de un código de alto rendimiento en el navegador. En esta era es necesario recalcar el uso reciente de la Inteligencia Artificial y la integración con Machine Learning, al igual que el diseño “mobile-first” y desarrollo progresivo (PWA). También se está llevando a cabo una integración más profunda con aplicaciones web a través de APIs compartidas, bases de datos en la nube y herramientas como Flutter, React Native o Capacitor, que permiten desarrollar aplicaciones móviles multiplataforma con una sola base de código, optimizando tiempos y recursos.

**2.2. Tecnologías**

Como se ha visto, dentro de las tecnologías actuales de desarrollo web se pueden encontrar diversas opciones, dentro de las cuales son especialmente interesantes y populares Angular y React, por lo que se ven en más detalle en los siguientes apartados.

**2.2.1. Angular**

Angular es desarrollada por Google, siendo su lanzamiento inicial en 2010 con el nombre de AngularJS y reescrito como Angular 2+ en 2016, pero actualmente se llama Angular simplemente. Usa un framework completo de desarrollo frontend y su lenguaje principal es TypeScript, que extiende de JavaScript pero añade un lenguaje tipado. Sus características más destacadas son su arquitectura basada en componentes, sus inyecciones de dependencias, su enrutador integrado, su sistema de formularios reactivos y plantillas dinámicas, su CLI oficial para generación y mantenimiento de proyectos y su soporte completo para un desarrollo a gran escala.

A continuación se desglosan una serie de características técnicas de Angular que serán fundamentales para la elección de tecnología en este proyecto:

* **Componentes**: Angular basa su arquitectura en componentes, lo que facilita el diseño modular y reutilizable del código, y es útil para la mantenibilidad y escalabilidad de las aplicaciones.
* **Lenguaje**: Angular utiliza TypeScript, lenguaje basado en JavaScript, tipado y estructurado facilitando un desarrollo robusto de las aplicaciones.
* **Data binding bidireccional**: Sincroniza el modelo y la vista con actualizaciones automáticas que aceleran y facilitan al desarrollador el proceso de trabajo.
* **Dependencias**: Angular funciona con un sistema integrado de inyección de dependencias, pudiendo ser estas servicios u objetos compartidos.
* **CLI oficial**: Utiliza Angular CLI como herramienta de línea de comandos para ayudar a la generación de código, la automatización de tareas o la configuración inicial del proyecto.
* **Compilación**: Angular utiliza Ahead-of-Time (AOT) como método de compilación, lo que hace que se compile antes de ejecutarse en el navegador y mejora el tiempo de carga de las aplicaciones.
* **Testing**: Angular tiene de forma nativa frameworks de testing unitarios y e2e como Jasmine o Karma, evitando la necesidad de usar un framework a parte y facilitando este proceso.
* **Licencia**: Angular está distribuido bajo licencia MIT, lo que permite hacer uso libre de esta tecnología, y su código fuente está enteramente alojado en GitHub.
* **Comunidad**: Al ser OpenSource, Angular cuenta con una gran comunidad e información en internet que ayudan al desarrollador a solucionar errores comunes de manera más rápida.

**2.2.2. React**

React, por otro lado, está desarrollado por Meta y su lanzamiento inicial fue en 2013. El tipo de framework es biblioteca (library) de UI, enfocada principalmente en la vista, y su lenguaje principal es JavaScript (con JSX). Sus características clave son su arquitectura basada en componentes funcionales o de clase, el uso de JSX para mezclar HTML con JavaScript, su flujo de datos unidireccional, su virtual DOM para un renderizado eficiente y su gran ecosistema de librerías complementarias como Redux o React Router. Es más flexible y minimalista que Angular.

A continuación se desglosan, al igual que se ha hecho en Angular, una serie de características técnicas de React.

* **Enfoque declarativo**: Lo que ayuda a describir cómo debe verse la interfaz de usuario en cada estado, y permite construirlas de forma fácil de depurar.
* **Componentes**: Al igual que Angular, hace uso de componentes.
* **Sintaxis**: Usa una sintaxis propia: JSX (JavaScript XML)
* **Flujo de datos**: En React el flujo de datos es unidireccional, de padre a hijo, facilitando el mantenimiento.
* **Integración con bibliotecas**: React es muy modular, lo que favorece la integración con una gran cantidad de librerías que se adapten a cada proyecto.
* **Licencia**: Usa, al igual que Angular, licencia MIT, y su código fuente está alojado en GitHub.
* **Comunidad**: Al ser OpenSource, React cuenta con una gran comunidad en foros de internet, tanto oficiales como no oficiales.

**2.2.3. Comparativa**

Tanto Angular como React son dos de las tecnologías de mayor uso en desarrollo de aplicaciones web actuales, fundamentalmente en proyectos que necesitan una arquitectura de tipo Single Page Application (SPA).

En un escenario de crecimiento donde se trabaja mediante tecnologías de aplicaciones móviles y bases de datos como backend en la nube, ambas tecnologías son muy útiles y eficientes, no obstante.Angular brinda una integración directa y bien documentada con dichos servicios, ayudada además por la ventaja de que todas estas tecnologías están respaldadas por Google. Esto se traduce en mayores niveles de coherencia en la documentación, en compatibilidad entre SDKs y en tener soluciones oficiales y establecidas para la integración de funcionalidades como Firestore, autenticación o despliegue en tiempo real. Esta alineación tecnológica puede ayudar a reducir fricciones durante la creación y a facilitar la conversión a través de lenguaje común en la conversación entre frontend web y servicios móviles o backend.

Para ver de manera más esquemática las diferencias entre tecnologías, se presenta una tabla comparativa con algunas de las características mencionadas.

|  |  |  |
| --- | --- | --- |
| **Aspecto** | **Angular** | **React** |
| **Desarrollador** | Google | Meta (Facebook) |
| **Licencia** | MIT (open source) | MIT (open source) |
| **Lenguaje principal** | TypeScript | JavaScript + JSX |
| **Basado en componentes** | Sí | Sí |
| **Uso en SPA (Single Page Apps)** | Muy adecuado | Muy adecuado |
| **Curva de aprendizaje** | Media-alta (estructura más rígida) | Media-baja (más libre y gradual) |
| **Reutilización de componentes** | Sí | Sí |
| **Enfoque de datos** | Bidireccional | Unidireccional |
| **Gestión del estado** | RxJS, NgRx, servicios inyectables | Redux, Context API, Zustand, Recoil, etc. |
| **Ruteo (Routing)** | Incluido de forma nativa | Mediante librerías externas |
| **Testing** | Integrado con herramientas como Karma/Jasmine | Compatible con Jest, React Testing Library, etc. |
| **CLI oficial** | Sí, Angular CLI | Sí, como Create React App o Vite |
| **Documentación y soporte** | Muy completa, con comunidad muy activa | Muy completa, con comunidad muy activa |
| **Popularidad** | Popular en empresas grandes y proyectos escalables | Popular en startups y grandes plataformas |
| **Integración con mobile (nativo)** | Compatible con frameworks externos (ej. Ionic) | Compatible directamente con React Native |
| **Actualizaciones** | Mantenido por Google con ciclos regulares | Mantenido por Meta |

Tabla 3 - Comparativa entre Angular y React