https://es.linkedin.com/pulse/tecnolog%C3%ADas-actuales-para-el-desarrollo-de-m%C3%B3viles-tello-villalobos

Enfoques para el desarrollo de aplicaciones móviles

Hoy en día existen distintos enfoques para poder enfrentar el desarrollo de aplicaciones móviles:

- *Aplicaciones Nativas*
- *Aplicaciones Móviles web*
- *Aplicaciones Progresivas web (PWA)*
- *Aplicaciones Híbridas*
- *Aplicaciones de Compilación cruzada*
- *Aplicaciones Interpretadas o de Scripting nativo*

Esta diversidad de alternativas muestra una evolución positiva en el desarrollo de aplicaciones móviles, en el sentido que en sus orígenes solo se podían construir para un sistema operativo específico, usando herramientas y lenguajes propios para esos sistemas (forma nativa). Con el paso del tiempo, surgen otras formas de desarrollar aplicaciones móviles, que permiten en un mismo código escrito en un solo lenguaje, tener una aplicación para diferentes sistemas operativos.

Aplicaciones Nativas

El desarrollo nativo implica desarrollar con el lenguaje o framework específico para cierto sistema operativo móvil. Existen varios sistemas operativos móviles, donde algunos se pueden ver en la sgte. tabla junto a las habilidades de programación requeridas:

Tipo de SO móvil	Conjunto de habilidades requeridas	Fuente
iOS	Swift	[Apple Inc., 2016]
Android	Java, Kotlin	[Android Developers, 2020]
Tizen	C, C++ o CSS/HTML/Javascript	[Tizen, 2012]
KaiOS	CSS/HTML/Javascript	[KaiOS, 2020]
Sailfish OS	QML, C++, Python	[SailfishOS, 2020]
Windows Mobile	.NET	[Charland y Leroux, 2011]
Windows 7 Phone	.NET	[Charland y Leroux, 2011]
RIM Blackberry	Java	[Charland y Leroux, 2011]
Symbian	C, C++, Python, HTML/CSS/Javascript	[Charland y Leroux, 2011]
HP Palm webOS	HTML/CSS/Javascript	[Charland y Leroux, 2011]
MeeGo	C, C++, HTML/CSS/Javascript	[Charland y Leroux, 2011]
Samsung bada	C++	[Charland y Leroux, 2011]
Ubuntu Touch	QML, HTML5	[Osman, 2014]
KDE Plasma 5	QML	[KDE, 2020]

https://es.linkedin.com/pulse/tecnolog%C3%ADas-actuales-para-el-desarrollo-de-m%C3%B3viles-tello-villalobos

Actualmente el mercado se puede dividir en 3 partes: **Android, iOS y otros**. La principal ventaja del desarrollo nativo es: el rendimiento (existe vasta literatura que estudia el rendimiento de apps nativas vs apps no-nativas, pero escapa del alcance de este artículo, se recomienda leer algunas de las referencias al final), como también la experiencia de usuario ya que se utilizan los componentes ofrecidos por los sistemas de diseño de cada plataforma. Debido a que el rendimiento es óptimo, es recomendado para aplicaciones grandes y complejas.

Sin embargo, la principal desventaja es tener que enfocarse en una plataforma a la vez, por lo cual puede ser necesario tener un equipo desarrollando para Android y otro para iOS implicando mayores costos a nivel de gestión y también mayor esfuerzo para el equipo de diseño al tener que diseñar diferentes flujos para las diferentes plataformas.

Aplicaciones Móviles web

Las aplicaciones móviles web no son aplicaciones nativas que pueden descargarse y ejecutarse, sino una aplicación web normal que fue adaptada al formato móvil. Diseñadas para ejecutarse dentro de una aplicación móvil de navegador (por ejemplo: Chrome, Firefox, Safari, etc.), son desarrolladas con tecnologías web estándar (HTML5, CSS3, Javascript), con un patrón de diseño **MVC** normalmente, son alojadas en servidores remotos, entregadas a través de protocolos estándar (por ejemplo: HTTP) y presentan varias ventajas**.**

- No requieren adaptación a ningún entorno operativo.
- Son fáciles y rápidas de ejecutar.
- Una sola aplicación entrega una experiencia de usuario uniforme a lo largo de las distintas plataformas ofreciendo un desarrollo rápido, mantención simple y portabilidad completa de la aplicación.

Sin embargo, estas aplicaciones son menos atractivas porque no dan la sensación de estar en una aplicación nativa, la interacción cliente-servidor tiene mayor latencia que las aplicaciones nativas, tienen restricciones sobre el acceso a las capacidades nativas del dispositivo y además presentan riesgos de seguridad debido a que el código es ejecutado a través del navegador.

Aplicaciones Progresivas web (PWA)

Para compensar la apariencia y experiencia de un sitio web, las aplicaciones progresivas web (PWA) fueron introducidas por Google. Las PWA mejoran las aplicaciones tradicionales web con los llamados service workers (permitiéndoles correr código en un thread de segundo plano), un manifiesto de la aplicación web (para proveer metadata), capacidades off-line y una experiencia de usuario similar a una instalación.

https://es.linkedin.com/pulse/tecnolog%C3%ADas-actuales-para-el-desarrollo-de-m%C3%B3viles-tello-villalobos

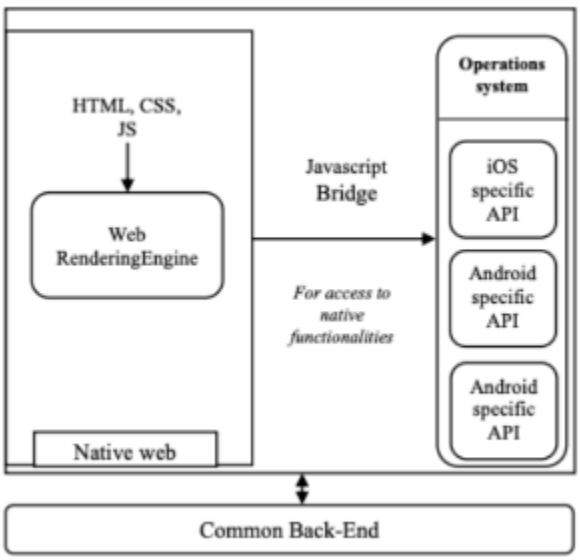
Si bien una PWA tiene acceso a privilegios del dispositivo y la plataforma más allá de lo que es típico para una aplicación móvil web, todavía son limitadas en términos de acceso a funciones: una PWA no puede acceder a funciones de dispositivo o de plataforma no expuestas a través del navegador web que se ejecuta dentro:

Feature	Interpreted	PWA	Hybrid	Native
Installable	Yes	Yes[a]	Yes	Yes
Offline capable	Yes	Yes	Yes	Yes
Testable before installation	No	Yes	No	No
App marketplace availability	Yes	Yes[b]	Yes	Yes
Push notifications	Yes	Yes[c]	Yes	Yes
Cross-platform availability	Yes	Limited[d]	Yes	No
Hardware and Platform API access	Yes	Limited[e]	Yes[f]	Yes
Background synchronisation	Yes	Yes	Yes	Yes

Existe una gran cantidad de frameworks para el desarrollo web (**React.js** por ejemplo), el grupo Web Fundamentals demostró el agnosticismo del framework mediante la implementación de PWA en tres frameworks diferentes.

Aplicaciones Híbridas

Una aplicación híbrida es una aplicación que ni es una verdadera aplicación móvil web ni una aplicación nativa. Es básicamente un enfoque que usa el motor del navegador en el dispositivo y sincroniza el contenido en HTML, CSS y Javascript en contenedores web nativos tales como, **WebView** en Android y **WKWebView** en iOS.



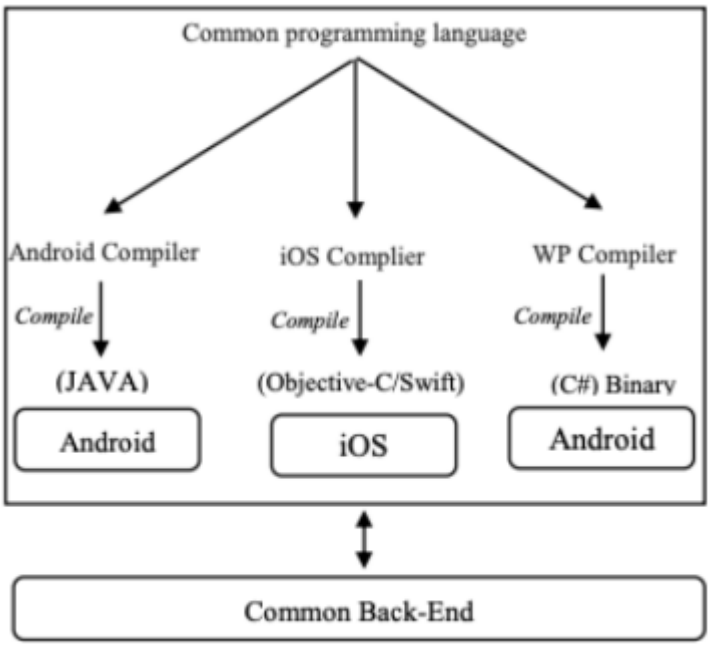
https://es.linkedin.com/pulse/tecnolog%C3%ADas-actuales-para-el-desarrollo-de-m%C3%B3viles-tello-villalobos

Estos contenedores web tienen acceso a funcionalidades específicas de las plataformas a través de APIs, es así que las aplicaciones híbridas brindan acceso a funcionalidades nativas del dispositivo y permiten reducir tanto los tiempos como los costos de desarrollo y mantención, ya que produce una única base de código para múltiples plataformas y hardware; pero la experiencia de usuario puede ser desagradable debido a la falta de uso de componentes nativos en la interfaz además de la lentitud de la aplicación relacionada a cargar los contenedores web.

A diferencia de las aplicaciones móviles web, a las que se accede mediante un navegador, las aplicaciones híbridas son distribuidas mediante app stores. Algunos ejemplos de frameworks para el desarrollo de aplicaciones híbridas son **PhoneGap**, **Ionic**, **Framework7**, **JQuery Mobile**, **Onsen UI**, **Mobile Angular UI**, **Chocolate Chip UI** y **Apache Cordova**.

Aplicaciones de Compilación cruzada

Una aplicación de compilación cruzada es una aplicación escrita en lenguaje no-nativo la cual puede ser compilada en una aplicación totalmente nativa usando un compilador cruzado.



Fuente: [Latif et al., 2016]

La aplicación entera es desarrollada utilizando un framework de compilación cruzada, el cual será capaz de compilar el binario nativo correcto para las distintas plataformas. Ya que el código es compilado en archivos específicos para cada plataforma, componentes nativos reales pueden ser usados y por lo tanto conseguir un verdadero sentimiento nativo en la aplicación.

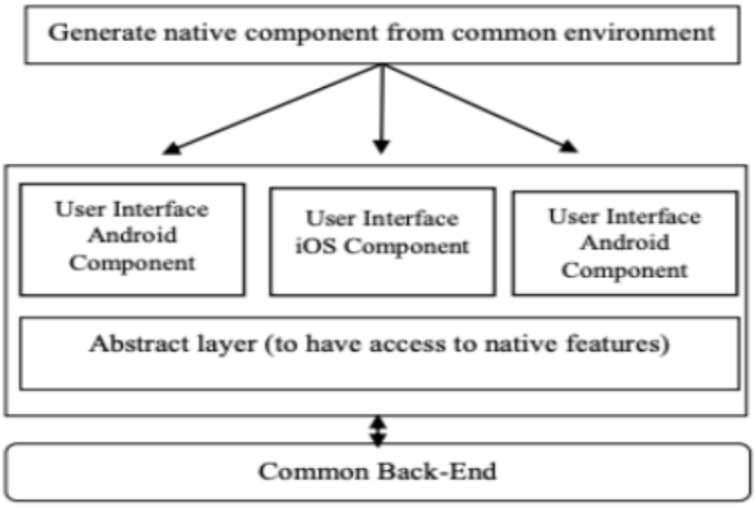
La ventaja de este enfoque es que las aplicaciones generadas logran un alto rendimiento general debido al código nativo y proporciona todas las características de la aplicación nativa, incluidas sus componentes de interfaz nativa. El principal inconveniente es que la interfaz de usuario no se puede reutilizar. Además, muchas de las funciones no se pueden reutilizar como

el método para acceder a algunas funciones, por ejemplo, acceso a la cámara; la geolocalización, etc. ya que son diferentes para cada plataforma, además este método no puede soportar la carga de grandes y sofisticadas aplicaciones. Es más adecuado para aplicaciones pequeñas.

Algunos ejemplos de frameworks para el desarrollo de aplicaciones con compilación cruzada son **Applause**, **Embarcadero Delphi XE6**, **Xamarin**, **Flutter** y **CodeNameOne**.

Aplicaciones Interpretadas o de Scripting nativo

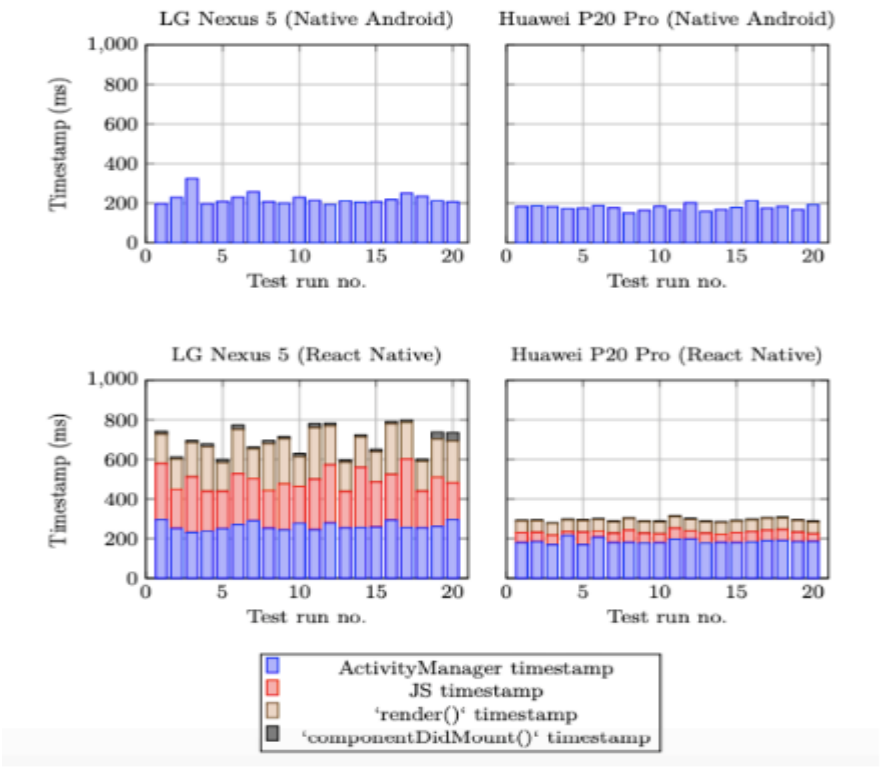
Aplicaciones interpretadas, o de scripting nativo, son aplicaciones nativas que usan un lenguaje común para escribir el código de la interfaz de usuario y generar el componente nativo equivalente para cada plataforma además de un intérprete que es empaquetado con la aplicación en el dispositivo móvil. El intérprete ejecuta el código durante el tiempo de ejecución para hacer consultas a las APIs nativas, este enfoque puede usar cualquier lenguaje de scripting que pueda ser interpretado en un dispositivo, pero la mayoría de los nuevos frameworks utilizan Javascript como su principal lenguaje.



Fuente: [Latif et al., 2016]

Las ventajas de este enfoque son similares a los frameworks de compilación cruzada: la interfaz de usuario va a verse nativa si se diseña correctamente y el rendimiento de la aplicación será muy similar al de una aplicación nativa. Sin embargo, el rendimiento sigue siendo mejor en aplicaciones nativas, además de consumir mayor memoria y energía**,** por lo que es recomendado para apps en equipos modernos con buen hardware.

https://es.linkedin.com/pulse/tecnolog%C3%ADas-actuales-para-el-desarrollo-de-m%C3%B3viles-tello-villalobos



Algunos ejemplos de frameworks utilizados para este enfoque son **NativeScript**, **React Native** y **Smartface App Studio**.

Conclusiones

Cómo se pudo apreciar, existen diversas formas de desarrollar aplicaciones móviles hoy en día: desde el enfoque hasta el lenguaje o framework a utilizar dentro de un mismo enfoque. Para cada enfoque podemos concluir:

- En el enfoque nativo: su principal ventaja es el rendimiento y la UX al mantener los componentes nativos. Sin embargo, no es recomendado para MVPs ya que implica un esfuerzo doble si se desea abarcar Android e iOS, es más recomendado para apps con un modelo de negocio establecido y que necesitan un buen rendimiento como requisito no-funcional.

- En el enfoque web: aprovecha los navegadores dentro de cualquier SO móvil por lo que (aplicando un buen diseño responsivo) es posible entregar un producto o servicio para todas las plataformas. Sin embargo, está expuesto a vulnerabilidades al ejecutarse en el navegador, mayor latencia vs una app nativa y puede tener una UX que no sea del agrado de los usuarios finales.

- En el enfoque PWA: presenta las mismas ventajas que el enfoque web y además tiene acceso a capacidades nativas, pero aún sigue estando limitado a estas.

- En el enfoque híbrido: reduce los costos y los tiempos de desarrollo al desarrollar con una sola base de código para distintas plataformas, pero al utilizar contenedores web presenta lentitud además de afectar la UX al no utilizar los componentes nativos

- En el enfoque de compilación cruzada e interpretado: presenta la misma ventaja que el enfoque híbrido además de tener un buen rendimiento muy cercano al nativo y poder utilizar capacidades nativas, pero siguen teniendo limitaciones, por lo que es recomendado para aplicaciones pequeñas destinadas a MVPs.

En conclusión, si se requiere desarrollar una aplicación móvil es esencial analizar si se necesita: abarcar usuarios de Android e iOS, validar rápidamente el concepto tras la app y si el rendimiento es un requerimiento no-funcional con gran peso. Para qué, teniendo todas estas variables definidas y con un alcance delimitado, se elija un enfoque correcto dependiendo del proyecto.

Además, es necesario sopesar los conocimientos del equipo de desarrollo, porque si bien los enfoques no-nativos ofrecen reducir los tiempos de desarrollo, existirá una curva de aprendizaje considerable con ciertos frameworks; por ejemplo, Flutter tiene su propio lenguaje (Dart) donde no muchos lo dominan hoy en día al ser "nuevo" a pesar de tener sus años existiendo. Aunque siempre será positivo que el equipo maneje frameworks/lenguajes para desarrollar apps nativas y no-nativas por las razones antes expuestas.