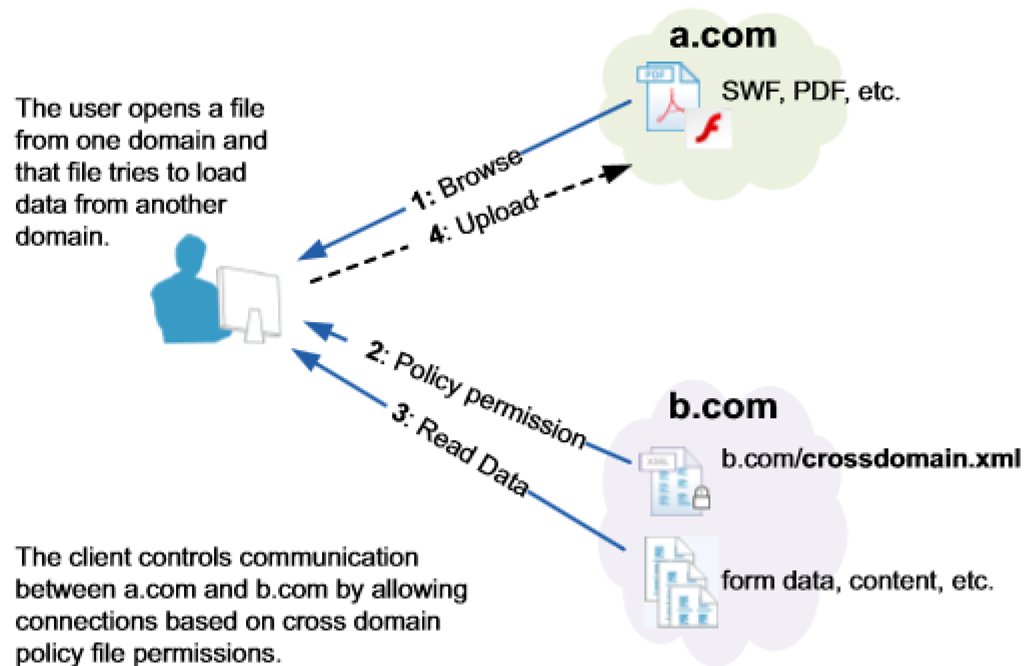
Security

CORS

Cos'è:

CORS: Cross-Origin Resource Sharing



Cos'è:

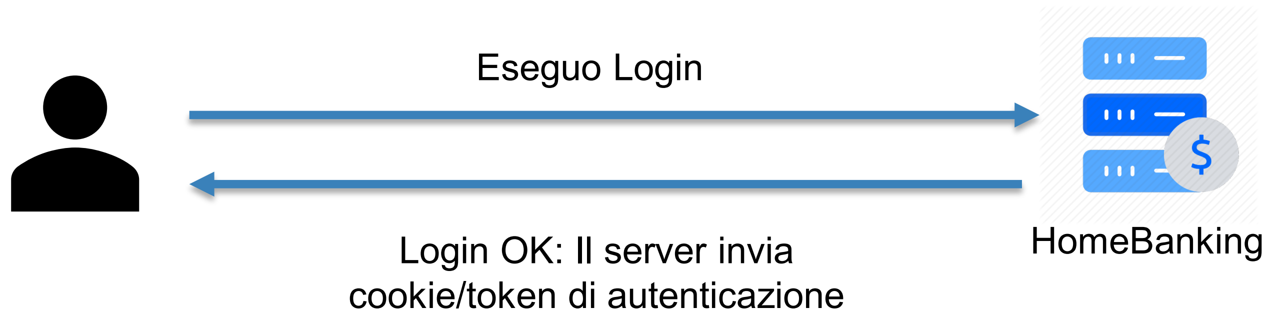
CORS: Cross-Origin Resource Sharing

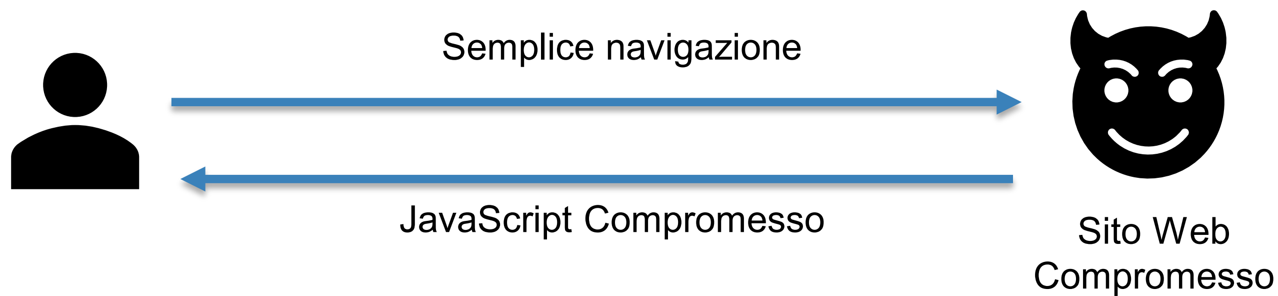
Il Cross-Origin Resource Sharing (CORS) è un meccanismo che usa header HTTP addizionali per indicare a un browser che un'applicazione Web in esecuzione su un'origine (dominio) dispone dell'autorizzazione per accedere alle risorse selezionate da un server di origine diversa. Un'applicazione web invia una **cross-origin HTTP request** quando richiede una risorsa che ha un'origine (protocollo, dominio e porta) differente dalla propria.

Esempio di cross-origin request: Il codice Javascript di frontend per un'applicazione web servita da `http://domain-a.com` utilizza `XMLHttpRequest` per inviare una richiesta a `http://api.domain-b.com/data.json`.

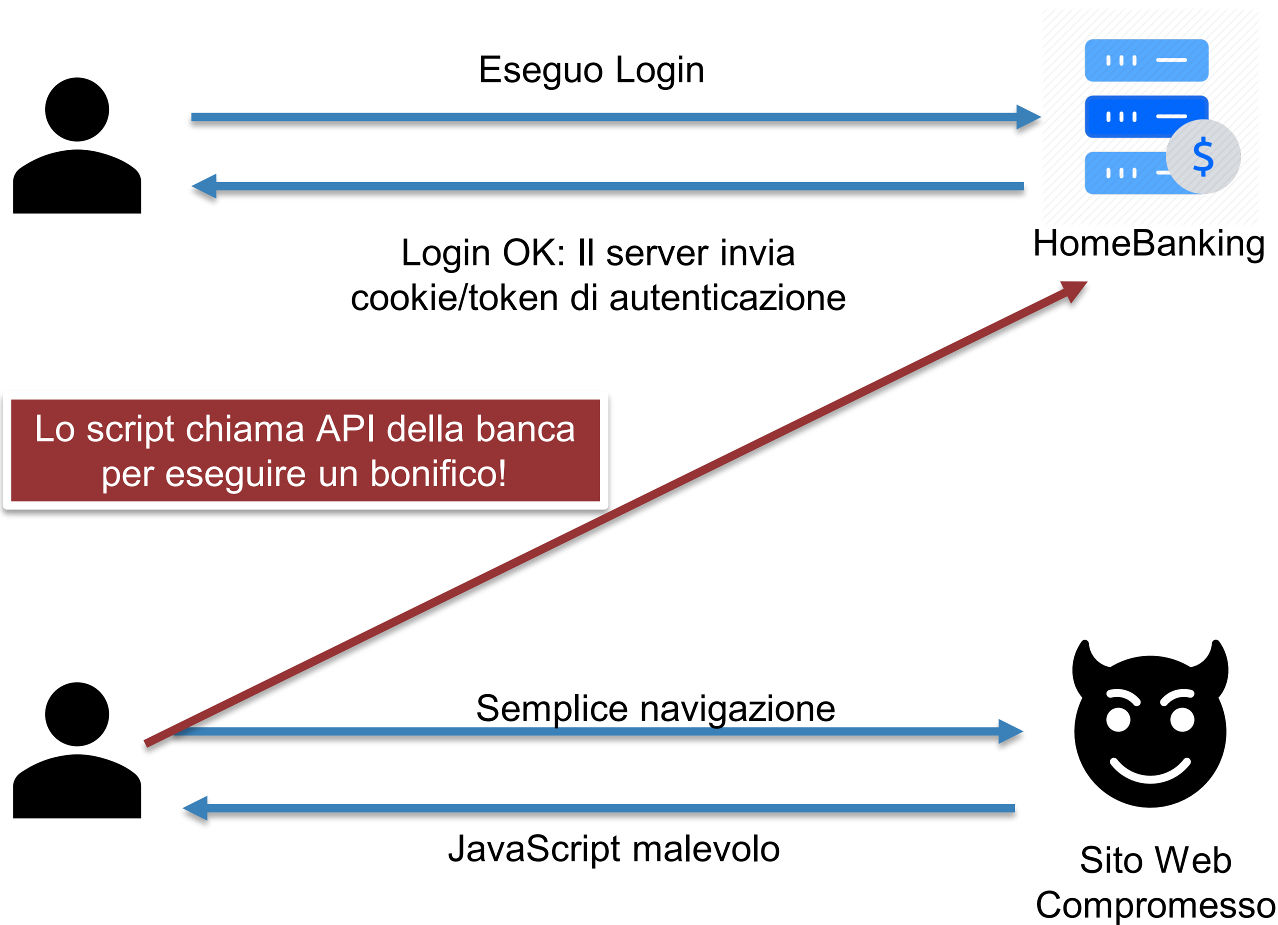
Importante: per permettere l'accesso su server di origine diversa si deve necessariamente **abilitare** e non disabilitare il CORS.

Perché è importante il CORS e cosa combatte?





Perché è importante il CORS e cosa combatte?



Tipologie

Ogni app è un mix di tecnologie differenti

Tipologie

Nativa	Ibrida	Web (PWA)
Si basa su ambienti di sviluppo e SDK proprietari della piattaforma ed il codice non risulta portabile.	Si basa su ambienti di sviluppo e SDK scelti dallo sviluppatore ed il codice risulta facilmente portabile.	Si basa su ambienti di sviluppo e SDK web ed il codice è unico.

Tipologie

Nativa

Pro:

- Performante
- Accesso all'hardware
- GUI specifica
- Presente negli store
- API subito disponibili

Contro:

- Onerosa (per ogni piattaforma ho un SDK)
- Codice non portabile

Ibrida

Pro:

- Sviluppo veloce
- Abbastanza Performante
- GUI specifica in alcuni casi
- Codice quasi portabile
- Presente negli store

Contro:

- Accesso all'hardware limitato
- API non sempre disponibile

Web (PWA)

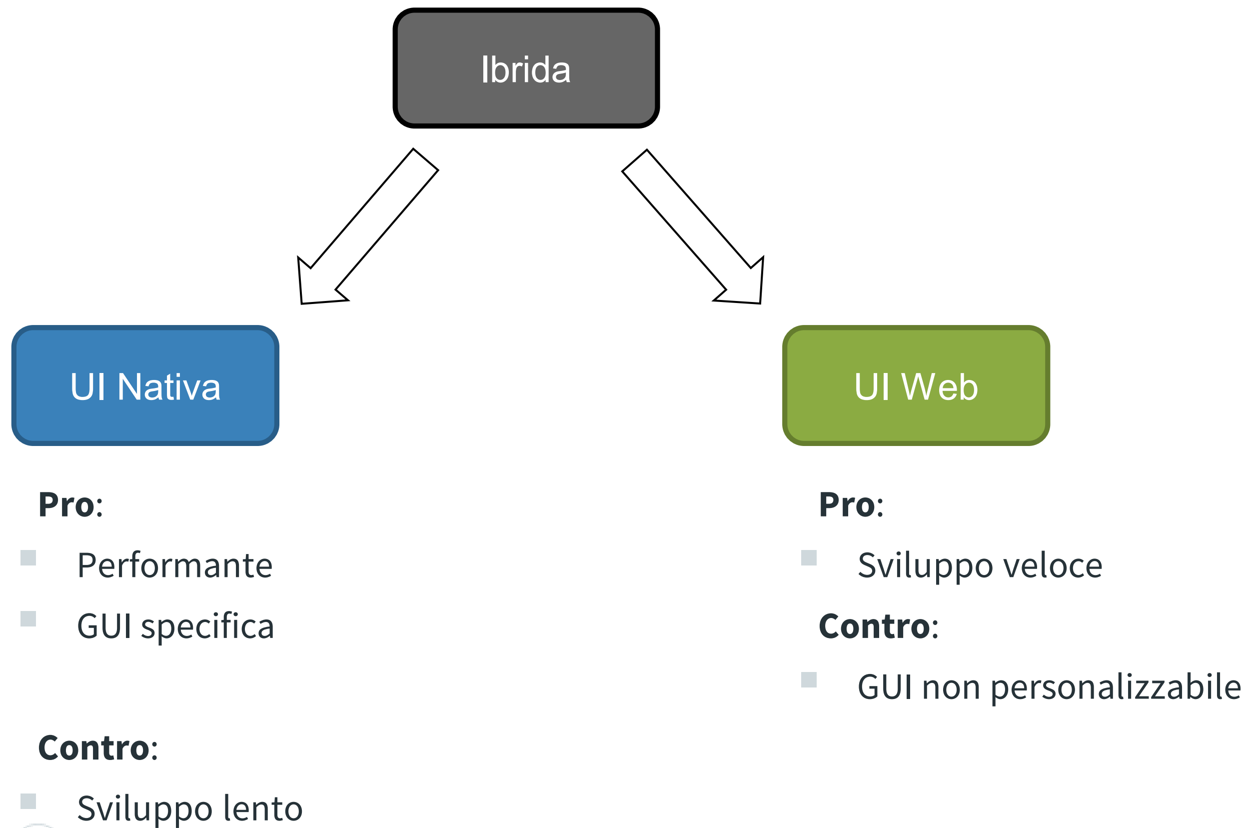
Pro:

- Sviluppo velocissimo
- Un solo codice

Contro:

- Poco performante
- GUI generica
- Accesso all'hardware limitato
- Non presente negli store

Tipologie ibride



Pro:

- Performante
- GUI specifica

Contro:

- Sviluppo lento

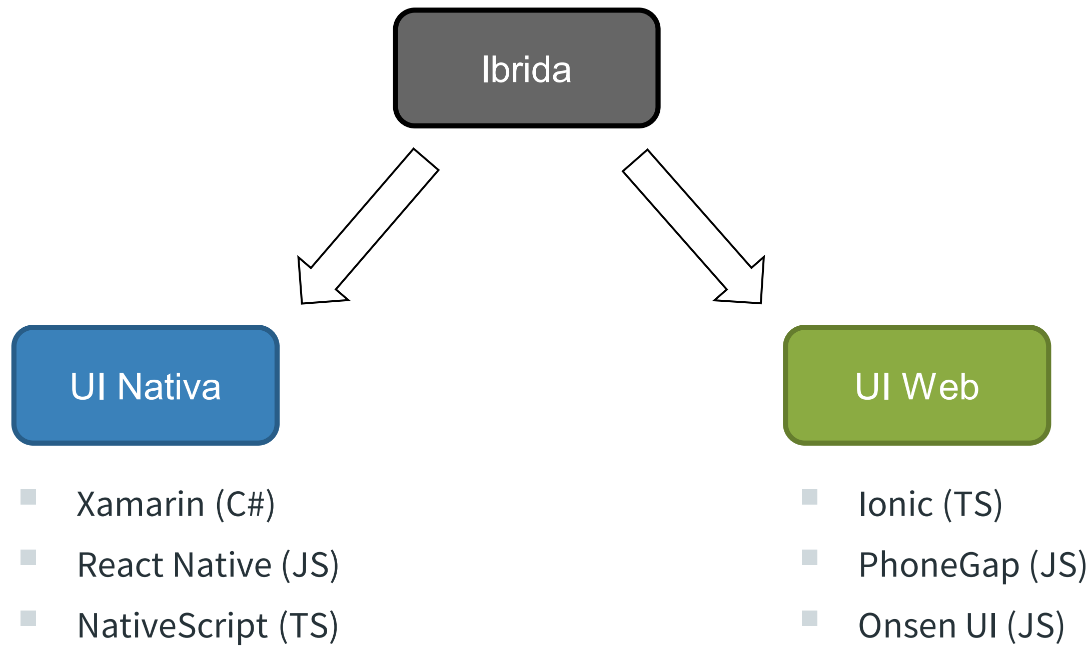
Pro:

- Sviluppo veloce

Contro:

- GUI non personalizzabile

Framework per sviluppo ibrido



Under the hood

AOT/JIT/Marshaling

Compilatore

AOT

ahead-of-time

La compilazione avviene una sola volta

JIT

just-in-time

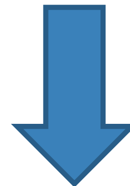
La compilazione avviene ad ogni avvio

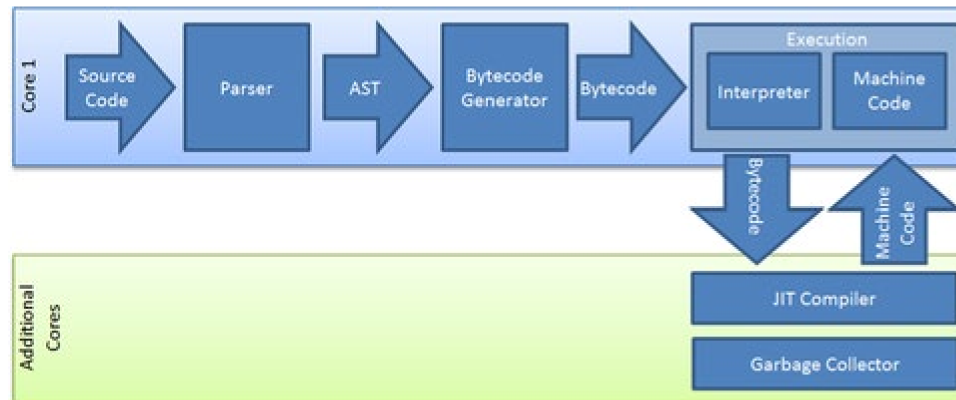
- Android permette il JIT
- iOS non permette la compilazione JIT al di fuori della WKWebView

Nativa	Ibrida	Web (PWA)
AOT	**AOT/JIT**	**JIT**

Compilatore

V8 Engine – Compilazione JIT di un JS





Ma come possiamo superare la mancanza di JIT in iOS?

No, JavaScriptCore on iOS 7+ won't be able to JIT compile for you, because iOS disallows mapping writable/executable pages of memory as a hard rule, and that's a requirement for JIT. Only MobileSafari.app, Web.app and a handful of other system apps carry an entitlement that allows them to JIT compile. The new WKWebView in iOS 8 is rendered in a separate process that is allowed to JIT compile, so JavaScript in a WKWebView is faster than a UIWebView or plain JSContext.

4

share edit flag

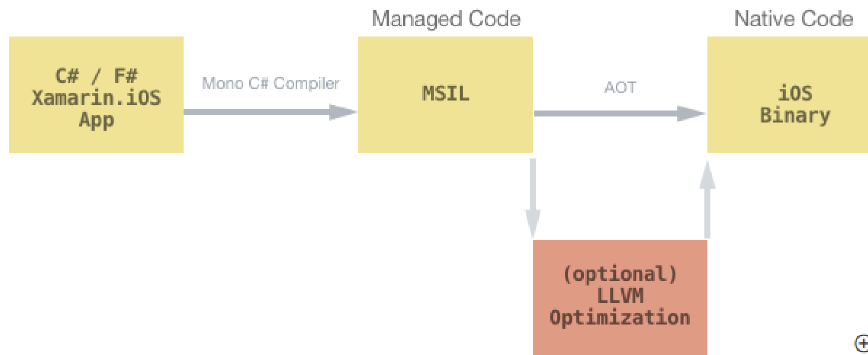
answered Jan 4 '15 at 20:52

Xamarin approach (C#)

AOT

When you compile any Xamarin platform application, the Mono C# (or F#) compiler will run and will compile your C# and F# code into Microsoft Intermediate Language (MSIL). If you are running a Xamarin.Android, a Xamarin.Mac application, or even a Xamarin.iOS application on the simulator, the .NET Common Language Runtime (CLR) compiles the MSIL using a Just in Time (JIT) compiler. At runtime this is compiled into a native code, which can run on the correct architecture for your application.

However, there is a security restriction on iOS, set by Apple, which disallows the execution of dynamically generated code on a device. To ensure that we adhere to these safety protocols, Xamarin.iOS instead uses an Ahead of Time (AOT) compiler to compile the managed code. This produces a native iOS binary, optionally optimized with LLVM for devices, that can be deployed on Apple's ARM-based processor. A rough diagram of how this fits together is illustrated below:



https://docs.microsoft.com/en-us/xamarin/ios/internals/architecture

https://docs.microsoft.com/it-it/xamarin/ios/internals/limitations

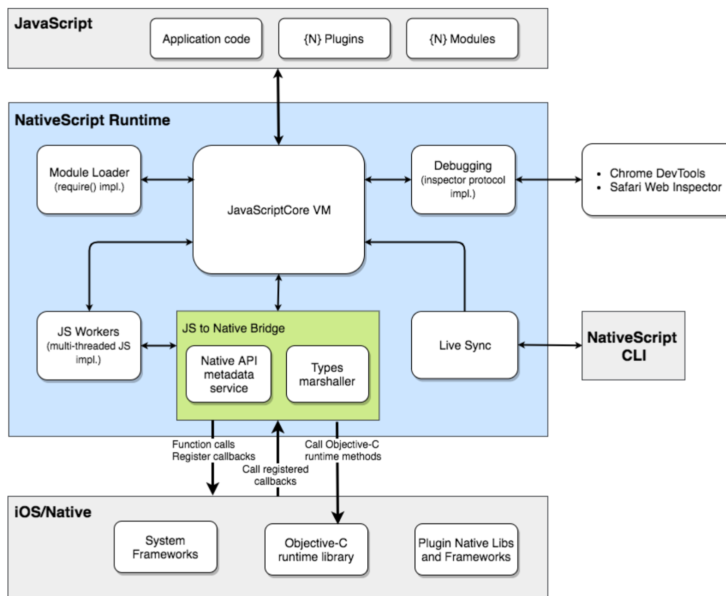
Come superare il limite di iOS ed usare JS?

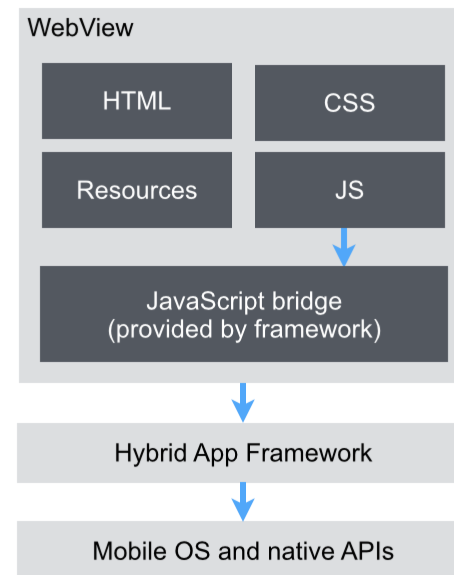
Interprete

L'applicazione viene interpretata a runtime o pre compilata

WebView

L'applicazione vive in un browser contenuto nell'app





https://docs.nativescript.org/core-concepts/android-runtime/overview
https://docs.nativescript.org/core-concepts/ios-runtime/Overview
https://www.nativescript.org/blog/the-new-ios-runtime-powered-by-v8
https://v8.dev/blog/jitless

Come superare il limite di iOS ed usare JS?

Interprete

L'applicazione viene interpretata o pre compilata

- Approccio complesso
- Performante
- UI Nativa
- Limiti nella compilazione
- Accesso hardware diretto
- Marshalling

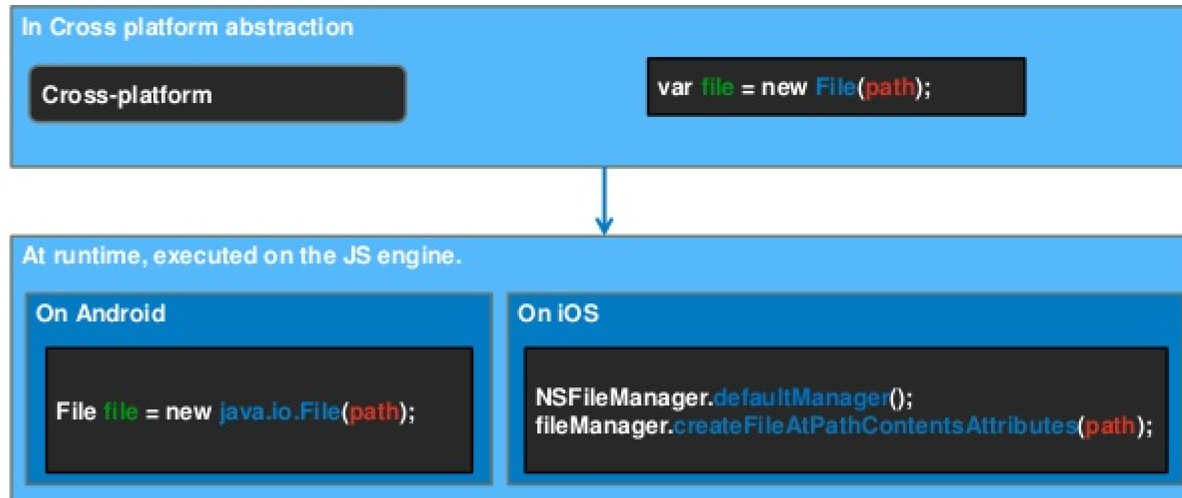
WebView

L'applicazione vive in un browser embedded

- Approccio semplice
- Lenta
- UI Web
- Nessun limite (JIT presente)
- Limiti nell'accesso hardware

Marshalling

https://docs.nativescript.org/runtimes/android/marshalling/overview
https://docs.nativescript.org/runtimes/ios/marshalling/Marshalling-Overview



Differenze sostanziali

https://www.nativescript.org/blog/nativescript-and-xamarin