



Travail d'étude et de recherche en informatique

Technologies de développement
d'application mobile multi-plateforme

-

Étudiant : Paul LEGENDRE

Encadrant : Olivier FLAUZAC

Master ASR - Université de Reims Champagne-Ardenne

1 Étude et recherche	3
1.a Introduction.....	3
1.b Applications mobiles.....	4
1.b.1 Types et caractéristiques.....	4
1.b.2 Support décisionnel.....	5
1.b.3 Application native.....	6
1.b.4 Application web progressive.....	6
1.b.5 Application hybride-native.....	7
1.b.6 Application hybride-web.....	8
1.c Étude de concept.....	9
1.c.1 Présentation des technologies.....	9
1.c.2 Frameworks concurrents.....	18
1.c.3 Concepts sous-jacents.....	19
1.d Résultats obtenus.....	25
2 Conclusion et perspectives	26

1 Étude et recherche

1.a Introduction

A l'heure actuelle, le mobile est l'écran numéro un pour se connecter à Internet en France.

75% des français en sont équipés et les applications concentrent la majorité du temps passé sur un smartphone.

La demande n'a donc cessé de croître depuis 2008, et l'évolution des technologies a entraîné un tournant important dans les techniques de développement.

La diversité des systèmes d'exploitation oblige à considérer la portabilité du code comme un enjeu majeur lors du développement d'une application mobile multi-plateforme.

Quelles technologies d'application mobile sont adaptées à votre besoin ?





Cette étude présente les concepts de développement d'une application, de la phase d'analyse jusqu'à la conception, les frameworks et les solutions permettant le support de plusieurs systèmes d'exploitation.

Du fait de l'expansion des services d'infogérance Cloud, une application ne se limite plus à une simple partie applicative, pour être efficace elle communique avec un serveur d'application également. Le choix des technologies serveurs et solutions d'hébergement Cloud sont donc également deux critères importants dans un projet de développement de ce type.

1.b Applications mobiles

1.b.1 Types et caractéristiques

De nombreux caractéristiques régissent les types d'application mobile :

Type d'applications →	Natif	Natif-Hybride	Hybride-Web	Web progressive
Portabilité & code unique / OS	0%	~ 85% - % de code natif	98%	99%
Performances IU	max	max-moyen	moyen	faible
Performances Graphiques	max	technos liées: faible/moyen/haut	moyen-faible	moyen-faible
Accès API	100%	100%	100%	0%
Offline	oui	oui	possible	non
Base site web	non	non	oui	oui
Coût	élevé	élevé	moyen	faible
Maintenance facile	conception liée	fastidieuse	facile	conception liée
Mise à jour facile	~ oui	non	oui	~ oui
Haute productivité	non	50%	90%	non
Délai de développement (équipe a nombre égal)	long	long/moyen	moyen/court	court
Temps de déploiement	moyen	long	court	court
Compatibilité	1 OS mobile	3 OS mobile	6+ OS mobile & ...	Navigateur internet
Langage	OS lié ou C++	JS/TS ou Spec + OS lié	JS/TS +HTML/CSS ...	HTML/CSS/JS ...
Rendu Serveur	non	possible	oui	oui^
Modularité	~ oui	~ oui	oui	~ oui
Exemples de technologies :	 Android SDK iOS SDK NDK/Interface	 React Native NativeScript Xamarin Flutter Kotlin mobile Appcelerator Titanium	 Cordova + beaucoup de framework web	 JQuery Mobile Mobile Angular UI Angular Symfony Django Meteor Spring Mobile

En mettant de côté l'aspect financier, les critères principaux qui orienteront votre choix dans le cadre d'une application mobile multi-plateforme seront les performance, l'unicité et la degré de portabilité du code pour chaque OS ainsi que les délais de développement.

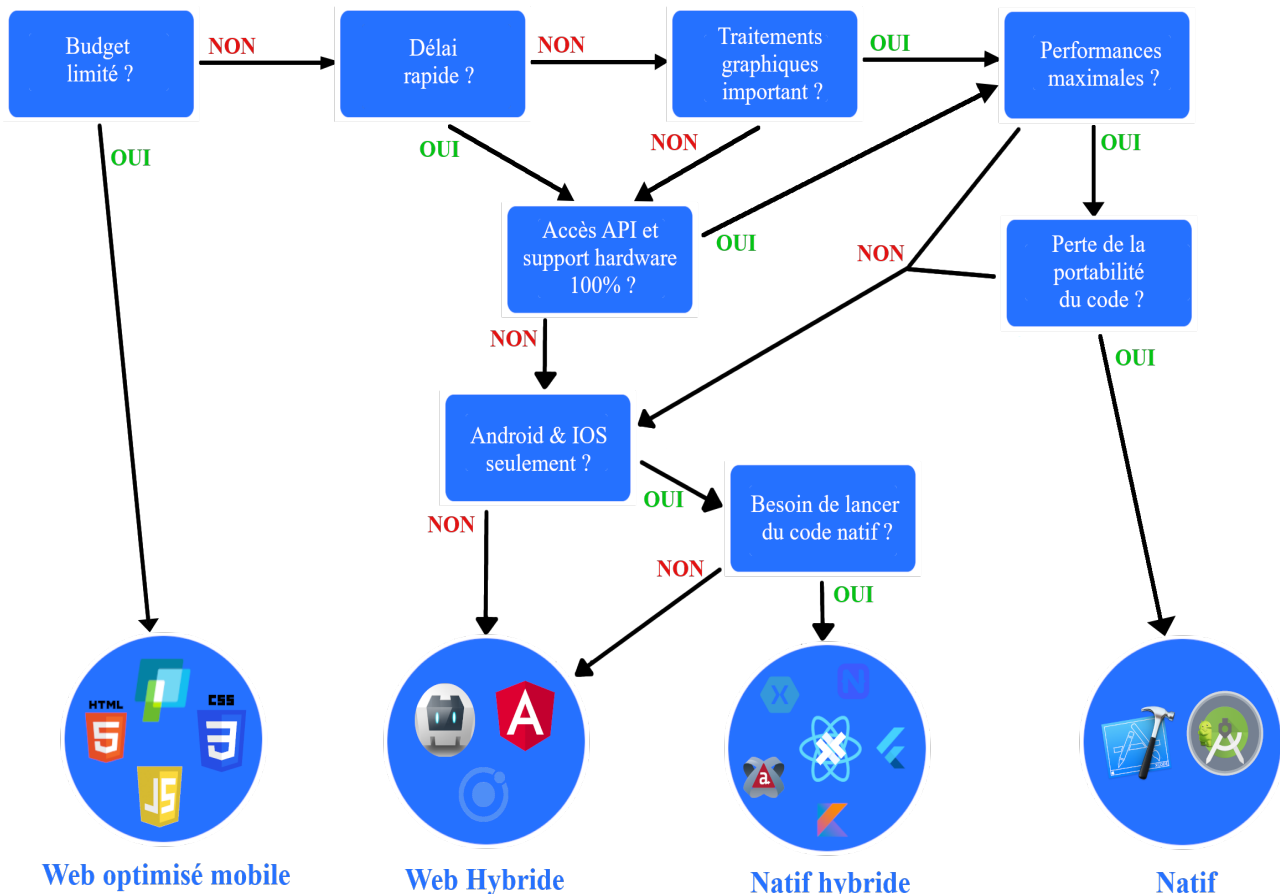
Chaque type a ses avantages et inconvénients.

Le but, les besoins et les objectifs seront également au cœur du processus décisionnel.

1.b.2 Support décisionnel

Ce support est à suivre après avoir mené les étapes préalables à l'étude de son application.

Quelle type d'application correspond à vos besoins ?



Ce support n'est pas figé et doit être considéré comme un outil décisionnel complémentaire.

1.b.3 Application native

L'application native est le meilleur choix pour les performances à tous niveaux, l'accès à l'intégralité de l'API de la plateforme cible, et par extension l'accès aux périphériques hardware.

La maîtrise du langage propre à la plateforme est requise et aucun code ne sera réutilisable pour une autre plateforme mobile. Autrement dit, il faudra coder l'application pour chaque plateforme.

Dans le cadre d'une application destinée à plusieurs plateformes, il faudra prévoir un budget important et des délais qui seront nettement plus longs comparés aux autres types d'application mobiles.

1.b.4 Application web progressive

L'application web progressive (PWA) est en faite une application web optimisée mobile au niveau de la navigation et de l'expérience visuelle de l'utilisateur.

Elle est accessible par un navigateur internet et ne s'installe pas explicitement sur le smartphone. A ce titre, elle n'est pas soumise aux mêmes contraintes (soumission sur les stores, gestion des ressources consommées sur l'appareil, etc ...).

Le multi-plateforme est assurée par la présence d'un navigateur au sein de chaque OS et son potentiel de référencement sur les moteurs de recherche est identique à celui d'un site web.

Grâce à la gestion du cache avec l'intermédiaire d'un [Service Worker](#), le contenu(après avoir été chargé une première fois) reste consultable hors connexion.

Aucun accès aux API natifs et donc hardware du téléphone n'est possible, cela implique une baisse considérable du potentiel des fonctionnalités et au final de l'expérience utilisateur.

1.b.5 Application hybride-native

L'application hybride-native utilisera du code portable pour Android et IOS et permettra l'accès à l'API natif et aux périphériques hardware du téléphone. Ces accès peuvent ne pas pas être exhaustifs.

Au besoin, elle permet également d'appeler du code natif dans le(s) langage(s) de la plateforme cible.

Une de leur force est également de ne pas subir la perte de performance de l'IU en comparaison avec les WebView (conteneur web). En effet, la plupart des technologies permettent le rendu de leurs composants graphiques en natifs à travers un bridge, une machine virtuelle ou d'autre manière.

Dans le cadre de ReactJS pour la plateforme IOS par exemple, un bridge permet l'exécution des instructions JS comme un code natif. Il fonctionne de la manière suivante :

1. Le bridge possède 2 parties développées en C++ : le JS Bridge et le Native Bridge.
1. Depuis un thread indépendant, il utilise le framework JavascriptCore permettant d'[évaluer des instructions Javascript en code natif C/C++](#) et ce de façon bidirectionnelle.
2. Il utilisera une file de message de type AMQP et des données de configuration JSON.
3. L'interfaçage entre C/C++ proposé par Swift/objectif-C assurera l'évaluation en langage natif des instructions Javascript.

Pour Android cela se passe sensiblement de la même façon, bien que ce soit JNI(Java Native interface) qui soit utilisé.

Le cross-plateforme est donc possible mais la portabilité du code ne sera intégrale, en effet cela dépendra de la part de code natif que vous développerez pour chaque plateforme.

De ce fait, la maintenance y est étroitement liée et pourrait devenir rapidement fastidieuse.

Enfin les performances pourraient être différentes selon les systèmes d'exploitation.

Par exemple, le [framework Xamarin](#) n'utilise pas du tout les mêmes techniques pour les applications Android et IOS. Au lancement de l'application sur Android, il utilisera la compilation JIT (compilation à la volée) du code IL et fonctionnera dans l'environnement d'exécution Mono en lien avec ART(Android Runtime) qui convertit ou compile le code d'un langage de haut niveau en langage machine.

Sur IOS par contre il n'y aura qu'une compilation standard à l'installation, l'application utilisera des sélecteurs et des registres afin que le C# soit exposé à l'objectif-C.

Un point important est donc que, certaines fonctionnalités peuvent être rapidement développées pour une plateforme mais pourraient demander plus de temps pour une autre.

Si le code natif nécessaire à l'application devient volumineux, il serait envisageable de se tourner vers une application mobile de type native.

1.b.6 Application hybride-web

Les applications de type web hybride regroupent donc la souplesse des applications web progressives : rapidité de développement, utilisation des langages et technologies web et présentent d'autres avantages principaux, le 1^{er} est l'unicité de votre code sur chaque OS, le 2eme est de permettre un cycle de développement rapide et productif ainsi qu'un déploiement et une maintenance aisée.

Ces applications utilisent les technologies web et interprètent leur code comme un site web le ferait avec un serveur HTTP et le moteur de rendu d'un navigateur web.

L'utilisation des conteneurs web pour afficher les vues de l'application est souvent pointée du doigt au sujet des performances. Cela est dû au temps d'affichage, de la fluidité des animations, mais également au retard de l'IU(interface utilisateur) lorsque l'utilisateur interagit avec .

Statistiquement le retard se mesure en millisecondes(ms) et historiquement était de l'ordre de 300ms.

Ce retard permettait de déterminer l'action tactile (click, double click ou encore zoom ..) sur mobile :

- l'utilisateur touche l'écran, le navigateur se met en *écoute d'événement d'action tactile*
- l'utilisateur enlève ses(son) doigt(s) de l'écran, le navigateur se met en *fin d'écoute d'action tactile*
- le navigateur attend un certain temps pour voir si l'utilisateur tape encore
- si l'utilisateur ne tape plus, le navigateur exécute l'action de l'évènement.

Les navigateurs, moteurs de rendu web et certains framework ont proposé de nettes améliorations pour ces problèmes : click fantôme, suppression du retard au zoom, meilleur gestion du tactile ...

Un autre inconvénient étroitement lié est le déclenchement des actions click. En raison d'un retard avant exécution de l'action liée au click, il est possible que l'utilisateur clique à nouveau avant la fin de l'exécution du précédent évènement. De la même façon, l'utilisateur peut appuyer involontairement sur l'écran et déclencher un click.

Ces considérations peuvent provoquer des bugs d'affichage et de traitement par exemple.

L'accès à l'API natif et à l'hardware est limité et se fait indirectement par l'intermédiaire de plugins. Cela masque les implémentations de code natif derrière une interface commune Javascript.

Leur portée, leur performance et leur fiabilité sont très restreintes.

Récemment, une technologie est apparue permettant de lancer du code natif à la demande depuis une application web-hybride devenant ainsi une application « web-hybride-native ».

Cependant l'interface utilisateur conserve toujours les mêmes contraintes liés au WebView.

1.c Étude de concept

Le choix d'un type d'application mobile conditionne très fortement les technologies, framework, librairies et langage qu'il sera possible d'utiliser.

Ces technologies ont leurs spécificités mais peuvent partager des concepts communs. Elles ont leurs avantages, leurs contraintes et leurs limites.

1.c.1 Présentation des technologies

Technologies Natives

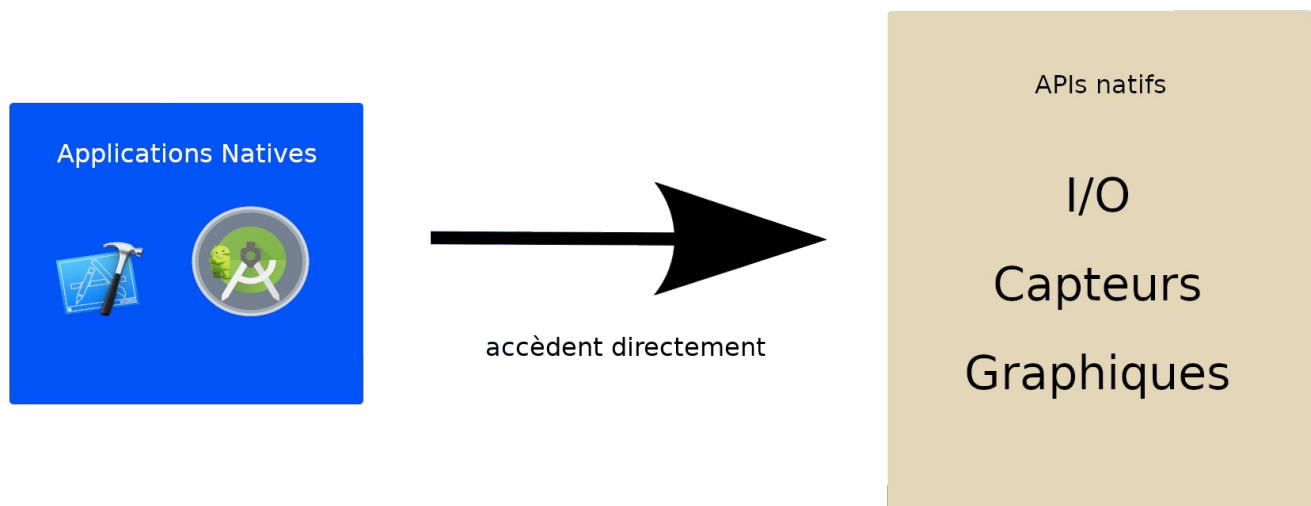
Que ce soit pour Android ou iOS, les langages natifs Java et Swift/Objective-C interagissent directement avec l'API.

Il est donc important de noter que les performances sont au maximum, aucune couche intermédiaire n'agit lors de ce processus.

On note donc une réactivité maximale de l'IU et un accès complet au matériel du smartphone.

La performance a donc un coût, celui de devoir développer un code différent pour chaque plateforme.

Les délais de développement et le budget seront au plus haut.



Technologies des applications natives :

Android : Java avec Android SDK, C++ avec Android NDK

IOS : Swift / Objectif-C avec IOS SDK

Technologies web progressive

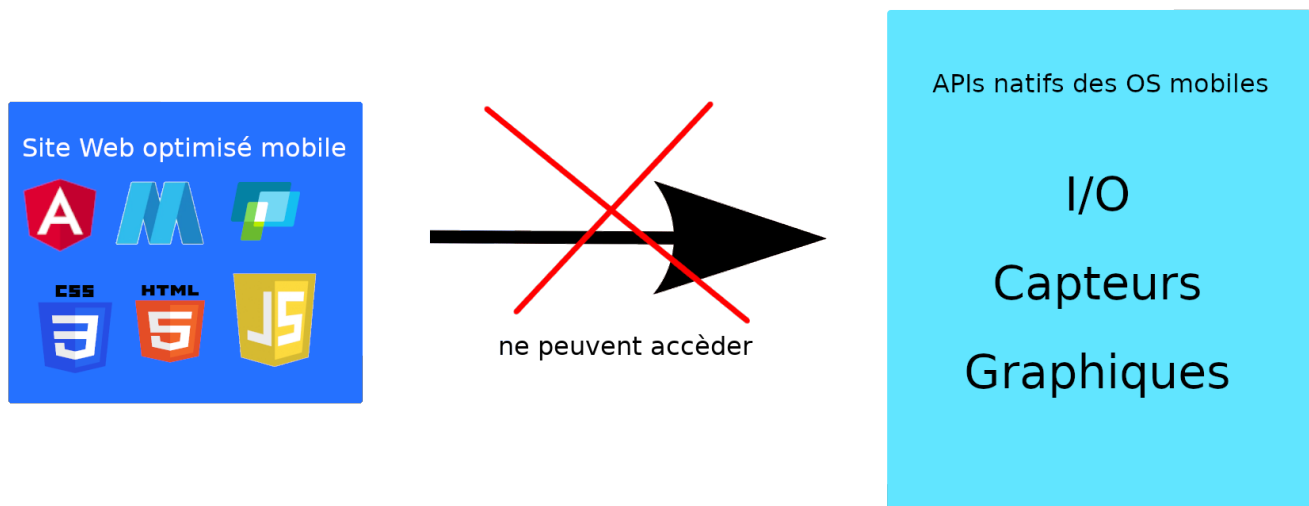
Les frameworks web permettent la prise en charge du mobile à travers un design responsive offrant une expérience de lecture et de navigation optimales pour l'utilisateur.

Historiquement, la prise en charge des résolutions d'écrans, des incompatibilités des navigateurs et autres obligeaient de s'assurer de la responsivité de chaque module dans tous les cas.

Les framework ont permis d'affranchir en grande partie le développeur de cette contrainte.

L'optimisation est faite au niveau des composants d'interfaces. Elle était rapidement devenu un des objectifs des frameworks web lors de l'apparition des mobiles et tablettes.

Rendre responsive mobile un site internet ou un module web ne lui donne pas pour autant l'accès à l'API natif de l'OS mobile, et donc par extension aux périphériques hardware du smartphone.



Les technologies liées sont les framework web populaire, JQuery Mobile, Angular Material UI et les langages de base HTML, CSS & Javascript.

D'autres framework web populaires existent également mais avec la possibilité de les allier avec d'autres outils afin d'en faire une application mobile à part entière, capable d'accéder aux API.

Ces framework couplés avec ces outils deviennent donc des technologies de développement d'application mobile hybride.

Technologies hybrides

Plusieurs frameworks web ont donc délégué ou intégré des technologies hybrides :

- Angular (Cordova, Capacitor)
- Symfony (Cordova)
- Django (React Native)
- Meteor (Cordova ou Native-script)
- Spring (React Native ou cordova)

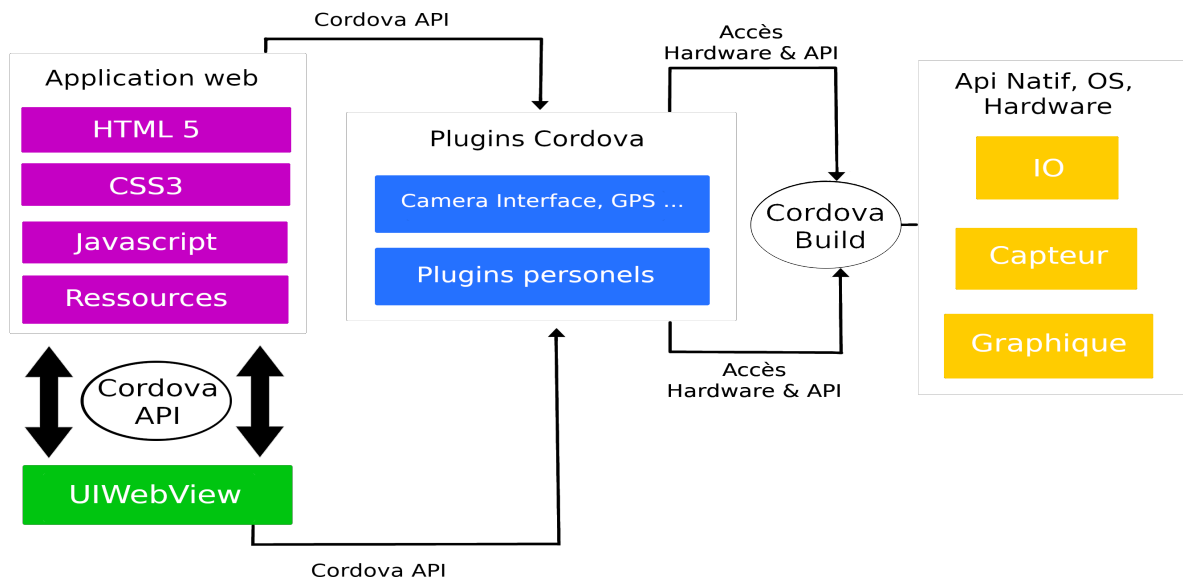
Il existe 2 types de technologies d'application hybride :

Technologies web hybrides

Les technologies web hybrides accèdent aux API natives à travers Cordova ou Capacitor. Elles utilisent un principe simple celui de charger leurs interfaces web à travers des WebView. Ce sont des composants issues de la librairies WebKit ou UIKit.

➤ Cordova

Cordova permet de manager le code JS/TS, HTML&CSS dans un container web(UIWebView) et assure l'interaction avec les API de la plateforme cible à travers des plugins.



Ionic n'est plus un framework entièrement lié à cordova. Il apporte de réels avantages à un projet l'utilisant mais Capacitor lui permet de tourner son application vers du « natif-web-hybride ».

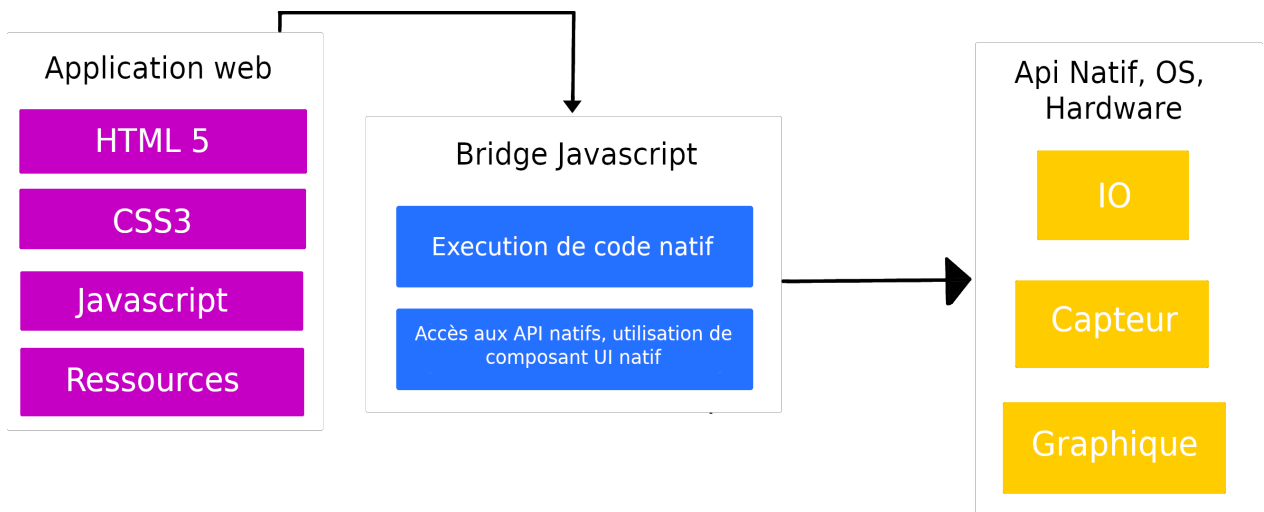
Technologies natives hybrides

➤ React Native

React Native propose une solution différente de Cordova et Capacitor. Il utilise le framework React Javascript et ses composants d'interface sont destinés à être traduits en composants d'interface natif. La compilation est plus lente et la taille de l'application générée plus importante.

Son moteur de rendu Yoga lui permettra de convertir ses vues en vues natifs de la plateforme dans un thread dédié. Son bridge Javascript a l'avantage de pouvoir lancer du code natif au besoin en fonction de la plateforme sur laquelle vous développez une fonctionnalité, d'où son type « natif-hybride ».

Sa gestion du DOM est également différente de Ionic(cf Angular) du fait qu'il réinterprétera toutes l'IU à chaque fois qu'un événement survient. De premier abord, ce concept peut être considéré comme non performant. Mais par l'intermédiaire d'un DOM virtuel et d'une technique de comparaison(Diffing) entre le DOM virtuel et le DOM actuel, il générera un différentiel. Ce dernier permettra de mettre à jour le DOM de façon performante et ceci en considérant le minimum d'opérations requises.



Le développeur n'a plus besoin de se préoccuper de l'état de son DOM, en revanche l'interface utilisateur est recalculé à chaque fois.

➤ *Flutter*

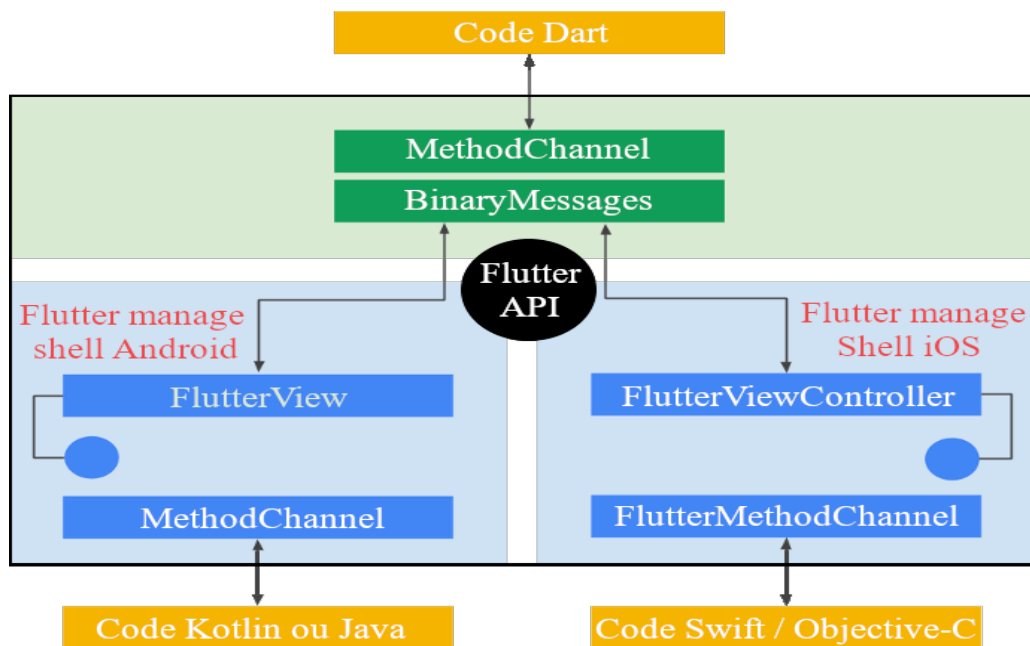
[Flutter](#) utilise le langage de programmation [Dart](#).

Il ne fait aucune concession sur les performances par plateforme, ce seront les mêmes. Il utilise ses propres widgets d'interfaces : « Material design » pour Android et « Cupertino » pour iOS. Afin de gérer l'IU il redessine au pixel près les composants natifs avec [Skia](#).

Le code [Dart](#) est compilé pour [ARM](#) en avance et permet de lancer du code natif sur IOS et Android par l'intermédiaire du « platform channel ».

Il est donc possible d'écrire du code natif en Java / Kotlin ou en Swift / Objective-C.

Flutter supporte également le fait de lancer son application sur un navigateur web.



➤ *Capacitor*

[Capacitor](#) est un API multi-plateforme avec une couche d'exécution qui permet de lancer du code natif grâce à un bridge Web.

Il possède également un API qui permet d'écrire des plugins en langage natif Kotlin/Java ou Swift/Objective-C.

C'est une partie centrale du framework Ionic depuis peu de temps.

Technologies sous-jacentes :

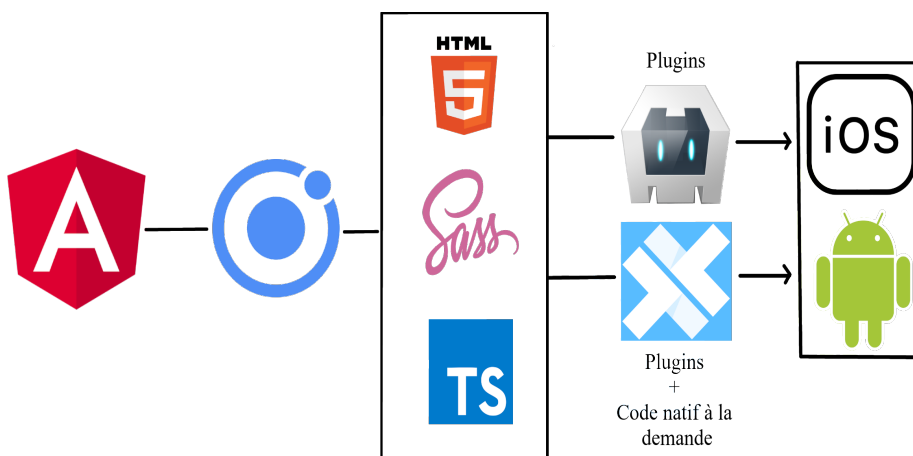
♦ [Ionic](#)

Ionic Framework est un framework open-source permettant de développer des applications web compatibles avec les fonctionnalités natives de l'OS.

Il utilise généralement HTML5, SCSS et TypeScript pour la conception des interfaces. Il utilise le framework Angular pour assurer la gestion de la logique et des données.

Il peut s'appuyer soit sur [Cordova](#) et/ou [Capacitor](#) afin de rendre compatible l'application web avec les OS mobiles comme Android et IOS et ainsi permettre la communication avec les API natives. Il utilise des web view [WKWebView](#) pour IOS.

Dernière version à ce jour : 5.0.1



♦ [SCSS](#)

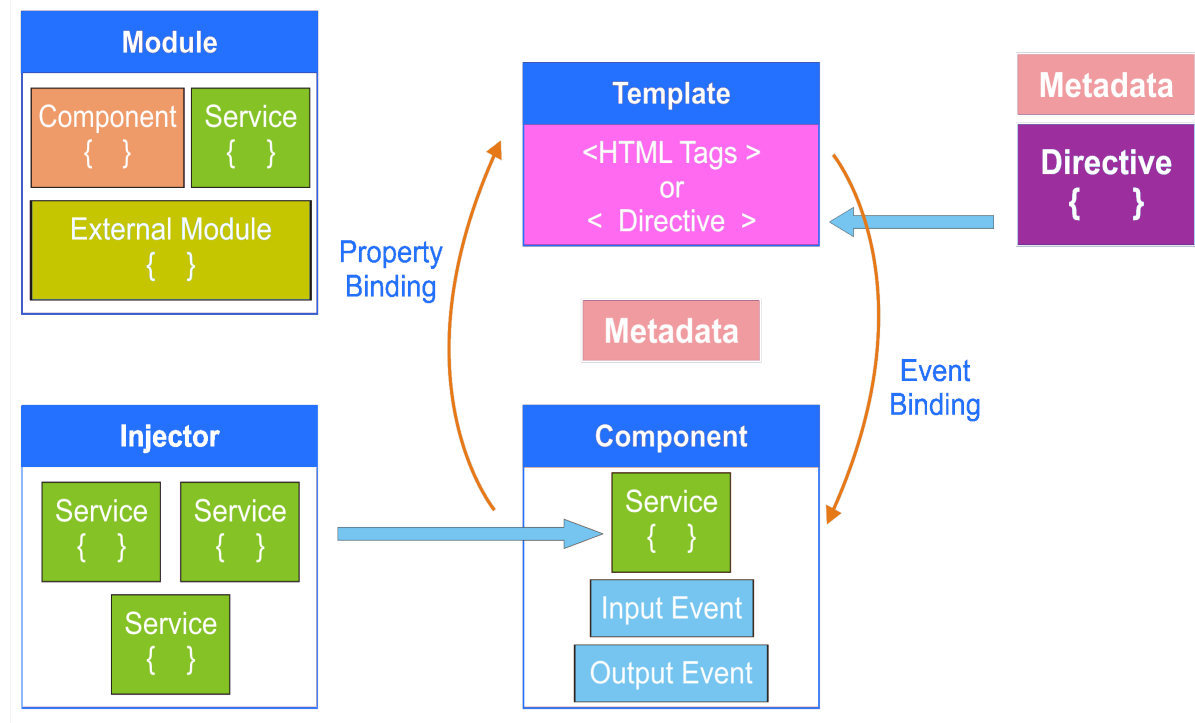
SASS (Syntactically Awesome Style Sheets) est un pré-processeur pour le langage CSS et permet de générer dynamiquement du code CSS tout en offrant une syntaxe simple et un code facilement réutilisable et maintenable. Une des syntaxes de SASS est SCSS qui a un formalisme proche de CSS.

Dernière version à ce jour : 3.5.6

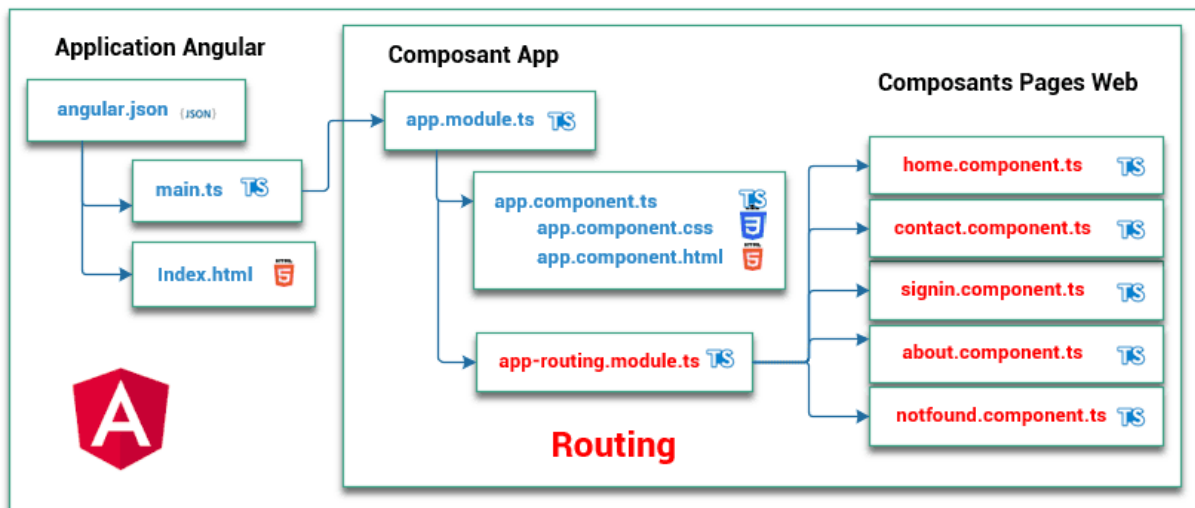
♦ [Angular](#)

Angular est un framework open source côté client permettant de développer des applications web dynamiques et immersives à page unique. Il utilise le langage TypeScript, HTML5 et CSS3.

Fonctionnement général d'une application Angular :



Décomposition d'un projet Angular :



Il est basé sur le concept de module, composants, services, templates HTML, directives, et de routes. Il utilise une compilation [AOT](#) dans la dernière version 9 ([JIT](#) Angular v8) car les composants et templates ne peuvent pas être interprétés directement par le navigateur. Cette étape convertit le code HTML et TypeScript en code Javascript efficace.

Dernière version à ce jour : 9.1.7

♦ TypeScript


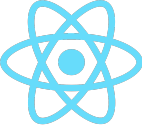



C'est un langage open source, développé comme un sur-ensemble de Javascript introduisant des fonctionnalités optionnelles comme le typage ou encore la programmation orientée objet.

Le TypeScript se transpile en Javascript, supporte les bibliothèques Javascript, partage sa documentation et apporte plusieurs avantages :

- typage du langage : il permet à la compilation de vérifier que le type de données manipulé est correct. Javascript étant un langage interprété, il permet de vérifier les erreurs avant exécution du code.
- concepts de POO et ses techniques: classes d'objets, héritage, généricité
- conversion en code Javascript efficace
- support de ES6 et rétro compatibilité avec ES5
- paramètres et type de retour explicités pour les fonctions (facultatif)
- tout code Javascript est valide en TypeScript(polyfill)

Dernière version à ce jour : 3.9.2

1.c.2 Frameworks concurrents

Framework →	 Ionic	 React native	 NativeScript	 Xamarin	 Flutter
Type application	HW ou HWN	HN	HN	NH	NH+
Accès natif	Cordova et/ou Capacitor	Bridge	Bridge	Android:ART VM IOS : OC+C# interfaçage	Dart VM
IU	WebView	JS Bridge	NativeScript Bridge	Mono Bridge	MD & Cupertino
Langage	TS/JS	JS/TS & React	JS/TS	C# & .Net	Dart
Portabilité	98% ou % code natif	% de code natif	% de code natif	70% + % de code natif	Code natif
OS supportés	Android, iOS, Ubuntu, Web et/ou Android, iOS, Web	Android & iOS, ou Web & Ubuntu	Android, iOS ou Web & Ubuntu	Android, iOS, W10	Android, iOS, Web, ~macOS
% accès à l'hardware/ API Natif	100% ou 100 %	100%	100%	100%	100%
Compilation	AOT(Angular v9) ou JIT(Angular v8)	JIT	JIT	Android : IL/ JIT IOS : AOT	JIT or AOT
Performance	moyen ou moyen+	moyen+ ou proche natif	moyen+ ou proche natif	moyen/lent ou proche natif	proche natif
Coûts	gratuit	gratuit	gratuit	gratuit + IDE haut	gratuit
Création	2013	2015	2014	2011	2017
Communauté	grand	grand	moyen-	moyen	petit
Développé par	Drifty & co	Facebook & co	Telerik	Microsoft	Google
Limites	-rendu graphique -VM Mac obligatoire	-accès natif spec = code pour chaque plateforme -taille app -VM Mac obligatoire	-accès natif spec = code pour chaque plateforme -taille app -VM Mac obligatoire	-rendu graphique -C# & .Net only -VS intégration -VM Mac obligatoire	-Dart seulement -taille app -VM Mac obligatoire
Avantages supplémentaires	-rendu serveur -1 code unique -productivité -migration WH vers NH possible	- intégration code natif	- intégration code natif	- intégration code natif	-support C- C++ avec dart:ffi - intégration code natif - IU performante

Il y a donc 2 grandes différences entre les technologies web-hybrides et natives-hybrides :

- L'affichage de la vue qui joue sur les performances et la réactivité de l'IU :
 - web-hybride : WebView
 - native-hybride : leur propre bridge ou leurs librairies graphiques
- L'accès aux API qui jouent sur les performances et l'exhaustivité des API natif supportés:
 - web-hybride : plugins ou Bridge Web
 - native-hybride : leur propre bridge ou interfaçage

Il y a également une technologie native-hybride qui se démarquent, Flutter développé par Google n'utilise pas de bridge pour son IU. Il utilise sa librairie graphique et ses composants d'interfaces MD et Cupertino pour redessiner l'interface. Son objectif est de se focaliser sur les performances. Bien qu'il utilise le langage Dart qui est un langage peu populaire, l'application permettra d'être lancée sur un navigateur web.

Enfin Ionic, dans sa version 5 avec Capacitor fait donc partie des technologies Hybride-Web-Natif.

1.c.3 Concepts sous-jacents

Les concepts importants de ces technologies sont :

Le concept asynchrone

Le langage Javascript est grandement basé sur ce concept, et donc par extension tout ce qui utilise ce langage.

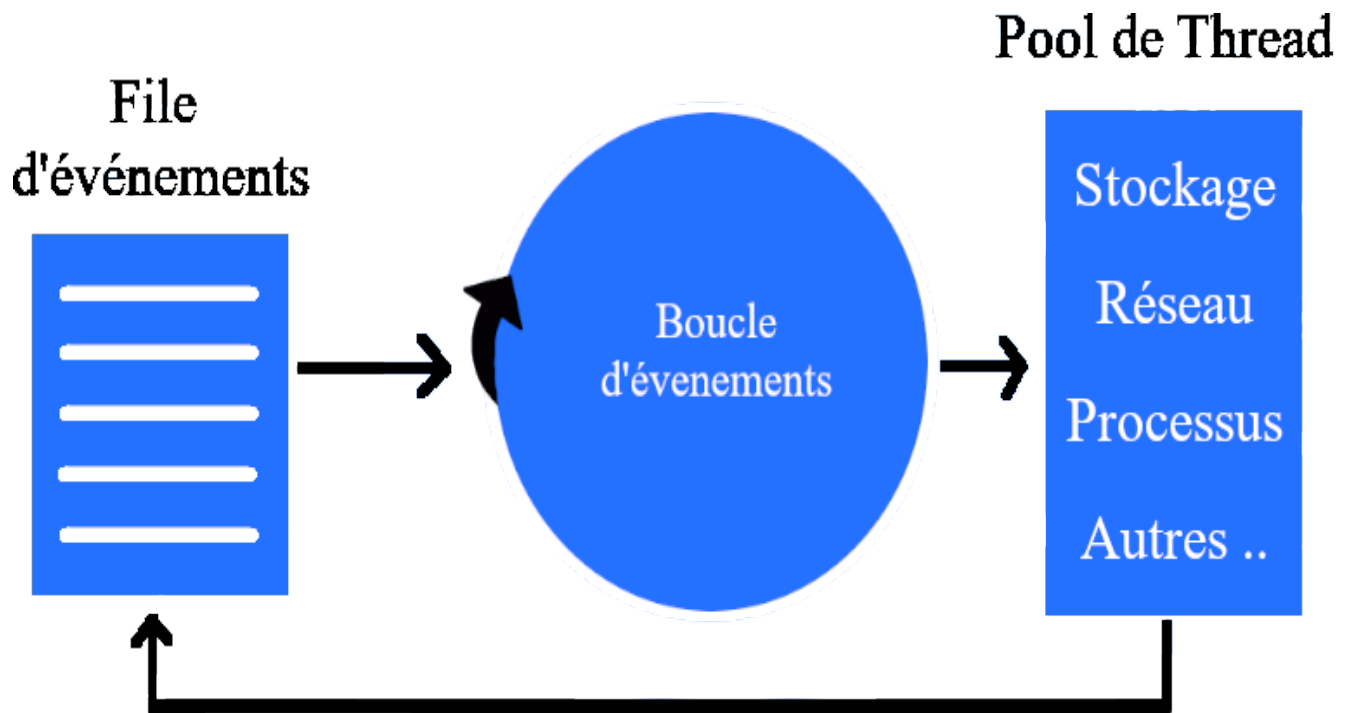
Le développement asynchrone impose au développeur de penser l'exécution de chaque tâche comme un exécution de sous tâches intermittentes.

L'asynchronisme n'exploite pas de parallélisme : le temps d'exécution de l'application peut être le même que dans le modèle synchrone.

Dans le cas où une tâche est bloquée, l'intérêt est de pouvoir exécuter une autre tâche capable de progresser. A ce titre, le modèle asynchrone est en capacité d'obtenir de meilleures performances que le modèle synchrone. Par exemple, dans le cas où il y a beaucoup de tâches indépendantes à réaliser, ainsi, quand une tâche est en attente(I/O) une autre tâche sera toujours capable de progresser dans son exécution.

Le paradigme asynchrone est donc associé à une boucle d'événements asynchrones :

NodeJS est un environnement d'exécution qui en tire fortement partie:



Programmation réactive

La programmation réactive a pour but de conserver une cohérence d'ensemble en propageant les modifications d'une source réactive (modification d'une variable, entrée utilisateur etc ...) aux éléments dépendants de cette source.

Elle s'appuie sur des « Observer » qui séparent les producteurs d'événements (Observable) des consommateurs d'événements (Observateurs).

Certains framework web et bibliothèques liés aux développement d'applications hybride mobiles les utilisent, comme RxJS dans Angular notamment.

Observable, Promise et callback

Les fonctions synchrones s'exécutent séquentiellement tandis que l'exécution des fonctions asynchrones ne garantissent pas leur ordre d'exécution.

Callback (fonction de rappel) :

Il s'agit d'une fonction A qui est passée en argument dans une autre fonction B, puis, depuis un autre endroit du programme on appelle la fonction B avec la fonction A comme argument : B(A).

La fonction A est appelée « callback » ou « fonction de rappel »

exemple 1 :

```
<!doctype html>
<meta charset="UTF-8">
<script>

function callback() {
    console.log("Je suis la fonction de rappel A");
}

function B(callback) {
    setTimeout(()=> {
        callback();
    }, 3000);
}

B(callback);

</script>
</html>
```

exemple 2 :

```
function fct_asynchrone(i, fct_callback) {
    setTimeout(()=> {
        fct_callback(i+1);
        console.log("valeur de i =" + i);
    }, 1000);
}

fct_asynchrone(1, (fct_asynchrone) => {
    console.log("la fonction de callback est exécuter");
});
```

Cependant lorsque notre programme utilise beaucoup de fonctions de rappel entre elles (chaînage), on se retrouve avec des imbrications de callback difficiles à comprendre, à maintenir et à déboguer aussi.

```
B("je suis", (chaine)=> {
  C(chaine, (chaine)=>{
    D(chaine, (chaine)=> {
      E(chaine, (chaine)=> {
        setTimeout(()=> {
          console.log(chaine);
        }, 1100);
      })
    })
  })
});
function B(chaine, myCallback) {
  console.log(chaine);
  myCallback(chaine="une imbrication");
}
function C(chaine, myCallback) {
  console.log(chaine);
  myCallback(chaine="de plusieurs");
}
function D(chaine, myCallback) {
  console.log(chaine);
  myCallback(chaine="fonctions");
}
function E(chaine, myCallback) {
  console.log(chaine);
  myCallback(chaine="de rapels");
}
```

Les promise permettent de palier ce problème.

Promise :

Une promise est faite pour réaliser un traitements asynchrone, elle retournera un objet intermédiaire qui a son tour pourra renvoyer une valeur ou rien du tout.

Une promise se crée de cette façon :

```
let ma_promesse = new Promise( (resolve, reject) => {

  let result = 1+Math.random();
  console.log("result= "+result);

  if(result < 1.5) {
    console.log("resolve va être appelé");
    resolve(true);
  }
  else {
    console.log("reject va être appelé");
    reject(false);
  }
});
```

Elles remplaceront la folle imbrication de callback précédente :

```
B("je suis une").then(C).then(D).then(E).then((chaine) => {
  console.log(chaine);
})

function B(chaine) {
  return new Promise( (resolve) => {
    resolve(chaine);
  });
}

function C(chaine) {
  return new Promise( (resolve) => {
    resolve(chaine+=" manière plus");
  });
}

function D(chaine) {
  return new Promise( (resolve) => {
    resolve(chaine+=" propre de gérer");
  });
}

function E(chaine) {
  return new Promise( (resolve) => {
    resolve(chaine+=" une imbrication de callback");
  });
}
```

- *Promise.All* permettra d'exécuter plusieurs promises indépendantes
- *Async/Await* permet de s'affranchir du *then()* des promises afin de conserver la lisibilité du code.

Observable :

Disponible sous Angular avec l'API RxJS et disponible pour beaucoup de langages avec [ReactiveX](#). Ils sont basés sur les promise et permettent donc de créer des producteurs d'évènements (Observable) et les consommateurs d'évènements (Observateurs).

A la différence des Promesses, les Observables sont des streams qui émettent plusieurs valeurs jusqu'à la fin de leur surveillance.

Observable exemple :

```
// Producteur d'événement (toutes les 2sec)
const mon_observable = Observable.interval(2000);

// Consommateur d'événement
mon_observable.subscribe( (value) => {
  console.log("value= "+value);
}, (error) => {
  console.log("erreur= "+error);
}, () => {
  console.log('observable terminé');
});

// Arrêter de consommer
mon_observable.unsubscribe();
```

Les différences entre une Promise et un Observable :

Observable	Promise
Émet des valeurs sur une période de temps	Émet une simple valeur une seule fois
N'est pas appelé tant qu'il n'est pas souscrit (paresseux, lazy)	Exécutée si elle n'est pas dans une fonction
Peut être interrompu avec la méthode unsubscribe	Ne peut pas être interrompue
Propose 100 méthodes supplémentaires : retryWhen, replay() ...	Aucun méthode supplémentaire

Les promesses peuvent cependant être utile lorsque l'on a besoin de traiter un événement une seule fois pour une seule source et qu'aucun opérateur ne nous est utile.

Rendu côté serveur (SSR)



Il est possible d'effectuer un rendu partiel ou complet de son application côté serveur.

Les avantages sont :

- gain de temps au chargement de la 1ère et toutes autres pages sur une entité à faible performance
- le temps de chargement de la première page sera très rapide.
- accès direct aux ressources de l'application (images, etc ...)
- solution efficace pour améliorer le chargement des pages si elle contiennent beaucoup de Javascript.

L'inconvénient est que l'application doit être connectée en permanence.

Différentes solutions de rendu serveur

	Server				Browser
					
	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is removed .	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side (request-response, HTML)	Built as if client-side (components, DOM*, fetch)	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML and JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. (thin client)	Delivers static HTML	Renders pages (navigation requests)	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js , Razzle , etc	Gatsby, Vuepress, etc	Most apps

1.d Résultats obtenus

L'étude et l'expérimentation de l'ensemble des technologies ont été menées sur des applications basiques, la compilation pour ces plateformes a été menée sur une machine physique pour Android et sur une machine virtuelle MAC pour iOS. De plus, le même environnement de test a toujours été utilisé « smartphone milieu gamme » et « tablette d'entrée de gamme » pour Android, sur IOS seul l'émulateur était disponible.

Les performances de compilation et du serveur applicatif, la qualité des outils de développement et de la documentation n'ont pas été prises en compte lors de l'expérimentation.

Les solutions répondant au multi-plateforme ont été évaluées selon 5 critères liés au cahier des charges : le délai, le coût, l'interopérabilité, la performance et l'évolutivité.

Productivité et interopérabilité

Le framework Ionic, de part l'utilisation des technologies web et le concept du framework Angular assure de courts délais de développement et de mise sur le marché. De plus, sa souplesse d'évolutivité permettant de passer d'une PWA à une application-hybride mais aussi à une native-web-hybride lui confère un avantage important. Le coût également est bien sur nettement plus abordable. L'unicité du code et l'interopérabilité sont au même niveau que Flutter, tandis que ses défauts se situent au niveau des performances et de la réactivité de l'interface utilisateur dû aux WebView.

Ionic est le meilleur choix pour le développement d'une application mobile multi-plateforme avec peu de traitements graphiques lourds, l'évolutivité doit être considérée également. Ainsi cette solution implique des contraintes de productivité, délai et coût prioritaires dans le cahier des charges.

Performances et réactivité

Le framework Flutter, de part sa librairie de rendu graphique skia et l'exécution du code Dart en code natif font de lui le meilleur choix pour les performances. Le code Dart supporte également la compatibilité avec le navigateur. L'unicité du code et l'interopérabilité sont au même niveau qu'Ionic tandis que son langage de programmation très peu utilisé influe directement sur la productivité, des délais de développement longs et donc un coût élevé.

Flutter est le meilleur choix pour le développement mobile multi-plateforme qui nécessite un niveau de performance générale similaire au natif. L'évolutivité ne peut pas être considérée. Ainsi, cette solution implique des contraintes de performances et de réactivité prioritaires dans le cahier des charges

2 Conclusion et perspectives

Ce travail de recherche et d'étude a permis de déterminer les technologies de développement d'application mobile multiplate-forme susceptibles de répondre au mieux au cahier des charges. Le choix du type d'application mobile, des technologies, leurs fonctionnements et concepts sous-jacents représentent des informations importantes pour le choix d'une solution efficace.

Ionic semble être le meilleur choix lorsque le budget et le délai sont les impératifs tandis que Flutter est une meilleure solution lorsque les performances de l'IU et de traitement sont prioritaires.

Les applications mobiles occupent donc une importance croissante dans la vie quotidienne, que ce soit pour les particuliers ou les entreprises. Leurs fonctionnalités permettent de servir différents intérêts selon leur publique.

Les performances de l'interface des applications web-hybrides, précisément par le rendu côté serveur des WebView, mériteraient d'être passées à un test de performance afin de savoir si la réactivité de l'interface et la consommation CPU du smartphone s'améliorent.

Au delà de ces considérations, quelle serait la configuration Hardware CPU et GPU où les types d'applications mobiles n'engageraient pas autant les ressources et l'expérience utilisateur ?

Glossaire :

Compilation JIT

Compilation « Just in time » / « compilation à la volée »

C'est une compilation qui se fait lors de l'exécution du programme

Compilation AOT

Compilation « Ahead Of Time » / « Compilation anticipée »

C'est une compilation qui traduit un langage évolué en langage machine avant l'exécution d'un programme

Mono

Version open source du .NET Framework basé sur les standards ECMA de .NET.

Il existe depuis presque aussi longtemps que le .NET Framework et s'exécute sur la plupart des plateformes. L'environnement d'exécution de Mono gère automatiquement les tâches telles que l'allocation de mémoire, le garbage collection et l'interopérabilité des plateformes sous-jacentes.

Interfaçage entre langage

Définir un interface (contrat) contenant des fonctions et des attributs afin de piloter la liaison entre 2 langages

File de messages AMQP

AMQP : Advanced Message Queuing Protocole

Orienté inter logiciel, l'objectif de la file de message est de standardiser les échanges à travers plusieurs principes (orienté message, file d'attente, routage, fiabilité, sécurité)

Transpilation

C'est un type de compilation qui prend le code source d'une langage et le compile dans une autre langage.

DOM

DOM : Document Object Model / Modèle d'objets de documents

C'est l'ensemble de l'arbre de rendu d'une entité web, une arborescence de nœuds, un interface permettant de visualiser le contenu d'un document HTML. Le CSSOM associées au style des éléments est assigné au DOM par le moteur de rendu web.

Il peut être modifier par Javascript. Il est utilisé par les navigateurs pour déterminer ce qui peut être rendu à l'écran.

Framework

Infrastructure logicielle en Français, désigne un ensemble cohérent de composants logiciels, structures qui sert à créer les fondations d'un logiciel.

Il est constitué de bibliothèque, impose un cadre de travail guidant l'architecture logicielle, induit des patrons de conceptions

SDK

Software Development Kit / Kit de Développement Logicielle est un ensemble d'outils logiciels destinées au développement facilitant le développement d'un logiciel sur une plateforme données. Il inclut un compilateur, éditeurs de liens, bibliothèque et est conçu pour un ou plusieurs langages et une ou plusieurs plateformes. I peut élégamment inclure des logiciels (éditeurs de code, débogueur, ou émulateur).

Portabilité du code

C'est la capacité du code à pouvoir être adapté plus ou moins facilement en vue de fonctionner dans différents environnements d'exécutions (hardware ou software).

Interopérabilité

Capacité que possède un système à fonctionner avec d'autre systèmes et ce sans restriction d'accès ou de mise ne œuvre.

IL

Intermediate Language / Langage Intermédiaire il s'agit d'un produit de compilation de code écrit dans des langages .NET de haut niveau.

ART

Android Runtime est un environnement d'exécution utilisé principalement par Android et vise à remplacer la VM Dalvik et à remplacer le bytecode par des instructions natives.

JNI

Java native interface / Interface Native Java

C'est une bibliothèque logicielles d'interfaçage intégré au JDK de Java qui permet au code Java s'exécutant de l'intérieure de la JVM d'appeler ou d'être appelé par des applications natives d'une autre plateforme.

API

Application Programming Interface / « Interface de Programmation d'Application ».

C'est un ensemble normalisé de classes de méthodes de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

Modularité

Ce terme provient de la programmation modulaire qui décompose une application en plusieurs modules, groupes de fonctions , méthodes ou traitements pour pouvoir les développer, les améliorer indépendamment et les réutiliser dans d'autres applications .

Services Workers :

permet à une application d'utiliser des ressources mises en cache, fournissent une expérience en mode déconnecté . Il joue le rôle d'un proxy entre l'application le réseau et le navigateurs. Il permettra de synchroniser des données entre eux.

Polyfill :

Également appelé « shim », « prothèse d'émulation ». Polyfill désigne un palliatif logiciel implémentant une rétrocompatibilité d'une fonctionnalité ajoutée à un interface de programmation dans des versions antérieures de cette interface

Généricité :

Aussi appelé « programmation générique », la généricité consiste à définir des algorithmes identiques opérant sur des données de types différents. On définit ainsi des procédures ou des types entiers génériques (surcharges, polymorphismes).

AMI :

AMI pour « Amazon Machine Images » fournit la base d'un système d'exploitation configuré selon la configuration proposé ou celle que vous configurerez. Ces images permettront de démarrer plusieurs instances EC2 partageant la même configuration.

vCPU :

vCPU pour « Virtual CPU » correspond à un CPU virtuel. Chaque instance EC2 à un nombre de vCPU qui représente le nombre de thread du processeur virtuel.

NodeJS :

Environnement d'exécution multiplate-forme Javascript basé sur la machine virtuelle V8. Le module natif HTTP permet le développement de serveur HTTP.

MongoDB :

C'est un système de gestion de base de données orientés documents qui n'a pas besoin de schéma prédéfini des données. Il fait partie des SGBD NoSQL différent des bases de données relationnelles.

NGINX :

Serveur Web ou proxy inverse basé, chaque requête est traitée par un processus dédié selon un système asynchrone.

Reverse-proxy :

Proxy-inverse, c'est un serveur proxy qui peut être placé en frontal d'une infrastructure afin de permettre aux utilisateurs d'internet d'accéder à des ressources internes

Upstream :

Dans les configurations de NGINX, cela fait référence à un serveurs ou à un groupe de serveur.

Dockerfile :

C'est un document texte contenant toutes les commandes utilisés pour la configuration d'un conteneur.

TLS :

TLS pour « Transport Layer Security », « Sécurité de la couche de transport ». Successeur du protocole SSL, il est utilisé pour sécuriser les échanges sur internet selon un mode client-serveur assurant principalement l'authentification du serveur, la confidentialité des échanges et l'intégrité des données échangées

dh_param :

DH pour « Diffie-Hellman » . C'est un fichier de paramètre utilisé et généré par la boîte à outils OpenSSL pour la configuration du protocole TLS. Il est utilisé pour convenir d'une donnée partagé sur un canal non sécurisé. Le paramètre recommandé est 2048 bits.

Logjam :

Vulnérabilité qui vise à déchiffrer les communications lors de l'utilisation d'un paramètre `dh_param` inférieure 1024 bits.

Elastic Beanstalk :

Service du cloud AWS destiné au déploiement et à l'extensibilité des ressources pour déployer des applications et services web. Il prend en charge le déploiement de conteneurs virtuelles et assure la prise en charge de la configuration et gestion de l'infrastructure.

Elastic Container Registry :

Service du cloud AWS destiné à lancé des conteneurs.

Data binding :

C'est une technique qui permet de synchroniser les données d'une source à une autre.
Ex : dans une application Angular les données récupérés des web service et celles affichées dans la vue.

CORS :

« Cross-Origin Resource Sharing », « Partage des ressources entre origines multiples ».
C'est un mécanisme qui ajoute des entêtes HTTP afin de permettre à un consommateur web d'accéder à des ressources provenant d'une autre origine que la requête initiale.
Ex : Depuis un site A, charger les images provenant d'un domaine B.

Content-type :

C'est une entête du protocole HTTP qui indique le type MIME des données échangées.

VPC :

« Virtual Private Cloud », « Cloud Privé Virtuel ».
Ce service permet de définir un réseau virtuel avec les mêmes concepts qu'un réseau traditionnelle.
Chaque ressource du Cloud AWS peut avoir son propre VPC.

Références :

[Wikipédia](#)

[Mozilla développeur](#)

[Developpez](#)

[Site officiel Angular](#)