

Cas particulier des Applications mobiles

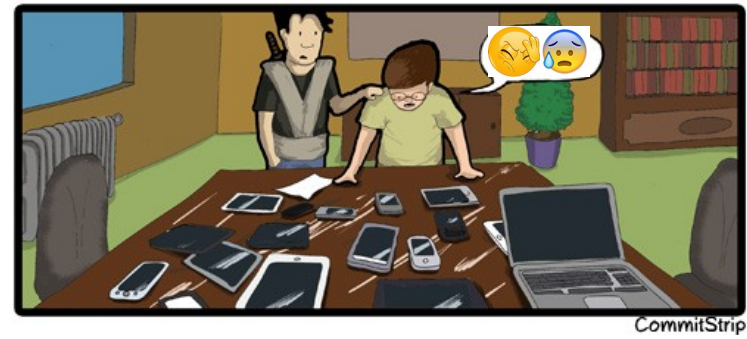
Stratégies

- Mobile-friendly / desktop-first : reactive web site, conçu pour grand écran puis s'adapte aux plus petits
- Mobile-first : Site web prévu pour être principalement accédé depuis mobile
- Mobile-only
 - Site web parfois simple redirection vers le store de l'app
 - Progressive web app
 - Application à installer



Les spécificités

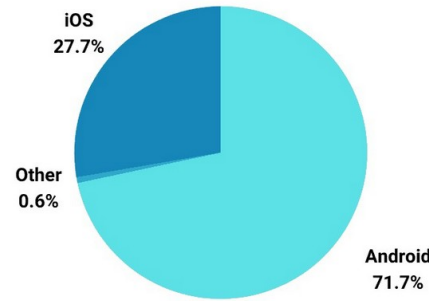
- 2 plateformes (Android, iOS)
- Souvent distribuées via un magasin d'applications
- Interfaces tactiles, périphériques spécifiques (GPS...)
- Grande variété de taille d'écran
- Connexion intermittente, à capacité variable
- Ressources matérielles plus limitées (CPU, espace disque, mémoire)
- Autonomie énergétique limitée
- Lancées et interrompues rapidement
- Appareils plus facilement égarés, utilisés avec des connexions wifi publiques



2 plateformes

- Plusieurs stratégies
 - Application native

Which Mobile OS Has the Most Users Worldwide?



- Application web
 - Simple redirection vers l'app native sur le store
 - Site web réactif
 - Progressive web app
- Développement cross-platform



Distribution via App Store et Google Play

- Publication contrôlée → délais + validations strictes
- Impact sur l'architecture :
 - Privilégier les mises à jour en passant par des APIs et des bases de données plutôt que du code en dur dans l'application
 - « feature flags » pour déployer progressivement des fonctionnalités sans mise à jour complète de l'application

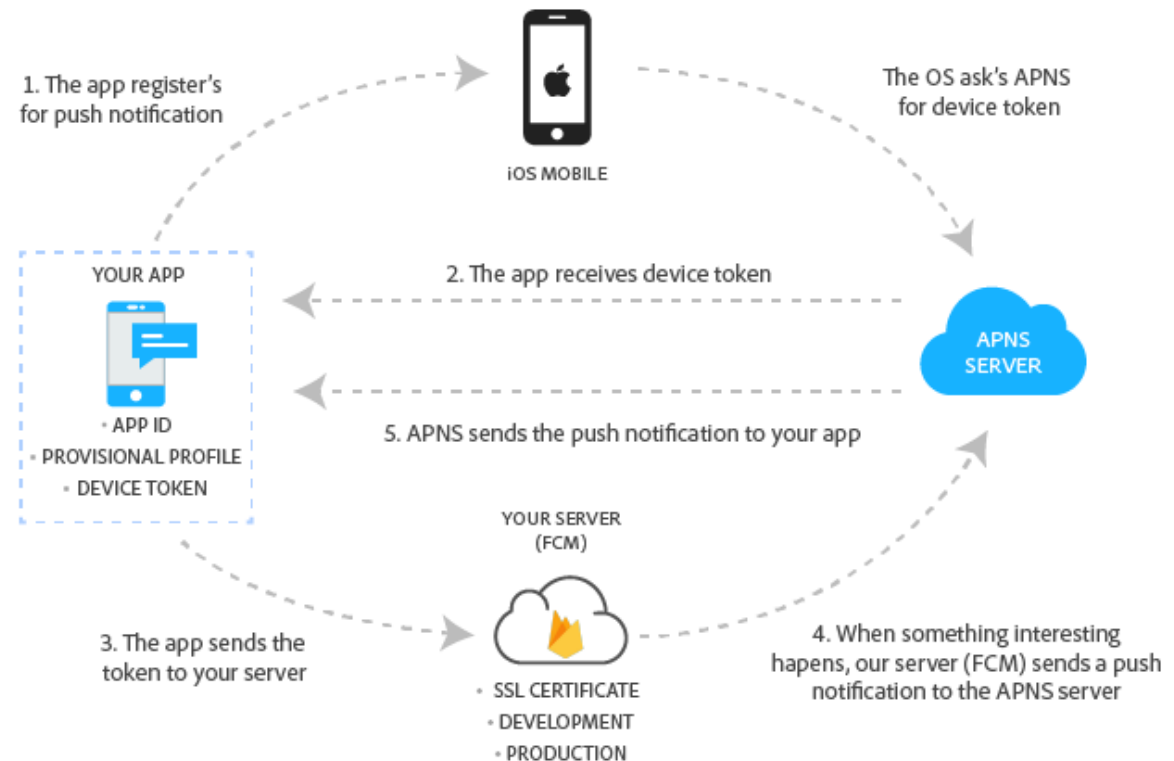
Connexion aléatoire

Architecture “sync-enabled”, conçue dès le départ pour :

- fonctionner hors ligne (offline-first ou offline-friendly)
- capturer les modifications locales (API Web Storage d’HTML5, BD embarquée)
- synchroniser ces changements avec le backend
- recevoir les mises à jour distantes, même si l’app n’est pas en cours d’utilisation (background fetch, notifications...)
- gérer les conflits de données (last-write wins, ...)

Les services *push*

- Notification / message poussé du serveur vers l'interface client (préalablement enregistré)
- Principaux services de notification
 - Google Firebase Cloud Messaging service (FCM)
 - Apple Push Notification Service (APNS) :



Ressources plus limitées

- Minimiser les requêtes réseau :
 - Pagination d'API, chargement paresseux
 - Compression des données
 - Éviter de maintenir des connexions réseau ouvertes en permanence
- Limiter l'utilisation du GPS, Bluetooth...
- Permettre un fonctionnement en mode économie d'énergie (avec désactivation des fonctionnalités non essentielles)

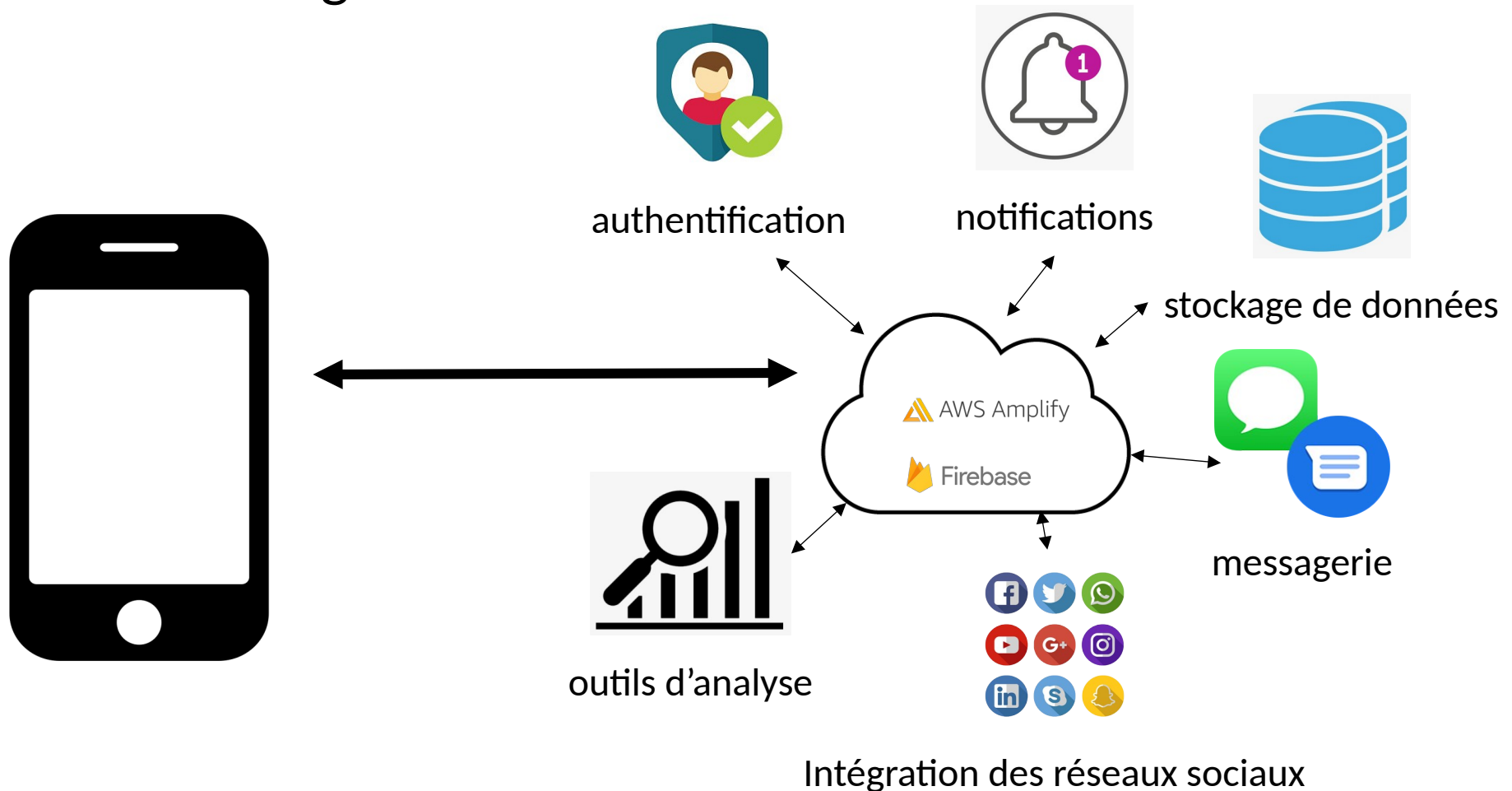
« Rotation » importante des app lancées

Doit démarrer rapidement, a une grande probabilité d'être fermée « brusquement »

- Gérer le cycle de vie de l'app (lancement, mise en arrière-plan, en veille, reprise, fermeture)
- Préserver les données d'état essentielles (session) en cas d'interruption ou fermeture
- État persistant aussi léger que possible (= reprise rapide)
- Architecture modulaire permettant le chargement sélectif

Mobile Backend as a Service

- Fonctions standard dans le nuage
- Client léger ou lourd



MBaaS

- Atouts :
 - Gain de temps, réduction des coûts
 - Passage à l'échelle « automatique »
 - Gestion simplifiée des fonctionnalités complexes (auth...)
- Inconvénients :
 - Dépendance au service
 - Non personnalisable
 - Données chez un tier

Bilan

- Nombreux principes architecturaux partagés :
 - Modulaire, pour évoluer plus facilement
 - Cloud-centric, pour faciliter la scalabilité
 - Réactivité, pour un rendu fluide
- Perspectives :
 - Tendance à intégrer des modèles IA directement sur le mobile
 - Davantage d'interactions avec des capteurs, montres, lunettes, voitures...
 - Confidentialité et souveraineté des données → architectures orientées privacy-by-design (anonymisation...) et edge-first (traitement local des données sensibles)
 - Automatisation de l'architecture par IA : bientôt des co-pilotes d'architecture, capables de proposer des patterns adaptés au contexte du projet mobile