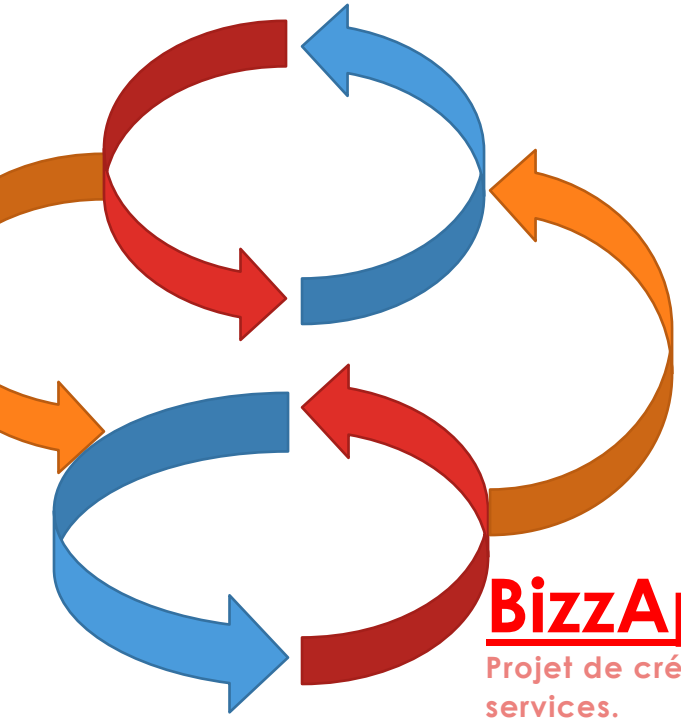
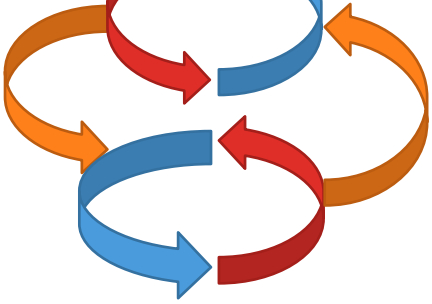


BizzApp

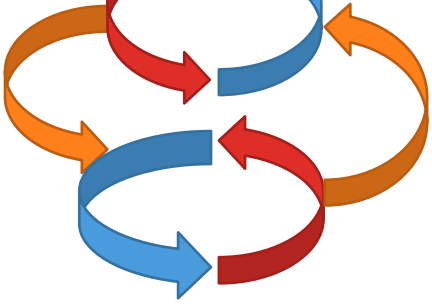
Projet de création d'une application mobile d'échange de services.

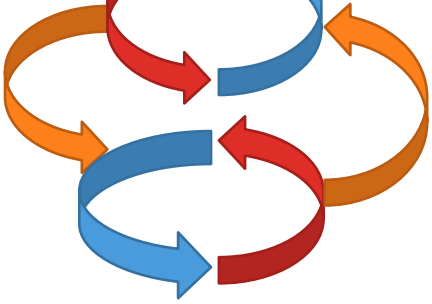
Dossier d'Architecture Technique

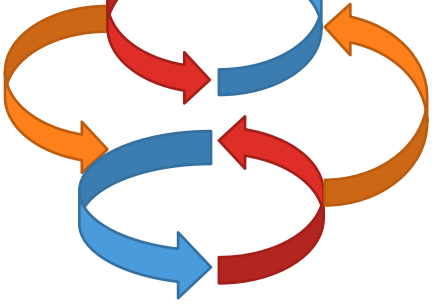
Dovénê AGOGUE

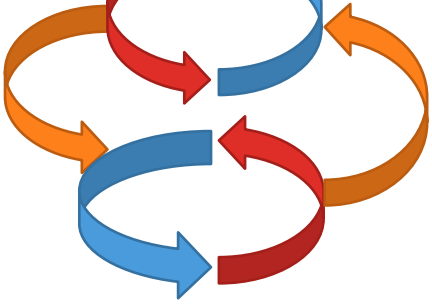


1 TABLE DES MATIERES









2 IDENTIFICATION DU DOCUMENT

2.1 HISTORIQUE DES REVISIONS

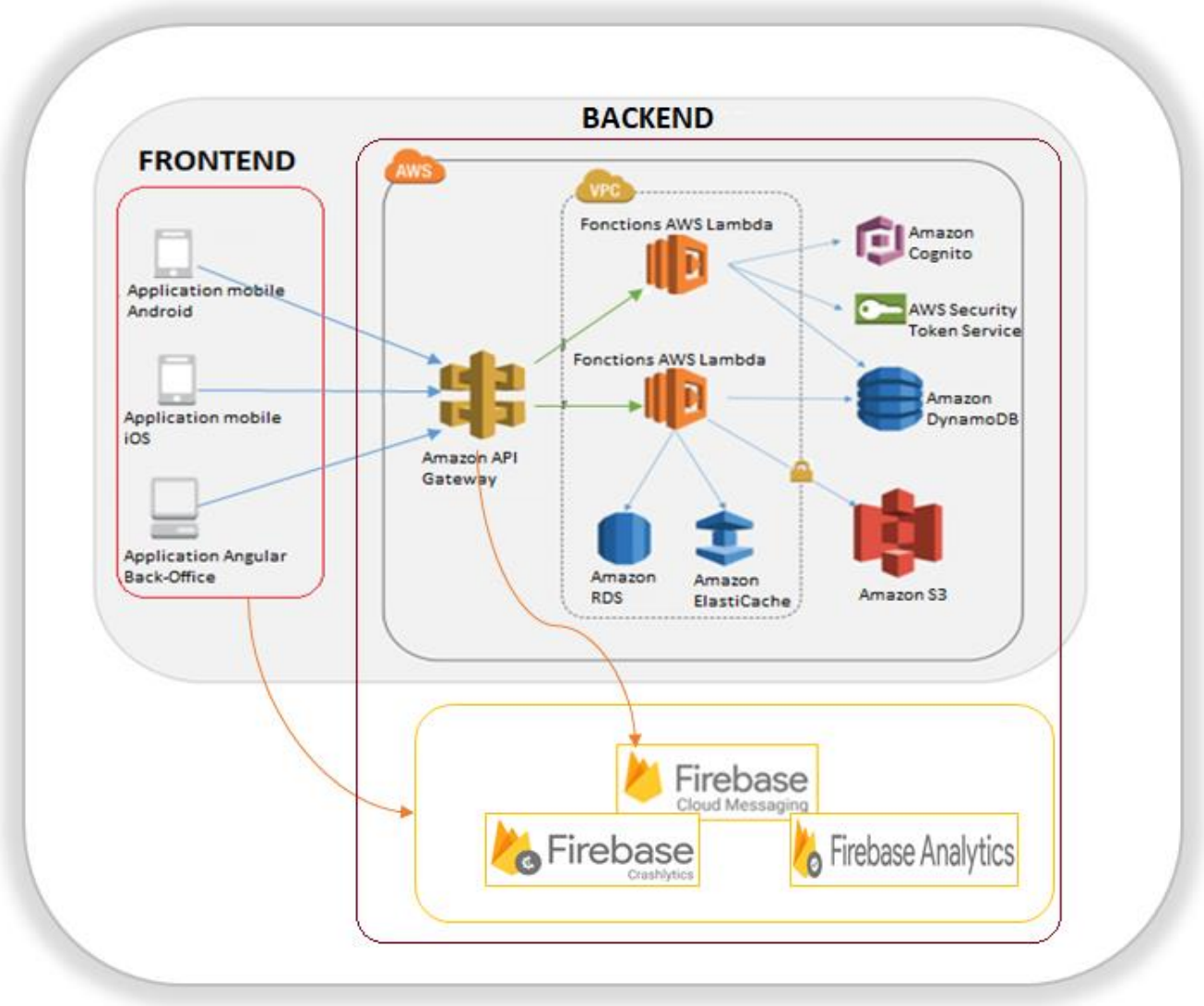
Version	Date	Auteur	Description
0.0.0	06/09/2019	Dovénê AGOGUE	Initiation

2.2 DIFFUSION ET VALIDATION

Nom	Rôle	Révision	Validation
Dovénê AGOGUE	Expert technique	Oui	Oui

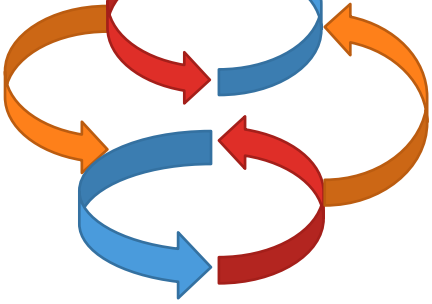
3 ARCHITECTURE GLOBALE DE LA SOLUTION

3.1 ARCHITECTURE DE LA SOLUTION



Le système à mettre en place est composé de :

- Un backend serverless entièrement basé sur le service Cloud Amazon AWS qui prend en charge à travers un cloud privé virtuel (VPC):
    - La gestion des accès utilisateurs, l'authentification et la sécurité des ressources.



    - La gestion des données liées aux processus métier.
- Un pack de composants Firebase, côté backend en charge des notifications push, des statistiques d'utilisation des applications et des rapports de crash.
- Un front end composé de :
    - Deux applications mobiles iOS et Android destinées aux utilisateurs.
    - Une application Web Angular destinée à l'administration en guise de back office.

3.2 DESCRIPTION DES COMPOSANTS

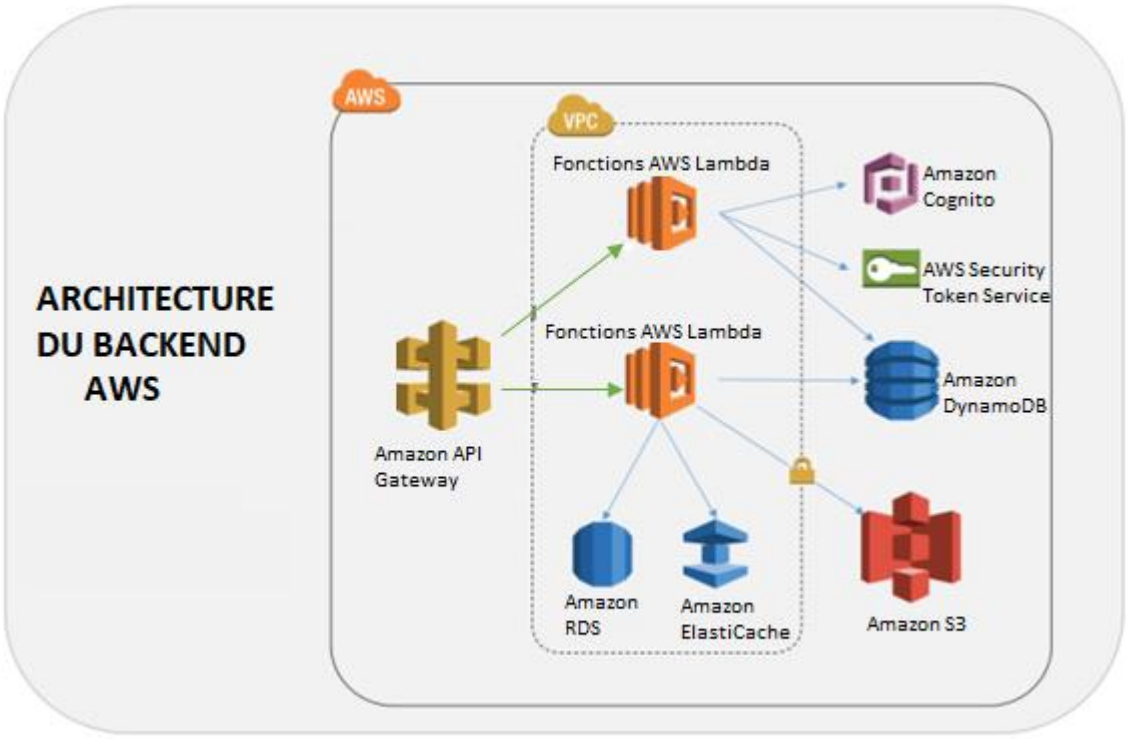
Composant	Description
Backend AWS Cloud	Bloc de composants AWS en charge de l'authentification, de la sécurité, du stockage et de la gestion des données métier.
Application mobile iOS	Application destinée aux utilisateurs d'un smartphone iOS disposant d'une connexion internet.
Application mobile Android	Application destinée aux utilisateurs d'un smartphone Android disposant d'une connexion internet.
Application web Angular	Application destinée à l'administration en back office connectée à internet.
Backend Firebase	Un pack de composants Firebase, en charge des notifications push, des analytics et des logs de crash.

3.3 FLUX DE DONNEES

Émetteur	Destinataire	Lien / Type	Sécurisation	Description
Applications mobiles (iOS/A)	Backend AWS	Double Sens	https et OAuth2, SSL Pinning, SRP.	Le backend AWS via Amazon API Gateway expose un ensemble de web services au sein d'un API Rest. Ces web services sont consommés par les applications mobiles via des appels de type GET, POST, PUT, PATCH et DELETE.
Application Angular	Backend AWS	Double Sens	https et OAuth2	Le backend AWS via Amazon API Gateway expose un ensemble de web services qui sont consommés par l'application web via des appels de type GET, PUT.
Backend Firebase	Backend AWS	Unique (AWS vers Firebase)	https et OAuth2	L'API Gateway envoie des requêtes à Firebase Cloud Messaging pour la génération de notifications push
Backend Firebase	FrontEnd (Applications mobiles et Web)	Double sens	https	Les applications mobiles et web communiquent avec Firebase Cloud Messaging pour la création de deviceId et la réception de message push puis avec Firebase Crashlytics et Analytics pour la remontée des statistiques et des crash.

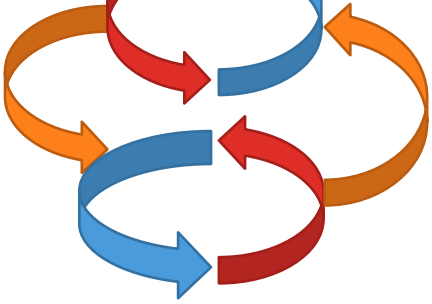
4 BACKEND

4.1 ARCHITECTURE DU BACKEND AWS



4.2 DESCRIPTION DES COMPOSANTS

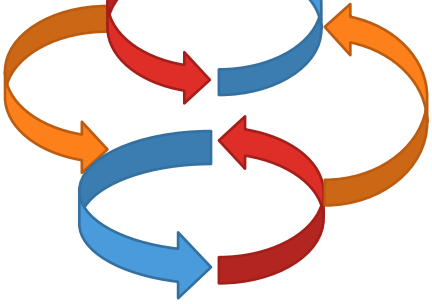
Composant	Description
Amazon API Gateway	C'est un service AWS permettant de créer, de publier, de maintenir, de surveiller et de sécuriser les API REST et WebSocket à n'importe quelle échelle. API Gateway crée des API REST qui : • reposent sur le protocole HTTP ; • respectent le protocole REST, ce qui permet la communication client/serveur sans état ; • mettent en œuvre les méthodes HTTP standard, telles que GET, POST, PUT, PATCH et DELETE.
Fonctions AWS Lambda	AWS Lambda est un service informatique qui permet d'exécuter du code sans nécessiter la mise en service ni la gestion de serveurs et une fonction AWS Lambda est une ressource que vous pouvez appeler pour exécuter votre code dans AWS Lambda.



	La fonction contient un code qui traite les événements et un environnement d'exécution qui transmet les demandes et les réponses entre Lambda et le code de la fonction. Vous fournissez le code et vous pouvez utiliser les environnements d'exécution ou créer les vôtres.
Amazon RDS	Amazon Relational Database Service (Amazon RDS) permet d'installer, de gérer et de mettre à l'échelle facilement une base de données relationnelle dans le cloud. Amazon RDS donne le choix entre six moteurs de base de données, notamment Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database et SQL Server.
Amazon ElastiCache	Amazon ElastiCache est un service de cache et de magasin de données en mémoire entièrement géré par Amazon Web Services.
Amazon Cognito	Amazon Cognito permet d'ajouter facilement et rapidement l'authentification aux différentes ressources du projet en permettant de gérer : • L'inscription des utilisateurs • La connexion des utilisateurs • Le contrôle des accès aux applications mobiles et web.
AWS Security Token Service	C'est un web service qui permet d'obtenir des informations temporaires et des droits limités afin de se connecter et d'accéder aux ressources AWS.
Amazon DynamoDB	Amazon DynamoDB est un service de base de données NoSQL rapide et souple, adapté à toutes les échelle. Il est composé de clés-valeurs et de documents, offrant des performances de latence de l'ordre de quelques millisecondes, quelle que soit l'échelle.
Amazon S3	Amazon Simple Storage Service (Amazon S3) est un service de stockage d'objet offrant une scalabilité, une disponibilité des données, une sécurité et des performances de pointe.

4.3 FLUX DE DONNEES

Émetteur	Destinataire	Lien / Type	Sécurisation	Description
Amazon API Gateway	Fonctions Lambda AWS	Double Sens	AWS Cognito/STS	API Gateway communique avec tous les autres services de AWS via des fonctions lambda.
Fonctions Lambda AWS	Amazon RDS	Double Sens	AWS Cognito/STS	Les fonctions lambda communique avec Amazon RDS pour faire du CRUD sur les bases de données relationnelles.
Fonctions Lambda AWS	Amazon ElastiCache	Double Sens	AWS Cognito/STS	Les fonctions lambda communique avec AmazonElastiCache pour le traitement des données en mémoire cache.
Fonctions Lambda AWS	Amazon Cognito	Double Sens	AWS Cognito/STS	Les fonctions lambda communique avec Amazon Cognito pour tout traitement relatif à l'authentification des utilisateurs sur AWS.
Fonctions Lambda AWS	AWS Security Token Service	Double Sens	AWS Cognito/STS	Les fonctions lambda communique avec AWS STS pour tout traitement relatif aux accès temporaires et droits limités.
Fonctions Lambda AWS	Amazon S3	Double Sens	AWS Cognito/STS	Les fonctions lambda communique avec Amazon S3 pour l'accès aux données stockées sur Amazon Cloud.



4.4 Stack technique & technologies utilisees

- NodeJS
- Express
- Visual Studio Code

4.5 Modele de donnees

Les données métier sont gérés par Amazon RDS et Amazon DynamoDB.

4.6 Securisation

AWS Cognito et AWS Security Token se chargent de la sécurisation des différentes ressources Backend.

4.7 Authentification

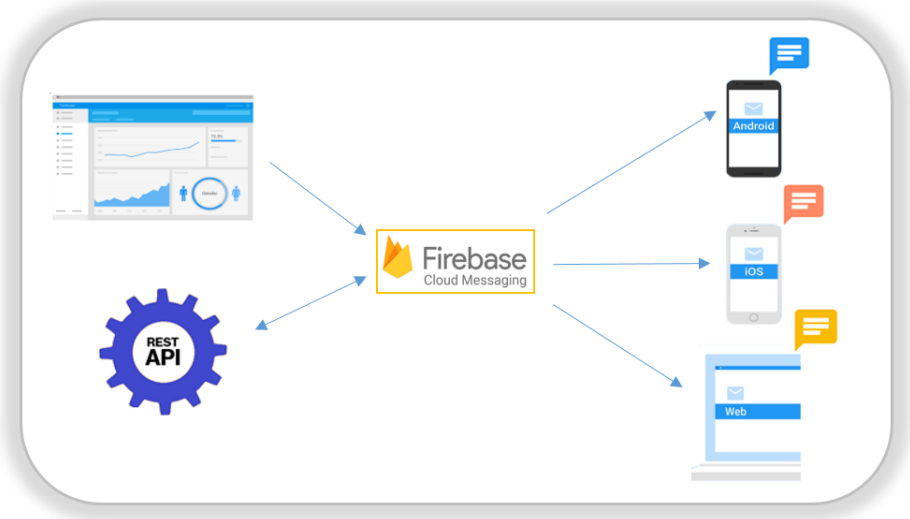
AWS Cognito implémente oAuth2 et prend en charge l'authentification des utilisateurs.

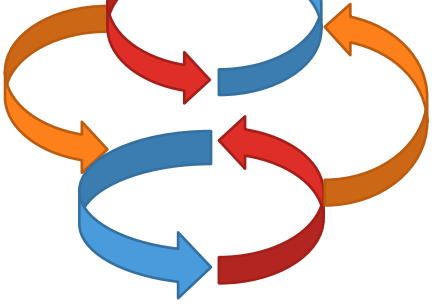
4.8 API REST (services web)

Les web services sont implémentés au sein de AWS API Gateway et permettent d'effectuer les différentes actions CREATE, PUT, PATCH, POST et DELETE.

4.9 Systeme de Push notifications

Les notifications push seront gérés par Firebase Cloud Messaging qui est la solution de Google permettant d'envoyer des notifications aux applications mobiles Android et iOS puis Web (Javascript).





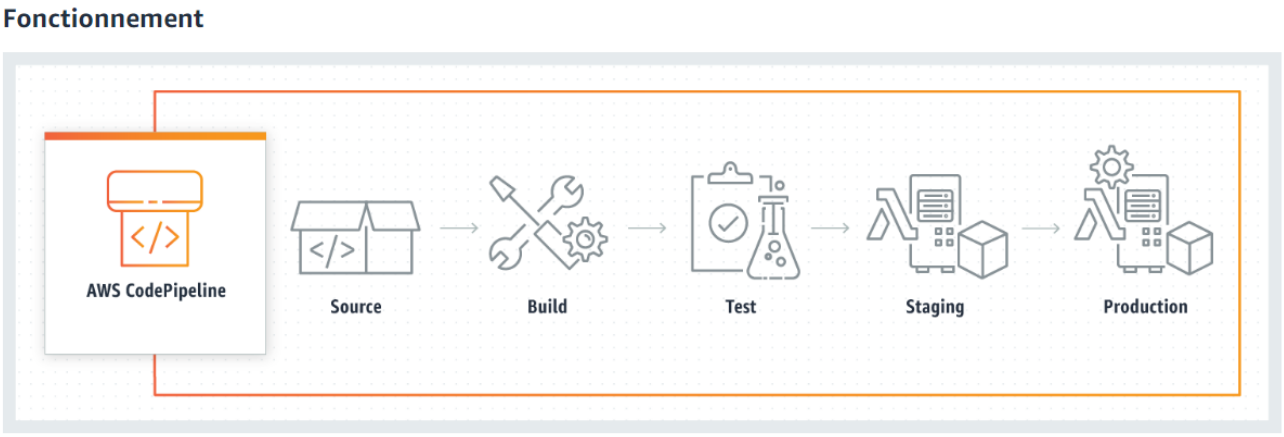
4.10 Guidelines / normes de developpement

- Tests Driven Development (TDD)
- Développement des tests unitaires et des tests End to End.
- Automatisation des tests.
- Intégration et Déploiement Continu.
- Dépôt des sources en continu sur le système de contrôle de sources.
- Documenter le code.
- Ne pas optimisez prématurément.
- Pensez à industrialiser les fonctionnalités récurrentes.

4.11 DevOps process

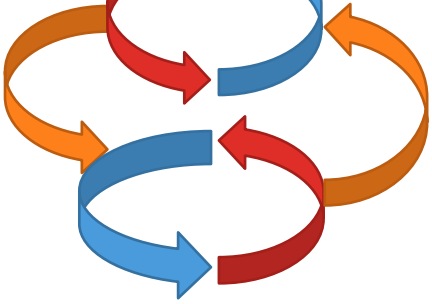
Pour faire de l'intégration et du déploiement continu nous utiliserons AWS CodePipeline qui est un service de diffusion continue entièrement géré, qui permet d'automatiser les pipelines de diffusion afin d'obtenir un déploiement rapide et fiable de mises à jour d'applications et d'infrastructures.

CodePipeline automatise les phases de développement, de test et de déploiement des processus de diffusion à chaque fois qu'un changement de code a lieu, en fonction du modèle de diffusion que vous avez défini.



4.12 Hebergement / Hosting

L'hébergement est centralisé chez Amazon Cloud.

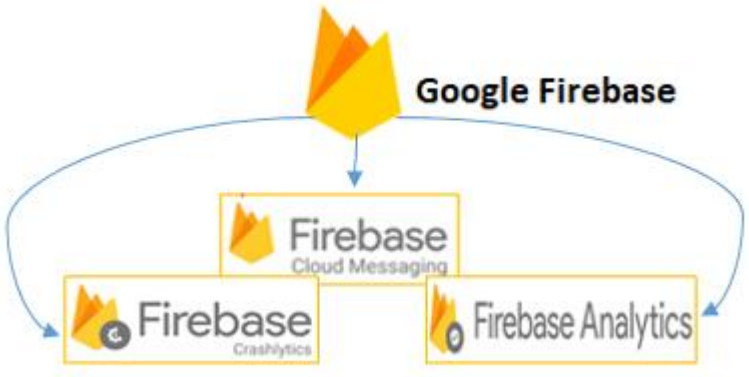


4.13 GESTION DE CONFIGURATIONS

AWS CodeCommit est le service de contrôle de source de AWS Cloud qui héberge des référentiels Git sécurisés. Il sera donc utilisé pour suivre les changements.

4.14 ARCHITECTURE DU BACKEND FIREBASE

4.14.1 SCHEMA DE L'ARCHITECTURE

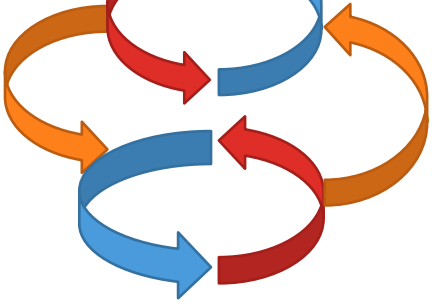


4.14.2 Description des composants

Composant	Description
Google Firebase Cloud Messaging	C'est un composant du service Google Firebase en charge des push notifications pour les applications mobiles iOS et Android puis les applications web basée sur du Javascript.
Google Firebase CrashLytics	C'est un composant du service Google Firebase en charge des rapports de crash pour des applications mobiles iOS et Android puis les applications web basée sur du Javascript.
Google Firebase Analytics	C'est un composant du service Google Firebase en charge des analytics (rapports multicritères) sur l'utitilisation des applications mobiles iOS et Android puis les applications web basée sur du Javascript.

4.14.3 Flux de données

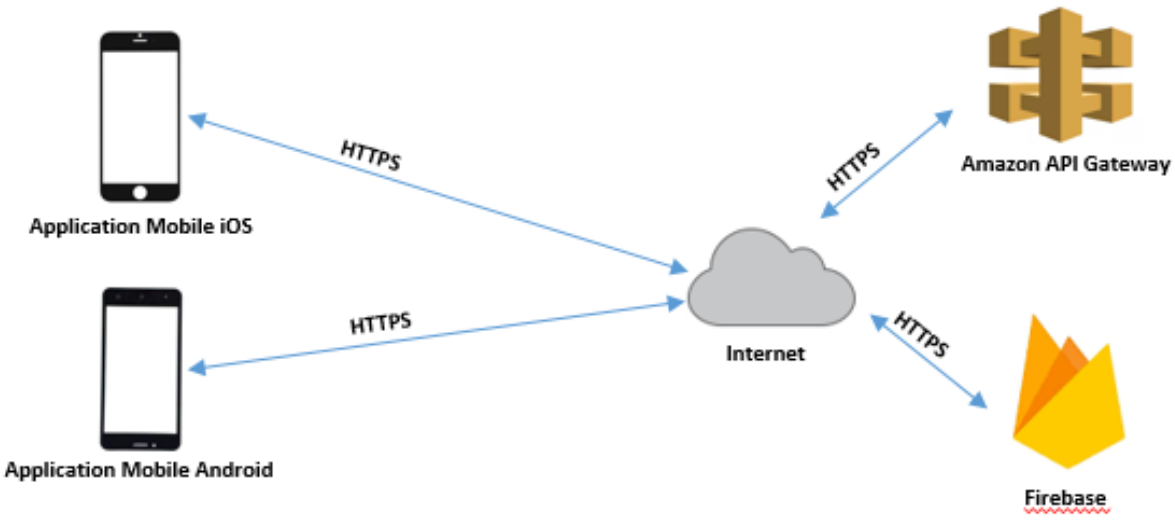
Émetteur	Destinataire	Lien / Type	Description
Google Firebase Cloud Messaging (FCM)	Google Firebase	Double Sens	Google Firebase via le service Firebase Core communique avec FCM pour la mise en œuvre des notifications push.
Google Firebase CrashLytics	Google Firebase	Double Sens	Google Firebase via et le service Firebase Core communique avec Firebase CrashLytics pour la remontée des crash et la génération des rapports de crash.



Google Firebase Analytics	Google Firebase	Double Sens	Google Firebase via et le service Firebase Core communique avec Firebase CrashLytics pour la génération des rapports d'utilisation des applications configurées.

5 APPLICATION MOBILE

5.1 POSITIONNEMENT DES APPLICATIONS MOBILES AU SEIN DE LA SOLUTION GLOBALE

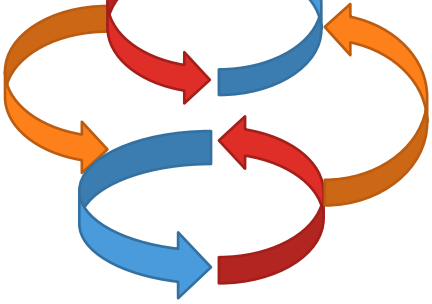


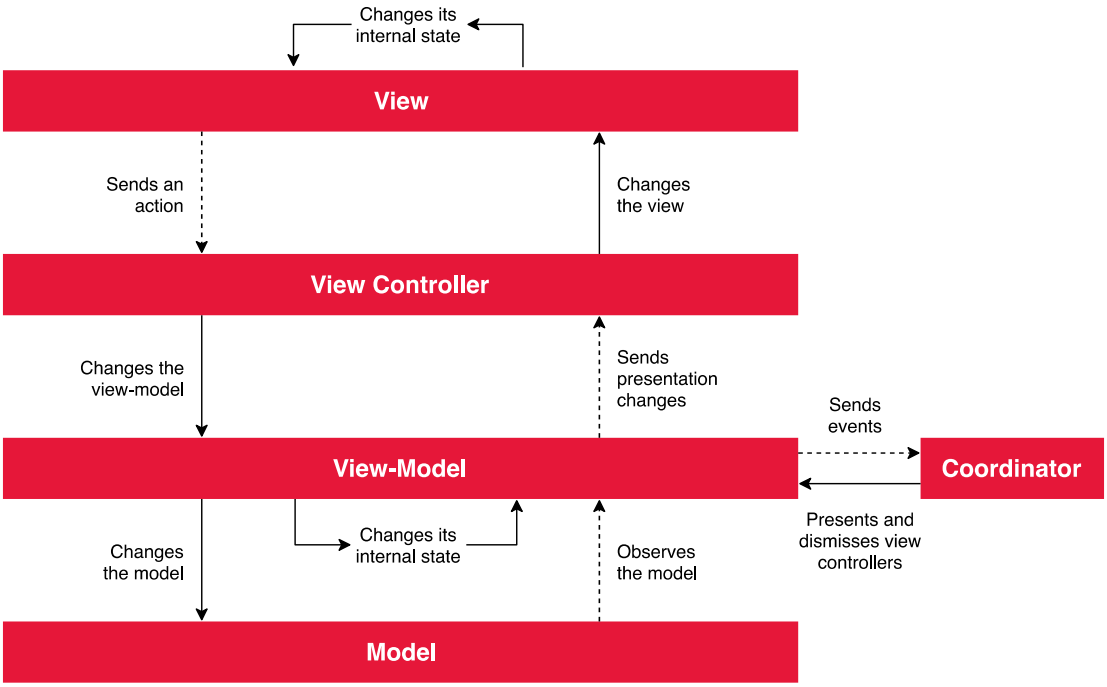
5.2 ARCHITECTURE DE L'APPLICATION MOBILE

5.2.1 Model-View-ViewModel with Coordinator (MVVM+C)

L'architecture cible identifiée pour l'application est l'architecture :

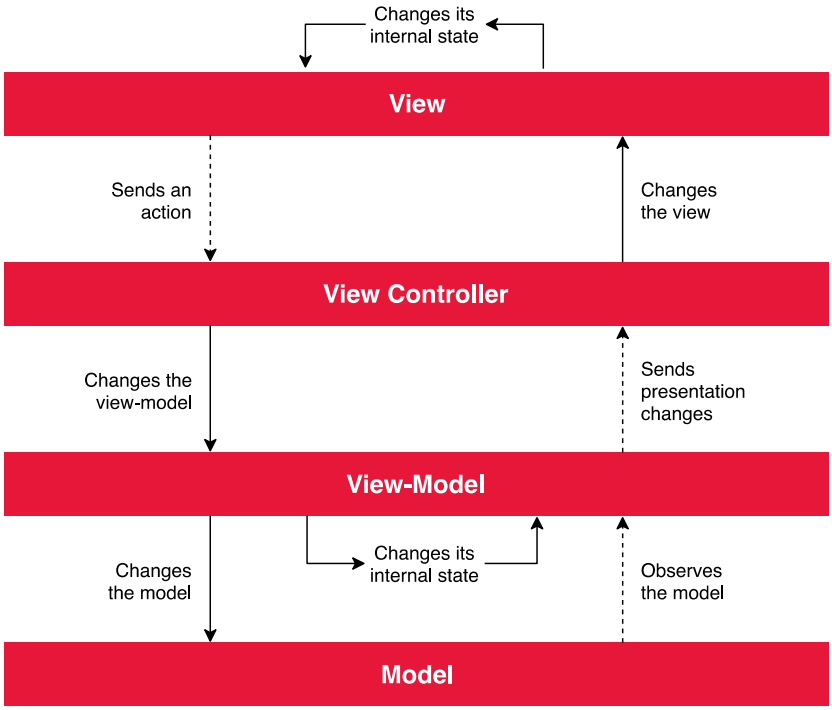
MVVM+C : Model-View-ViewModel with Coordinator

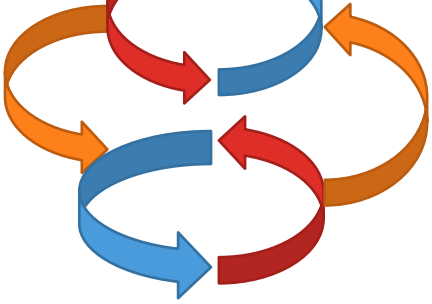




Note : Sur Android le View et le ViewController représentent une même entité.

5.2.1.1 Model-View-ViewModel (MVVM)





Cette architecture est une évolution du modèle d'architecture initial MVC introduit et préconisé par Apple pour iOS.

Depuis l'introduction des composants d'architecture Android, lors des Google I/O 2017, le MVVM est également préconisé par Android pour le développement des applications mobiles.

Source : https://developer.android.com/topic/libraries/architecture/viewmodel .

Model-View-ViewModel (MVVM) est un design pattern ayant pour objectif d'améliorer le patron ***MVC*** en déplaçant l'ensemble des tâches liées au modèle (mise à jour du *Model*, observation de ses changements, transformation des données du *Model* à des fins de présentation, etc.) de la couche *Controller* vers une nouvelle couche applicative nommée *View-Models*.

Le *View-Model* se situe alors, dans la plupart des implémentations d'app iOS, entre le *Model* et le *View Controller*.

Les objectifs de cette nouvelle couche *View-Models* sont :

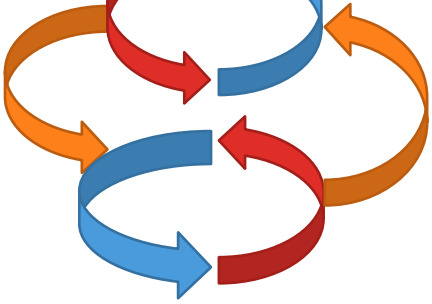
- D'encourager le fait de structurer les échanges entre le *Model* et la *View* par la mise en place d'un *pipeline* de transformations
- De fournir une interface indépendante du *framework* applicatif qui représente malgré tout l'état de présentation de la *View*

Ces deux points répondent aux deux plus grosses critiques envers le patron *MVC* :

- Le premier permet de réduire le périmètre de responsabilité des *View Controllers* en extrayant les tâches de transformation et d'observation du Model de la couche *Controller*.
- Le second fournit une interface claire de l'état de la vue d'une scène pouvant être testée indépendamment du *framework* applicatif as l'inverse des tests intégré du patron *MVC*.

Afin de synchroniser la *View* avec le *View-Model*, *MVVM* nécessite la mise en place de *bindings*, c'est-à-dire un mécanisme permettant de conserver les propriétés d'un objet synchronisées avec celles d'un autre objet.

Le *View Controller* construit ces *bindings* entre les propriétés exposées par le View-Model et les propriétés des vues de la scène que le *View-Model* représente.



5.2.1.2 Coordinator

La mise en place d'un composant *Coordinator* au sein de l'architecture applicative MVVM n'est pas obligatoire mais permet de décharger les *View Controllers* de deux autres responsabilités :

- Gérer la présentation des autres *View Controllers*.
- Arbitrer la communication des données *Model* entre *View Controllers*.

Le *Coordinator* est en charge de la gestion de la hiérarchie de *View Controllers* afin que chaque View Controller et View-Model se préoccupe uniquement de leur scène particulière.

5.2.2 Programmation réactive

L'utilisation d'une architecture MVVM se prête particulièrement bien à la mise en place d'un principe de programmation :

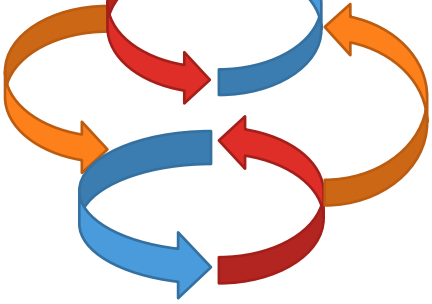
Programmation Réactive : ReactiveX
http://reactivex.io

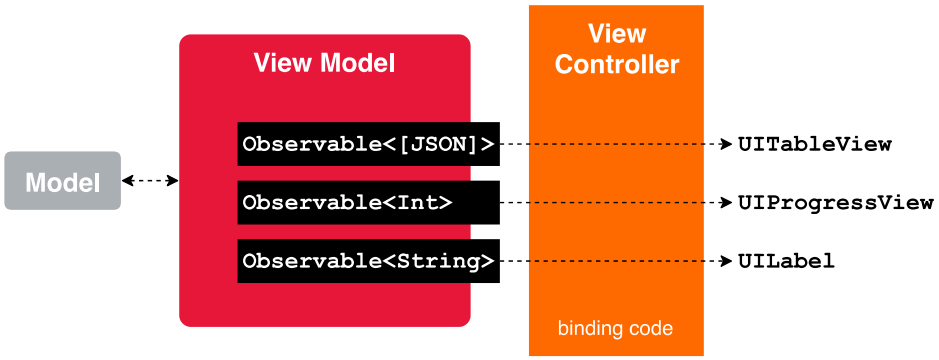
La programmation réactive permet de simplifier la gestion des programmes asynchrones par mise en place de flux d'événements observables et application d'opérations de transformation afin de garantir une cohérence d'ensemble du programme.

Dans notre architecture, la couche *ViewModel* est en charge d'exposer ces flux observables sous forme de propriétés spécifiques afin que les *View Controllers* effectuent le binding sur les différents éléments des *Views*.

La librairie *RxSwift* est l'instanciation Swift de *ReactiveX*. Cette librairie est utilisée afin de mettre en place les principes de programmation réactive et, couplée à RxCocoa, simplifie également les tâches de *binding* réactif entre les éléments de la vue avec la couche *ViewModel*.

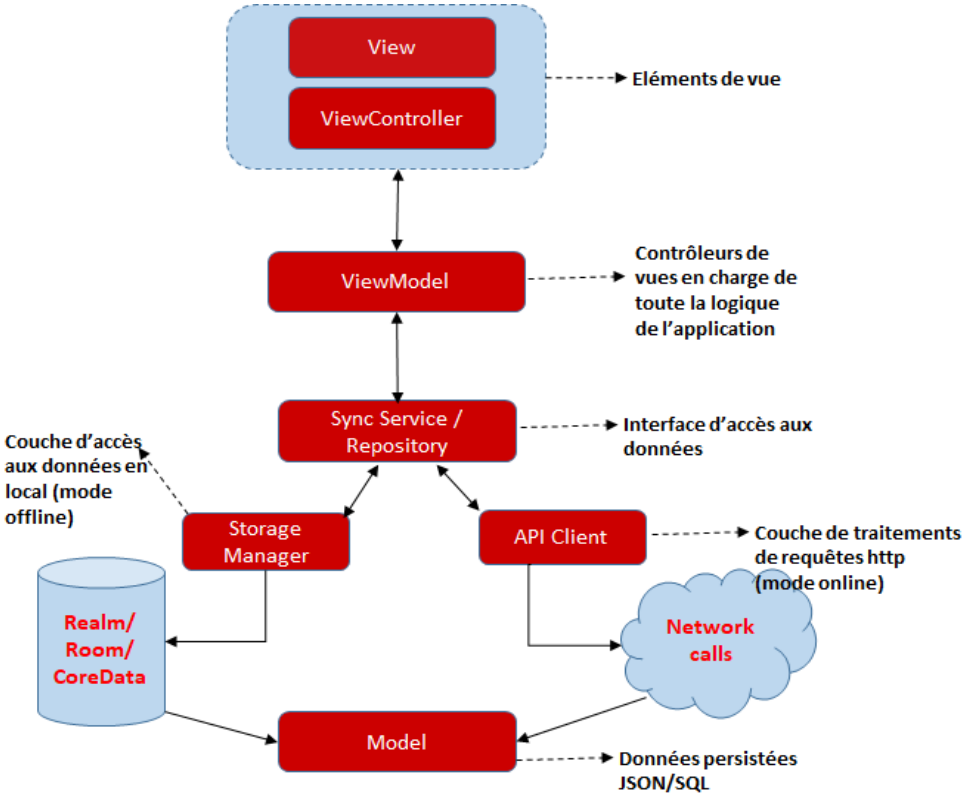
La librairie RxAndroid est l'équivalent de RxSwfit sur Android. Nous utiliserons également le LiveData qui fait partie des composants d'architecture Android permettant d'implémenter le concept de programmation réactive.

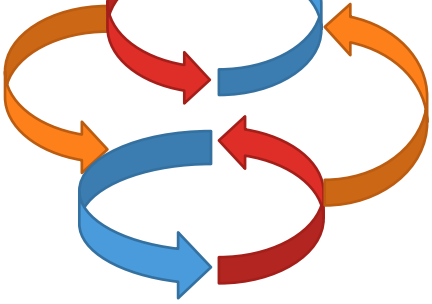




5.2.3 Architecture des couches de services.

5.2.3.1 Schéma d'architecture



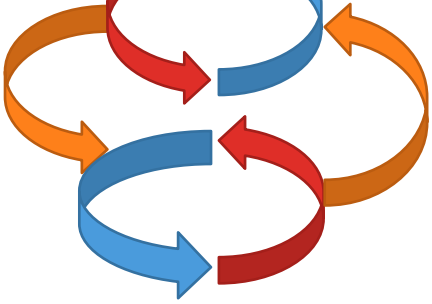


5.3 DESCRIPTION DES COMPOSANTS : ROLES ET RESPONSABILITES

Composant	Description	Responsabilités	« Caveats »
View	Ensemble de composants graphiques en charge de l'affichage des éléments dans l'application.	Affichage de l'UI et des divers items accessibles aux utilisateurs	Communique uniquement avec le ViewModel à travers des observables
View Controller	Composant en charge des bindings entre les *Views* et le *View-Model*.	Spécifique iOS, il permet de piloter des évènements relatifs au cycle de vie de la View	Communique uniquement avec le ViewModel à travers des observables
View-Model	Objet représentant l'état de la vue d'une scène et en charge des transformations et observations des objets métiers	Il écoute les interactions entre les utilisateurs et la View puis traite mise à jour de son état.	Communique uniquement avec le View et Repository via des observables
Model	Objets métiers de l'application	Il représente les données manipulées par l'application en local ou via des webservices.	Les données issues du Model ne sont accessibles que par le Sync Service ou Repository
Coordinator	Composant responsable de la navigation au sein de l'application	Il permet de centraliser et l'enchainement des écrans.	
Repository	Objet représentant l'interface d'accès aux données	Il est responsable des toutes les opérations de consultation/création/modification de données. Il utilise soit le Storage Manager pour les données en local ou l'API Client pour les requêtes http.	Communique uniquement avec le ViewModel, le Storage Manager et l'API Client via des observables
Storage Manager	Gestionnaire des opérations sur les données persistées en locale	Il est gère l'accès aux données stockés en local sur le device d'un utilisateur.	Communique uniquement avec le Repository
API Client	Gestion des requêtes HTTP	Il s'occupe de tous les appels webservices.	Communique uniquement avec le Repository

5.4 FLUX DE DONNEES

Émetteur	Destinataire	Lien / Type	Sécurisation	Description
App mobile	Backend BizzApp	HTTPS	TLS / SSL	Récupération des données de l'application mobile
App mobile	Firebase Cloud Messaging	--	--	Récupération d'un device token unique pour la transmission de push notifications Envoi de push notifications à l'application mobile



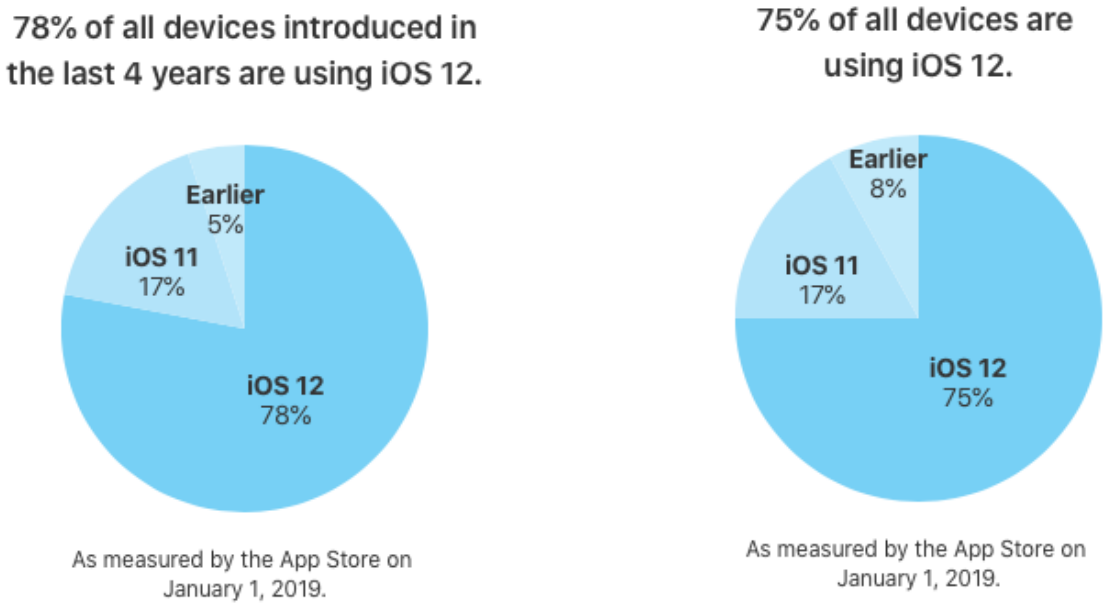
				Sur les applications iOS le service Apple Push Notification intervient également pour la gestion des notifications.
App mobile	Firebase CrashLytics	HTTPS	--	Envoi des données de crash de l'application
App mobile	Firebase Analytics	HTTPS	--	Envoi des données d'usage de l'application mobile

5.5 VERSIONS D'OS ET TYPES D'APPAREIL SUPPORTES

5.5.1.1 Application iOS

5.5.1.1.1 Statistiques actuelles des versions d'iOS installées

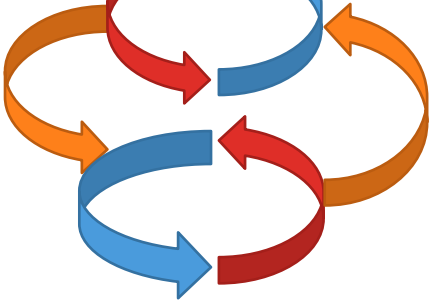
Fragmentation des versions d'iOS au 1er janvier 2019



Source : Apple, https://developer.apple.com/support/app-store/

5.5.1.1.2 Support cible de l'application

Versions d'iOS supportées	**iOS 11** et **iOS 12** uniquement.	➔ Permet d'adresser 92% des appareils iOS sur le marché



Type d'appareils supportés	iPhone uniquement.	➔ L'app n'est donc pas une Universal App (ne s'adapte pas aux formats d'écran des iPads).

5.5.1.2 Application Android

5.5.1.2.1 Statistiques actuelles des versions d'Android installées

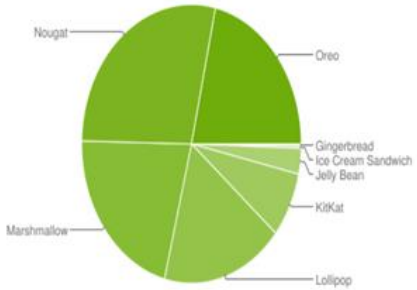
Fragmentation des versions d'Android au 26 Octobre 2018

APIs 19 and above cover 96,5% of the total market share of Android versions

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.1%
4.2.x		17	1.5%
4.3		18	0.4%
4.4	KitKat	19	7.6%
5.0	Lollipop	21	3.5%
5.1		22	14.4%
6.0	Marshmallow	23	21.3%
7.0	Nougat	24	18.1%
7.1		25	10.1%
8.0	Oreo	26	14.0%
8.1		27	7.5%

3,5%

96,5%

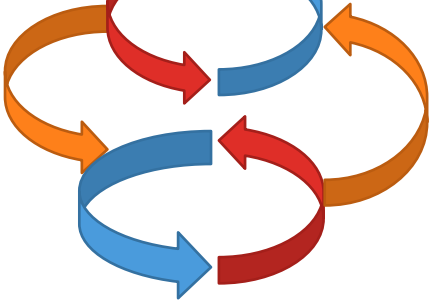


Data collected during a 7-day period ending on October 26, 2018 (update coming soon: data feed under maintenance). Any versions with less than 0.1% distribution are not shown.

Source : Google, https://developer.android.com/about/dashboards/

5.5.1.2.2 Support cible de l'application sur Android

Versions d'Android supportées	**Toutes les versions >= API 19**	➔ Permet d'adresser 96,5% des appareils Android sur le marché
Type d'appareils supportés	Smartphone uniquement.	➔ L'app n'est donc pas une Universal App (ne s'adapte pas aux formats d'écran des tablettes Android).



5.6 STACK TECHNIQUE & TECHNOLOGIES UTILISEES

5.6.1 Core technologies

5.6.1.1 Application iOS

Technologie	Version	Description
iOS SDK	Basé sur les versions d'iOS supportées	Framework Apple pour le développement d'application iOS native.
Swift	Dernière version stable	Langage de programmation pour les applications iOS
XCode	Dernière version stable	Environnement de développement intégré pour ls applications iOS

5.6.1.2 Application Android

Technologie	Version	Description
Android SDK	Basé sur les versions d'Android supportées	Framework Google pour le développement d'application Android native.
Kotlin	Dernière version stable	Langage de programmation pour les applications Android.
Android Studio	Dernière version stable	Environnement de développement intégré pour ls applications Android.

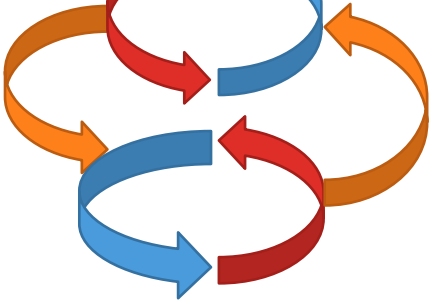
5.6.2 Gestion des dépendances (package management)

5.6.2.1 Application iOS

La gestion des dépendances sera effectuée à l'aide de :

Carthage

https://github.com/Carthage/Carthage



En cas d'utilisation de librairies non compatibles (*subspec* par exemple), l'utilisation de *CocoaPods* en complément de *Carthage* pourra être envisagée.

Carthage est privilégié à *CocoaPods* pour de multiples raisons :

- **Plus rapide :** les dépendances ne sont pas recompilées à chaque *Clean* de la solution
- **Plus moderne :** Carthage est écrit en Swift et évite les potentielles erreurs liées aux pré-requis Ruby de CocoaPods tant sur les postes de développement que sur les éventuels serveurs *CI*.
- **Plus simple :** Basé sur Git, l'ajout de dépendance avec Carthage s'effectue sans configuration complexe tant pour les frameworks publics qu'internes à l'application
- **Moins intrusif :** l'impact de Carthage sur la solution étant moindre (pas de création/modification du xcworkspace, modification limitée du pipeline de build, etc.), les adhérences entre l'application et le gestionnaire de dépendances sont plus limités ainsi que les risques d'erreur et de blocage associés.

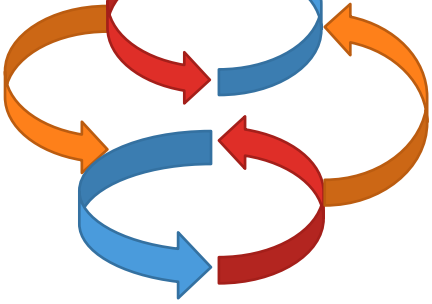
5.6.2.2 Application Android

La gestion des dépendances sera effectuée avec Gradle qui est le système de build embarqué dans Android Studio.

5.6.3 Librairies et frameworks tiers

5.6.3.1 Librairies communes à iOS et Android

Technologie	Version	Description
Amazon Amplify https://aws.amazon.com/amplify/	Version stable la plus récente	Authentification des utilisateurs de l'app avec Amazon Cognito
Firebase Analytics https://firebase.google.com/docs/analytics	Version stable la plus récente	Librairie permettant de transmettre les statistiques d'utilisation de l'application à la solution backend dédiée à cela.
Firebase Crashlytics https://firebase.google.com/docs/crashlytics	Version stable la plus récente	Librairie permettant de transmettre les informations de crash de l'application à la solution backend dédiée à cela.
Realm https://realm.io	Version stable la plus récente	Librairie permettant de stocker les données localement pour mise en place du mode offline : disponibilité des données en mode hors-connexion.
Firebase Cloud Messaging https://firebase.google.com/docs/cloud-messaging	Version stable la plus récente	Librairie permettant l'implémentation des notifications push.



5.6.3.2 *Application iOS*

Technologie	Version	Description
RxSwift / RxCocoa / RxSwiftCommunity https://github.com/ReactiveX/RxSwift https://github.com/RxSwiftCommunity	Versions stables les plus récentes	Librairies permettant la mise en place des patterns de programmation réactive.
Moya / RxMoya https://moya.github.io	Versions stables les plus récentes	Librairies permettant d'abstraire la couche réseau de l'application mobile.

5.6.3.3 *Application Android*

Technologie	Version	Description
RxAndroid https://github.com/ReactiveX/RxAndroid	Versions stables les plus récentes	Librairie permettant la mise en place des patterns de programmation réactive sur Android
Retrofit https://square.github.io/retrofit/	Versions stables les plus récentes	Librairie permettant d'abstraire la couche réseau de l'application mobile sur Android.
Dagger https://google.github.io/dagger/	Versions stables les plus récentes	Librairie permettant de faire de l'injection de dépendances

5.7 AUTHENTIFICATION

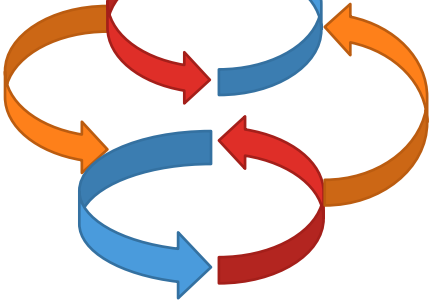
5.7.1 Identifiants de l'utilisateur

AWS Cognito est en charge de la gestion des accès utilisateurs.

5.7.2 Reconnaissance biométrique

L'authentification initiale de l'app reposera sur le couple identifiant et mot de passe de l'utilisateur.

En revanche, suite à la connexion initiale de l'utilisateur dans l'application, le processus d'authentification sera optimisé à l'aide d'une authentification secondaire « allégée ».



Pour mettre en œuvre cette fonctionnalité d'authentification, l'application tirera parti des capacités des appareils iOS actuels dans une démarche de type *Progressive Enhancements*.

L'authentification secondaire tirera parti des capacités suivantes (par ordre de priorité) :

Face ID	Identification à l'aide de la fonctionnalité native d'iOS permettant la reconnaissance faciale de l'utilisateur de l'appareil iOS.
Touch ID / Fingerprint	Identification à l'aide de la fonctionnalité native d'iOS/Android permettant la reconnaissance de l'empreinte digitale de l'utilisateur de l'appareil iOS/Android.
Saisie du code simple	Identification de l'utilisateur à l'aide du code simple défini lors de la création du compte.
Renvoi vers l'authentification primaire	En cas d'échec du processus d'authentification secondaire. L'utilisateur est renvoyé vers la procédure d'authentification primaire. Il doit donc ressaisir son couple identifiant et mot de passe.

La mise en place de ce processus d'authentification secondaire devra être pris en compte lors de l'authentification initiale et lors de la création d'un compte :

- Création du code simple (avec vérification)
- Activation de *Face ID* ou *Touch ID (Fingerprint sur Android à partir de API 23)*

L'utilisateur devra pouvoir choisir d'utiliser ou non les fonctionnalités de reconnaissance biométrique au travers du paramétrage de l'application.

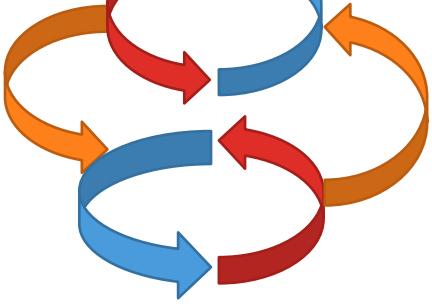
5.8 SECURISATION

5.8.1 Stockage des identifiants de l'utilisateur et des données confidentielles

Les règles de sécurisation suivantes seront mises en place :

Règle de sécurisation	Priorité
Si le stockage du mot de passe de l'utilisateur s'avère nécessaire, celui-ci sera stocké, dans le respect des standards de sécurité, à l'aide du *Keychain* d'iOS ou du KeyStore d'Android.	MUST
Aucune donnée confidentielle ne sera stockée au travers : • Des préférences utilisateurs (*NSUserDefaults sur iOS ou SharedPreferences sur Android*) • Du système de fichiers (sandbox de l'app : Documents/, Library/, tmp/, etc.) • Du stockage iCloud	MUST

5.8.2 Sécurisation des échanges / communications réseaux



Les règles de sécurisation suivantes seront mises en place :

Règle de sécurisation	Priorité	Plateformes
Tous les appels réseaux sont effectués en HTTPS (HTTP over TLS). Les certificats SSL en question sont valides et ne nécessite pas d'exceptions de sécurité dans l'application mobile.	MUST	iOS et Android
App Transport Security : la valeur *NSAllowsArbitraryLoads* et ses dérivés ne sont pas positionnées à *true*.	MUST	iOS
Prévention des attaques de type *Man-In-The-Middle* (*MitM*) par mise en place de *SSL Certificate Pinning*	SHOULD	iOS et Android
Mise en place d'un algorithme avancé de sécurisation des échanges : SRP (Secure Remote Password)	COULD	iOS et Android

5.9 PUSH NOTIFICATIONS

Firebase Cloud Messaging est en charge des notifications push.

5.10 STOCKAGE DES DONNEES

5.10.1 Mode offline

Les données locales seront stockées à l'aide de la base de données ***Realm*** non cryptées par défaut. Techniquement on a la possibilité de le faire sous réserve des prérequis par plateforme.

5.10.2 Modèle de données

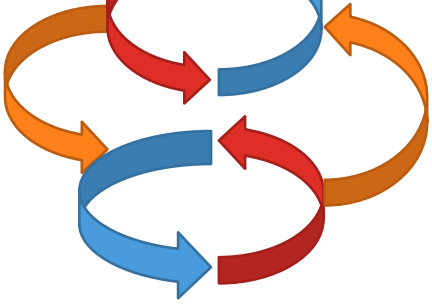
Le modèle de données local sera construit à l'aide de *POSO* (Plain Old Swift Object) / POJO (Plain old Java Object) convertible depuis les réponses de la *REST API* (*JSON*) et compatible avec le stockage *model object* de *Realm*.

La définition du modèle de données local ainsi que des objets métiers associés sera effectué suite à la définition des contrats *Swagger* de la *REST API* du *backend* ainsi qu'en fonction des exigences fonctionnelles pour la mise en œuvre du mode déconnecté de l'application.

5.10.3 Préférences applicatives

L'application bénéficiera de deux types de préférences applicatives :

- Préférences de l'application



- Préférences liées au compte de l'utilisateur

Les préférences applicatives pourront ou non être synchronisés avec le backend :

- Pour les préférences locales non synchronisées : l'utilisation de *User Defaults* (*NSUserDefaults*) sera privilégiée sur iOS et les SharedPreferences sur Android.
- Pour les préférences synchronisées avec le backend : celles-ci seront intégrées au modèle de données local basé sur *Realm*.

5.11 Statistiques d'utilisation (Analytics)

5.11.1 Stratégie de mise œuvre

Mise en place d'une couche d'abstraction permettant de minimiser l'adhérence de la solution d'Analytics avec le projet :

- Création d'un tracker d'événements générique basé sur un protocole permettant l'abstraction de la solution sous-jacente utilisée.

5.11.2 Solution envisagée

Solution envisagée : Firebase Analytics.

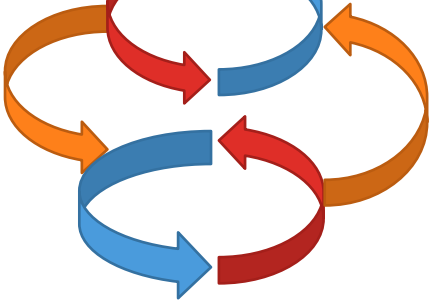
5.12 Crash reporting (Crash analytics)

5.12.1 Stratégie de mise œuvre

La solution de crash reporting devra être mise en œuvre avant le déploiement de la première beta.

5.12.2 Solution envisagée

Solution envisagée : Firebase Crashlytics.



5.13 SIGNATURE, CERTIFICATS ET PUBLICATION DE L'APP

5.13.1 Signature et certificats

Toutes les applications doivent être signés avec des certificats générés dans le cadre de ce projet et les informations de générations ainsi que les fichiers générés doivent être stockés avec les autres ressources du projet.

5.13.2 Publication de l'application BizzApp

Disposer d'un compte Apple Developer et Google Play Developer pour les app publiées au nom de BizzApp.

- Sur Android, il faut publier un appBundle au lieu d'un apk.
- **Attention sur iOS App Store il faut tenir compte des guidelines pour les templates app**

https://techcrunch.com/2017/12/20/apple-revises-its-controversial-guidelines-on-template-based-apps/

5.14 INTERNATIONALISATION & LOCALISATION

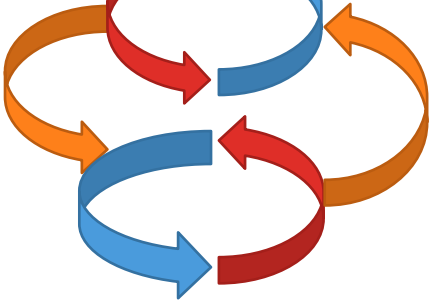
Les exigences de conception suivantes seront mises en place :

Règle de sécurisation	Priorité	Plateforme
L'application n'est pas internationalisée / localisée. L'application est donc disponible en Français uniquement.	MUST	iOS et Android
L'internationalisation de l'application est anticipée en prévoyant dès le début d'activer le mode ***Base Internationalization*** et en gérant tous les libellés dans les fichiers de ressource ****.strings* même s'il n'y a qu'une seule langue disponible.	SHOULD	iOS
Tous les libellés ou textes statiques contenues dans l'application doivent être stockées dans le fichier **res/values/strings.xml** qui représente la locale par défaut de l'application. Lors de la déclinaison de l'application dans une autre langue il faudra créer un fichier de ressources supplémentaire (Pour l'anglais res/values-en/strings.xml).	MUST	Android

5.15 NORMES DE DEVELOPPEMENT

iOS

- ⇨ Must : Apple Best Practices for iOS mobile App Development
- ⇨ Must : follow Apple HIG (impact sur le travail UX/UI pré-existant)



⇨ Could : SwiftLinter

RX best practices / VM

Networking => Api Client

Android

⇨ Must : Best practices Android / Kotlin dev

⇨ Should : Android Design Guidelines / Material design

⇨ Could : Linter (Android Lint)

5.16 Pipeline DevOps Et Versionnement des sources

- AWS CodeCommit
- AWS CodePipeline

5.17 Gestion des multiples versions de l'application, de la configuration et des differents environnements

Common :

Must : stratégie ➔ feature flags

iOS :

Pour les versions d'app : scheme XCode dédié pour chaque version.

Gestion de la configuration, des environnements : fichiers de ressources (plist, …).

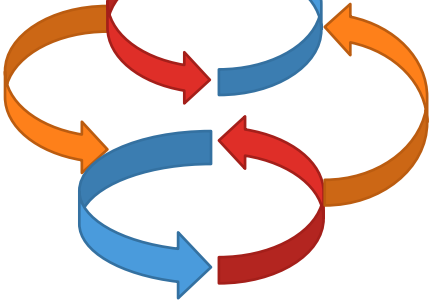
Android :

Version app : Flavor.

Gestion de configuration : fichier de ressource.

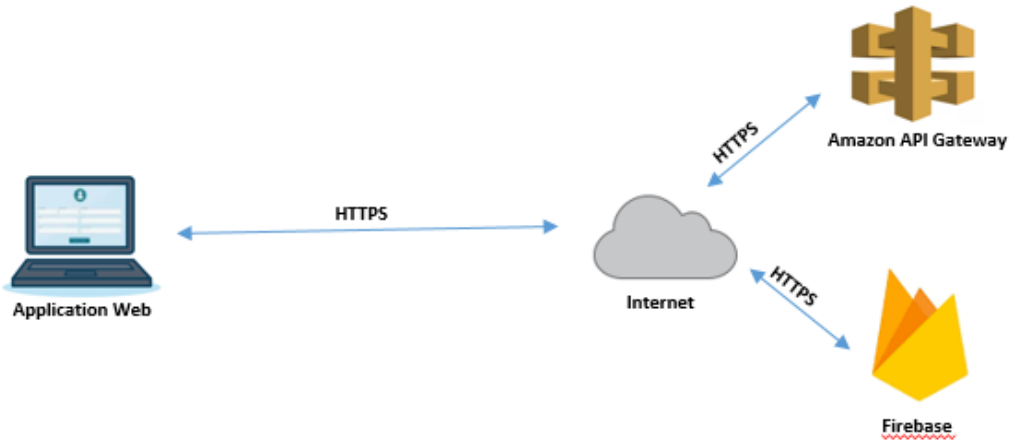
5.18 Strategie de test & tests automatises

Les tests unitaires et les tests automatisés sont développés en amont de tout développement.

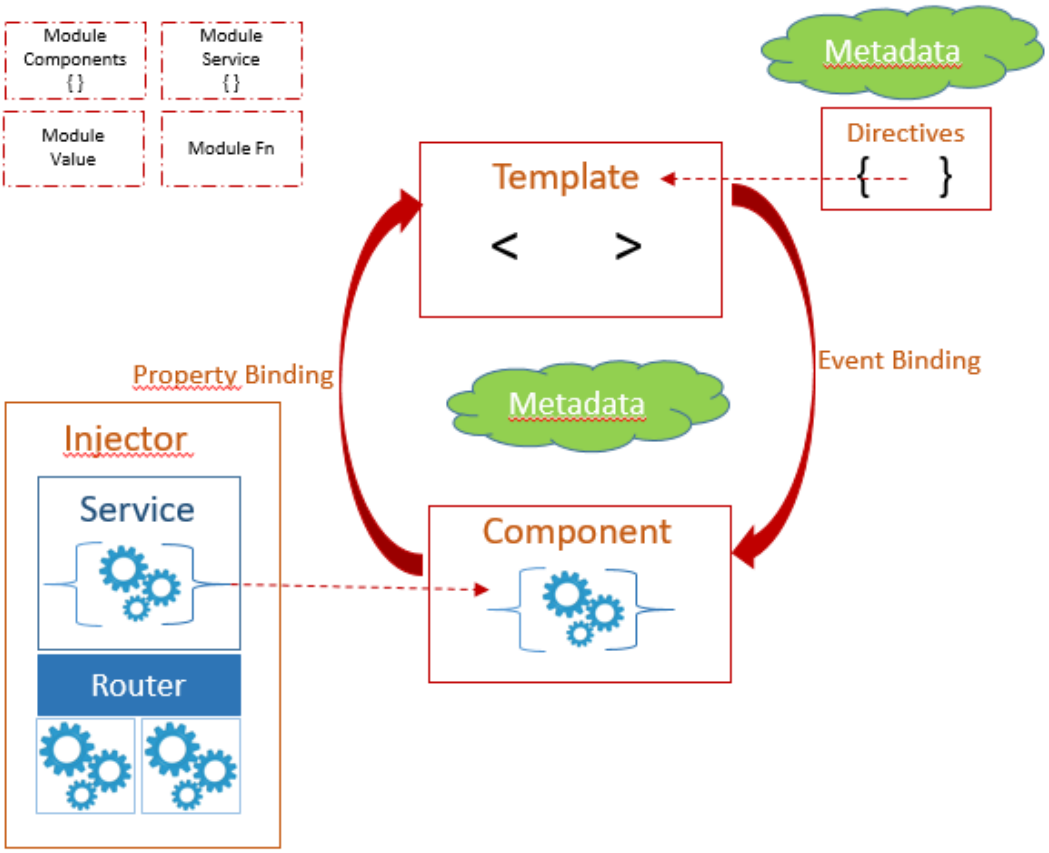


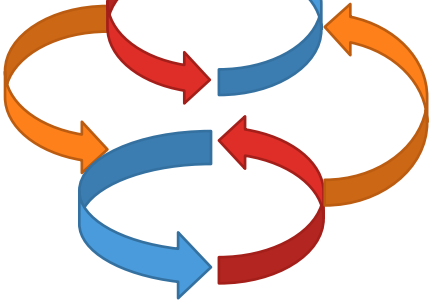
6 APPLICATION WEB

6.1 POSITIONNEMENT DE L'APPLICATION DANS LA SOLUTION



6.2 ARCHITECTURE DE L'APPLICATION WEB



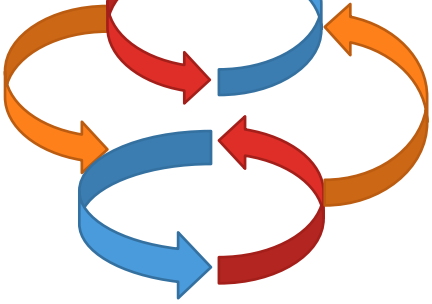


6.3 DESCRIPTION DES COMPOSANTS

Composant	Description
Template	Le template essentiellement dédié à l'UI combine de l'HTML et des balises Angular pouvant modifier les éléments HTML avant leur affichage.
Component	Un Component définit une classe qui contient des données d'application et une logique, et est associé à un modèle HTML qui définit une vue à afficher dans un environnement cible.
Metadata	Ce sont des données passées dans un décorateur afin de créer une classe spécifique (Component ou Service)
Directive	Les directives de Template fournissent une logique de programme et le balisage de liaison qui connecte les données d'application et le modèle d'objet de document (DOM).
Service	C'est une classe dont les données ou la logique ne sont pas associées à une vue spécifique et que l'on souhaite partager entre les composants.
Injector	L'Injector utilise l'injection de dépendance (ou DI) qui permet de garder les classes de composants légers et efficaces tout en fournissant les objets nécessaires à l'implémentation des logiques métier.
Event Binding	L'Event Binding permet à l'application de répondre aux entrées de l'utilisateur dans l'environnement cible en mettant à jour les données de votre application.
Property Binding	Le Property Binding permet d'interpoler les valeurs calculées à partir des données de l'application dans le code HTML.
Router	Le Router fournit un service qui permet de définir un chemin de navigation parmi les différents états de l'application et d'afficher les hiérarchies dans de l'application. Le routeur mappe les chemins de type URL aux vues au lieu des pages.

6.4 FLUX DE DONNEES

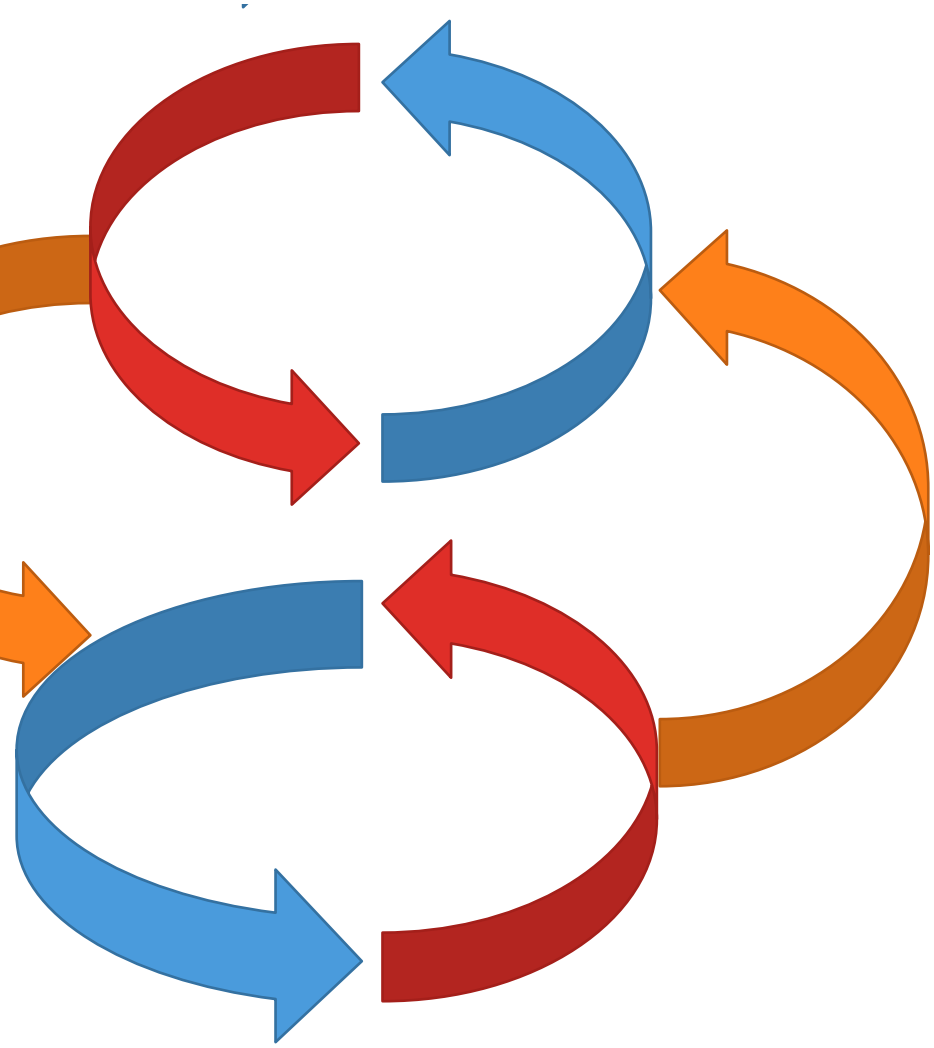
Émetteur	Destinataire	Lien / Type	Description
Service	Component	Sens Service -> Component	Les services sont intégrés dans les Component via l'injection de dépendance
Component	Template	Double Sens	Les Component et les Template s'échangent des données dans les 2 sens c'est le two-way data binding.
Router	Component	Sens Router -> Component	Les Component utilisent les services du Router pour la navigation dans l'application.



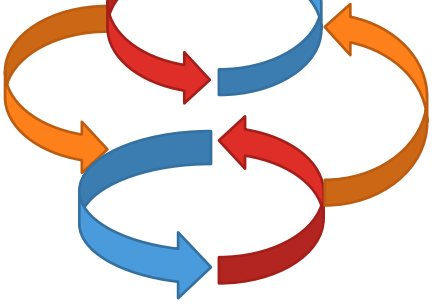
6.5 STACK TECHNIQUE & TECHNOLOGIES UTILISEES

- Angular 8
- TypeScript
- Visual Studio Code

Les autres points relatifs au développement de l'application web (tests unitaires, devops, source de contrôle) sont identiques à ceux évoqués pour les applications mobiles.



ANNEXES



ANNEXE A

Quelques ressources utilisées dans l'élaboration de ce document

- https://angular.io/guide/architecture
- https://docs.aws.amazon.com
- https://firebase.google.com/docs
- https://developer.android.com/docs
- https://developer.apple.com/documentation