



Dossier d'architecture et de conception, partie application mobile

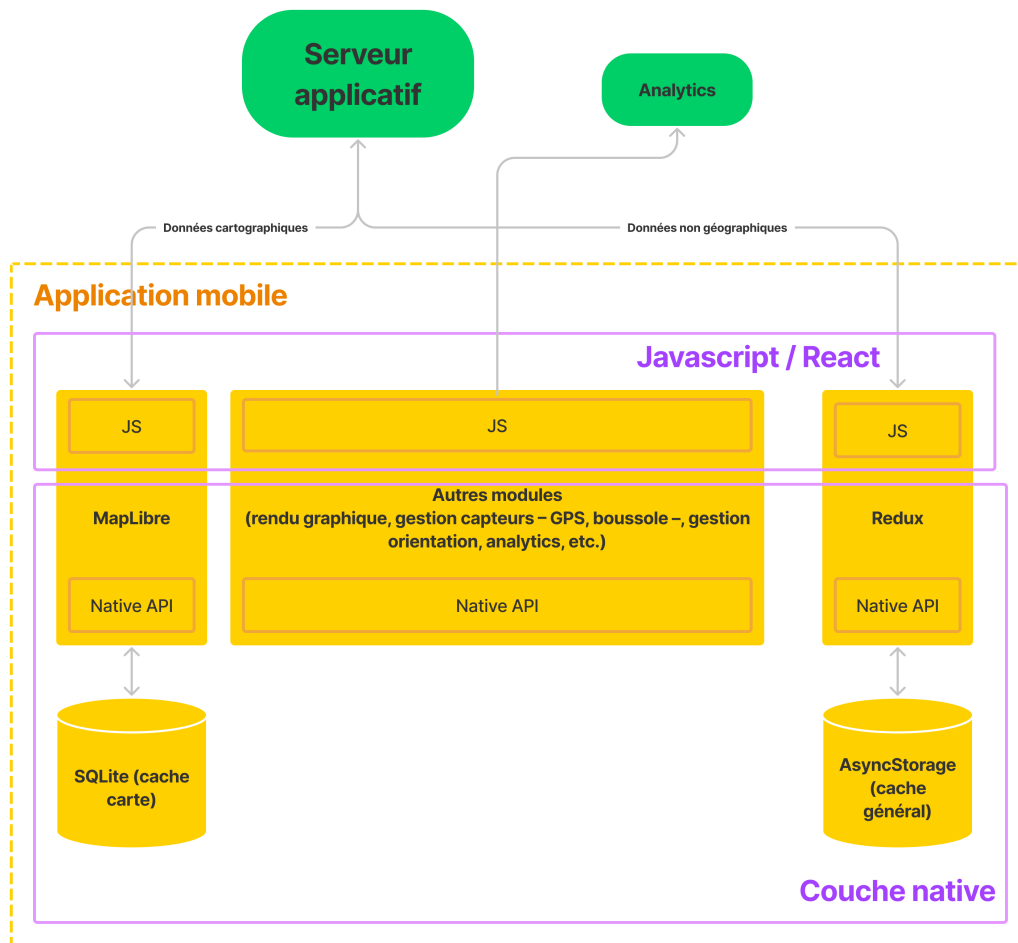
Contexte

Besoins fonctionnels et non-fonctionnels

Les besoins sont exprimés dans le **Cahier des clauses techniques particulières**, répartis selon le type (fonctionnels ou non-fonctionnels) et la priorité associée (P1 ou P2 pour les postes de développement initiaux, voire P3 pour les fonctionnalités envisagées).

⇒ Cf. CCTP

Architecture simplifiée



L'application mobile suit les préceptes de conception et de réalisation de React Native.

Elle communique avec le serveur applicatif.

À noter que le serveur Analytics n'est pas implémenté suite au retrait de Google Analytics et au remplacement éventuel par une solution alternative.

React Native, comme base applicative

Le choix s'est porté sur React Native étant la plateforme déjà en usage par le démonstrateur et de sa capacité à répondre

aux nouvelles exigences fonctionnelles et non-fonctionnelles.

Les plus de React Native : *cross-platform* Android et iOS, modules natifs, forte communauté.

Un **module React Native** est constitué à la fois :

- d'une partie **JavaScript**, langage hérité du Web, commun iOS/Android,
- et d'une partie en code natif pour Android (en **Java** le plus souvent) et pour iOS (en langages **Objective-C** ou **Swift**), propre à assurer des performances importantes par rapport à un système Web classique.

Le développement standard de l'application en elle-même se fait en **Typescript** (lui-même ensuite compilé en Javascript).

Protocoles de communication avec le serveur applicatif

L'application mobile doit disposer de l'accès à plusieurs ressources externes :

- à un serveur de diffusion de la donnée géographique, qui diffuse ces données,
 - pour les données vectorielles (POI environnementaux, balisage, AVINAV/AVURNAV, réglementation, etc.), sous la forme de flux WMTS, WMS, WFS
 - et pour les tuiles cartographiques Raster (du RasterMarine), comme proxy vers le service de diffusion de ces tuiles sur data.shom.fr, sous la forme de requêtes HTTP de fichiers images en x (longitude), y (latitude) et z (niveau de zoom).
- à une API rendant les photos des balises & amers, et des POI environnementaux (proxy).
- à une API de *back office* pour la gestion des données utilisateurs (gestion de l'authentification, éléments de profil, traces de navigation, etc.).
- à une API de publication des données de suivi de la navigation au sein de l'application (*analytics*).

Interface cartographique - MapLibre

La bibliothèque MapLibre est utilisée pour présenter l'interface cartographique. En tant que projet dérivé (*fork*), ses caractéristiques sont semblables au module utilisé par le démonstrateur (MapBox). Le principal avantage est d'être complètement open source et donc d'être modifiable en fonction des besoins, tout en reprenant les mêmes performances d'usage et les mêmes fonctionnalités.

Gestion des données internes - Redux

Les données issues du serveur et devant être mises en cache ou générées au sein même de l'application (par exemple, sauvegarder l'état des filtres cartographiques sélectionnés ou le maintien de l'authentification), doivent être stockées dans une base de données interne à l'application. Cette « persistance » est rendue possible par l'usage d'un concept lié à React appelé *Redux*. Il s'agit d'une bibliothèque open source, de gestion d'état largement utilisée par la communauté React Native. En outre, l'adjonction d'un module complémentaire appelé *Redux persist* permet de sauvegarder automatiquement l'état des différentes données lors de la fermeture de l'application puis de les retrouver à la réouverture.

La gestion des cartes hors-lignes se fait un peu différemment de par la nature des objets stockés (des images de tuiles) et de la bibliothèque utilisée (MapLibre). Une base de données SQLite est créée à cet effet et stocke les données de cache purement associées à MapLibre. Ainsi, chaque zone cartographique téléchargée pour être disponible hors ligne est stockée sans limite de durée (à moins que l'utilisateur décide de supprimer ce téléchargement hors ligne). Et chaque parcelle visualisée à l'écran, qui ne fait pas partie d'une carte hors ligne est aussi stocké temporairement en cache (pendant 48 heures) afin de réduire le besoin de téléchargement pour des zones géographiques consultées fréquemment.

Architecture détaillée

Organisation générale

La gestion standard d'un projet react-native nécessite de séparer les sources javascript du code natif iOS / Android. Seul le point d'entrée et les fichiers de configuration sont présents à la racine. Les sources sont ensuite réparties selon l'architecture suivante.

Data / Services

Les dossiers `data` et `services` comportent l'ensemble des éléments fonctionnels qui gèrent l'état de l'application et la communication avec le serveur. Chaque sous-dossier correspond à un type de donnée différent. Pour chaque type de donnée, un fichier « Action » indique la liste des actions pouvant être effectuées sur ce type de donnée et un fichier « Reducer » contient l'état initial lors du premier lancement de l'application ainsi que le code à exécuter pour chacune des actions.

Components

Le dossier `components` contient l'ensemble des composants (graphiques et/ou fonctionnels) génériques à l'application et communs à différents écrans. Il s'agit des différents composants d'interaction (boutons, champs de saisie), des vue modales et des composants d'affichage d'un type de donnée (affichage du profil, d'un bateau ...)

Screens

Ensemble des écrans de l'application. Les écrans sont répartis en sous-dossiers et regroupés par utilisation. Liste des dossiers et écrans inclus :

- `about/` : à propos
- `contactCross/` : page de contact du CROSS
- `help/` : pages d'aide, CGUs, accès à l'onboarding
- `home/` : page d'accueil
- `legalNotices/` : mentions légales
- `login/` : connexion, mot de passe oublié
- `map/` : écran principal d'affichage de la carte, sélection des catégories à afficher, détails des zones et POIs
- `offline/` : liste des cartes hors-lignes, téléchargement d'une carte
- `onboarding/` : tutoriel de l'application

- `region/` : sélection de la région lors du premier lancement
- `register/` : inscription utilisateur
- `tide/` : liste des ports, visionnage des marées
- `trips/` : visualisation des trajets effectués
- `updateRequired/` : mise à jour de l'application requise
- `user/` : visionnage du profil, modification des données, modification du mot passe
- `webViews/` : pages d'information diverses (réglementation pêche, météo, RIPAM, etc..)

Ressources assets

Ensemble des ressources (Images) utilisées dans l'application. Celles-ci sont pré-chargées au lancement de l'application afin de pouvoir être récupérées et affichées rapidement au besoin.

Utils

Le dossier `utils/` contient l'ensemble des fichiers utilitaires nécessaires pour la majorité des composants

- `localization` : Gestion des chaînes de caractères et des traductions
- `AppInfoManager` : Vérification de la version courante de l'application comparée à la version disponible et/ou requise
- `functions` : Fonctions génériques de traitement des chaînes de caractère
- `permissions`: Vérification des permissions et de l'application et demande d'autorisation si nécessaire
- `selection` : « font-icon » de l'application
- `styles` : styles globaux de l'application (marges, polices ...)
- `themes` : liste des couleurs et dimensions des éléments de l'application

Utilisation de Redux

Principe et sous-modules

Le module **redux-persist** est utilisé pour le stockage interne et le contrôle des données de l'application. Il permet la persistance des données au sein de l'application complète et les conservent lors de la fermeture.

Le module **migration** de Redux permet de conserver la compatibilité entre les versions. Lors du téléchargement d'une nouvelle version, une fonction de migration est appelée pour transformer les données stockées actuellement sur l'appareil et les convertir au nouveau format.

Le module **redux-logger** permet de conserver une trace des toutes les actions effectuées par l'utilisateur ainsi que l'état courant des données. Ces logs peuvent être utilisés à des fins de début de l'application lors du développement ou peuvent être traitées ultérieurement dans le contexte de l'analytics.

Structure des données et actions

Le module Redux permet de séparer les données stockées en différentes catégories. Chaque catégorie est traitée indépendamment et possède un état initial et une list d'actions possibles. Ces catégories sont appelées de **reducers**.

Liste des reducers utilisés dans l'application :

Reducers

Nom	Données traitées	Actions
data.category	Catégories des informations affichées sur la carte sélectionnées, ainsi que les options additionnelles (limite de taille de bateaux ou affichage des réglementations associées à un AVURNAV)	Sélectionner/Désélectionner les catégories + options additionnelles
data.geolocation	Position actuelle de l'utilisateur	Modifier la position
data.harbors	Liste des ports	Mise à jour de la liste
data.map	Etat de la carte	Modifier la position de la caméra Modifier le niveau de zoom Modifier le contexte
data.offlineDatasets	Liste des cartes hors-ligne	Liste des cartes hors-ligne

data.tracking	Enregistrement de la trace en cours	Début/fin d'enregistrement Ajout d'un point Annulation de la trace
data.trip	Liste des traces de l'utilisateur	Ajout d'un trajet Modification d'un trajet
data.user	Compte de l'utilisateur connecté	Modification des données Déconnexion
services.device	Premier lancement de l'application	Afficher/cacher l'onboarding
services.globals	Variables globales de l'application	Modification d'une variable
services.session	Jetons et statut de la session de connexion	Modification du status Déconnexion