**Documentation de l’application BoredOnBoard (BOB)**

**1. Introduction**

**1.1 Contexte et objectifs**

BoredOnBoard (BOB) est une application mobile hybride pionnière, conçue pour transformer les vols, qu’ils soient de courte ou longue distance, en expériences immersives et interactives. Initié en réponse à l’essor des vols low‑cost et à la demande croissante de solutions de divertissement personnalisées, BOB se positionne comme un compagnon de voyage capable d’allier services pratiques, distraction ludique et outils de socialisation.

**Origine du projet :**

* Né d’un constat : l’ennui et le manque d’informations fiables à bord génèrent inconfort et frustration.
* Inspiré par les retours des passagers et l’analyse des stratégies de fidélisation des compagnies aériennes, BOB vise à offrir une proposition de valeur supplémentaire, tant pour les voyageurs que pour les compagnies souhaitant enrichir leurs services complémentaires.

**Objectifs principaux :**

1. **Amélioration de l’expérience passager** : réduire la perception du temps de vol et proposer un contenu pertinent selon la phase du voyage (embarquement, vol, escale, arrivée).
2. **Mutualisation des données de vol et météo** : agréger plusieurs sources (FR24, AWVX, METAR/TAF) pour offrir des informations fiables, à jour et géolocalisées.
3. **Engagement et monétisation** : favoriser la découverte de services payants (premium maps, jeux exclusifs, partenariats avec des commerces locaux en escale) et la génération de leads pour les partenaires (agences de voyage, tour-opérateurs).
4. **Extensibilité et scalability** : architecture modulaire permettant d’ajouter facilement de nouveaux modules (réalité augmentée, contenus sponsorisés, intégration réseaux sociaux).

**Valeur ajoutée :**

* **Plug & Play** : installation rapide via Ionic/Capacitor, compatible iOS/Android, déploiement continu sans contraintes natives lourdes.
* **Offline first** : modes cache et synchronisation différée, garantissant l’accès aux données même en l’absence de connexion.
* **Personnalisation avancée** : profils utilisateurs, recommandations de contenus selon historique, préférences et destination.

**1.2 Portée et audience du document**

Ce document se veut un guide exhaustif, adapté à plusieurs profils :

* **Développeurs front-end** : compréhension des composants Angular standalone, du système de routing Ionic, et des conventions de structuration SCSS.
* **Développeurs back-end** : détails sur le Proxy Flight Data, l’implémentation Node.js/Express ou NestJS, et les bonnes pratiques de sécurisation des endpoints.
* **Architectes et chefs de projet** : analyse des patterns d’architecture (Clean Architecture, Domain Driven Design), maintien de l’intégrité du code, et modes de gouvernance des évolutions.
* **QA et DevOps** : configuration des workflows GitHub Actions, mise en place de tests E2E (Cypress), tests unitaires (Jest/Karma), et déploiement automatisé via Capacitor et App Center.
* **Stakeholders métier** : aperçu fonctionnel pour valider la roadmap, aligner les objectifs marketing (monétisation, partenariat) et anticiper les besoins de support client.

Le document est structuré pour permettre une lecture sélective : chaque chapitre peut être appréhendé indépendamment, avec un glossaire des termes techniques et une section FAQ en fin de document.

**1.3 Technologies clés**

BOB repose sur un ensemble de technologies éprouvées, articulées pour garantir performance, maintenabilité et réactivité :

* **Ionic 6 & Angular 16 (Standalone Components)** : adoption des dernières fonctionnalités Angular pour réduire la complexité du code, faciliter l’arborescence des composants et améliorer la build size.
* **Firebase Authentication & Firestore** : infrastructure serverless pour l’authentification (OAuth, Email/Password, SSO via Google, Apple) et la base NoSQL en temps réel, avec règles de sécurité et indexing avancé.
* **NestJS/Express Proxy** : création d’une couche intermédiaire pour agréger et normaliser les appels aux API externes (FR24, AWVX), avec gestion des quotas et cache Redis pour minimiser latence et coûts.
* **ngx-translate & i18n** : stratégie de localisation multilingue, support des formats ICU, détection automatique de la langue système et bascule manuelle via le menu utilisateur.
* **CI/CD via GitHub Actions** : pipelines parallèles pour linting (ESLint, Stylelint), tests unitaires et de couverture, builds web et natives (iOS, Android) et déploiement sur Firebase Hosting et App Store Connect/Google Play.
* **Capacitor : Plugin Ecosystem** : utilisation de plugins natifs (Geolocation, Network, Storage, Push Notifications), développement de plugins custom pour intégrer des SDK spécifiques (cartographie, analytics).
* **Monitoring & Analytics** : intégration de Sentry pour le suivi des erreurs, Google Analytics for Firebase pour les événements utilisateurs, et des dashboards Grafana connectés à Prometheus pour la métrique serveur.

**1.4 Vue d’ensemble de l’architecture**

Le schéma ci‑dessous présente la topologie globale et les flux de données :

+---------------------------------------------------------------+

| Client Ionic/Angular |

|---------------------------------------------------------------|

| • UI Standalone Components |

| • Services Angular (HTTP, Auth, Geo) |

| • Plugins Capacitor (LocalStorage, Geolocation, Push) |

| • Cache IndexedDB pour offline-first |

+---------------------------------------------------------------+

| |

v v

+---------------------------+ +---------------------------+

| Firebase Auth | | Firestore DB |

| (OAuth, Email, SSO) | | (Users, Trips, Chats…) |

+---------------------------+ +---------------------------+

^ |

| v

+---------------------+ +---------------------------+

| Flight Data Proxy | <---- | External APIs (FR24, AWVX)|

| (NestJS/Express) | | • Rate Limit Handling |

| • Redis Cache | | • JSON Standardization |

+---------------------+ +---------------------------+

| |

v v

+---------------------------+ +---------------------------+

| Monitoring & Logs | | Analytics Firebase |

| (Sentry, Prometheus) | | (Events, Conversions) |

+---------------------------+ +---------------------------+

**Points clés de l’architecture :**

* **Décorrélation** : le proxy isole le client des contraintes CORS et gère la mise en cache pour optimiser le débit.
* **Résilience** : fallback vers données locales en cas de perte de connexion, avec stratégie de synchronisation différée.
* **Sécurité** : Firestore Rules couvrent l’accès aux données, CORS configuré au plus restrictif, authentification multi‑factorielle en option.
* **Scalabilité** : sans serveur dédié pour l’auth et le stockage, on s’appuie sur l’infrastructure Firebase, capable de gérer des pics d’usage sans intervention manuelle.

**Chapitre 2 : Architecture technique et choix de stack**

Ce chapitre explore en profondeur la structure interne de l’application BOB, en détaillant les principes directeurs à l’origine de l’architecture, le découpage des modules, les technologies sélectionnées ainsi que les patterns et bonnes pratiques à respecter pour garantir la qualité, la maintenabilité et la performance du code.

**2.1 Principes directeurs**

Pour construire une application robuste, résiliente et évolutive, nous nous appuyons sur plusieurs principes fondamentaux :

1. **Séparation des responsabilités (SoC)** : chaque couche est clairement isolée :
   * **UI Layer** : gère uniquement l’affichage et les interactions utilisateur (composants, templates, styles).
   * **Business Logic Layer** : contient les règles métier et les services qui orchestrent la logique.
   * **Data Layer** : responsable des opérations de persistance, de cache et des appels externes.
2. **Modularité et réutilisabilité** :
   * **Angular Standalone Components** pour chaque composant, réduisant les dépendances et améliorant le tree‑shaking.
   * **Modules fonctionnels** chargés en lazy loading afin de minimiser le bundle initial.
   * **Bibliothèque interne** (shared library) réunissant les composants et utilitaires réutilisables entre projets.
3. **Domain Driven Design (DDD)** :
   * Organisation du code autour des domaines clés : Authentification, Gestion de vol, Voyage, Réseaux sociaux, Administration.
   * **Entités**, **Value Objects** et **Cas d’usage** (Use Cases) modélisés explicitement pour séparer les règles métier.
4. **Offline First & Synchronisation** :
   * **Cache local** via IndexedDB (Angular PWA) ou Capacitor Storage pour garantir l’accès aux fonctionnalités critiques sans réseau.
   * **Stratégie de synchronisation** différée permettant d’envoyer les mises à jour au serveur dès que la connexion est rétablie.
5. **Testabilité et qualité du code** :
   * Couverture supérieure à 80 % avec tests unitaires (Jest/Karma) et tests E2E (Cypress).
   * CI intégrant des vérifications automatiques de lint, format, tests et rapport de couverture.
   * Usage de **Turing analysis** (SAST/sonar) pour détecter vulnérabilités et code smells.
6. **Scalabilité et résilience** :
   * Utilisation de Firebase et de services serverless pour absorber les pics de trafic.
   * Proxy Flight Data dimensionnable horizontalement, avec mise en cache Redis et gestion de back‑off.
7. **Sécurité** :
   * Règles Firestore strictes basées sur les rôles.
   * CORS configuré au plus restreint, pratiques OWASP pour Angular et NodeJS.
   * MFA en option pour l’authentification critique.

**2.2 Hiérarchie et découpage des modules**

L’organisation des dossiers et modules suit les principes ci‑dessus. Chaque ensemble correspond à un domaine fonctionnel ou à une couche technique.

**2.2.1 Core Module**

Contient les services et utilitaires globaux disponibles dans toute l’application :

* **AuthService** : gestion de la session, login/logout, rafraîchissement automatique des tokens Firebase.
* **LoggerService** : envoi structuré des logs vers Sentry, stratification des niveaux (info, warn, error).
* **StorageService** : abstraction au-dessus de LocalStorage, IndexedDB et Capacitor Storage, avec fallback automatique.
* **Interceptors HTTP** :
  + Ajout automatique du header Authorization avec JWT.
  + Gestion des erreurs globales (401, 500) et affichage d’alerts standardisées.
  + Retry configurable en cas d’échecs transitoires.
* **Guards Angular** :
  + AuthGuard pour protéger les routes nécessitant une authentification.
  + RoleGuard pour limiter l’accès selon les rôles (admin, user, guest).

**2.2.2 Shared Module**

Regroupe les briques génériques réutilisables :

* **UI Components** :
  + Boutons (ButtonPrimary, ButtonOutline), cartes (CardInfo), modals (ModalConfirm, ModalInfo).
  + Champs de formulaire customisés (date picker, time picker, autocomplete).
* **Directives & Pipes** :
  + FormatDatePipe pour formater les dates selon le locale.
  + AutoFocusDirective pour placer le focus sur un input lors de l’ouverture d’un modal.
* **Assets globaux** :
  + Variables SCSS (\_variables.scss) pour thèmes, couleurs, typographie.
  + Mixins (\_mixins.scss) pour grilles CSS, breakpoint responsives.

**2.2.3 Feature Modules**

Chaque feature est isolée en module et en domaine :

|  |  |
| --- | --- |
| **Module** | **Contenu principal** |
| **FlightModule** | Composants de suivi de vol (cartographie, dashboard altimétrique, timeline), FlightService pour consommer le Proxy. |
| **TripModule** | Pages de création, édition et visualisation de voyages, TripService pour Firestore, gestion des POI et des notes. |
| **ChatModule** | Interfaces de chat, ChatService utilisant Firestore en temps réel ou WebSockets, gestion des groupes et canaux. |
| **GameModule** | Moteurs de mini-jeux (Quiz, 4096, Puzzle), GameService pour scores, persistance des meilleurs scores en local. |
| **AdminModule** | Dashboard d’administration (statistiques, modération), AdminService pour gestion Firestore et règles d’accès. |

Chaque Feature Module inclut :

* Un **Routing Module** dédié (<feature>-routing.module.ts) définissant les routes enfants et les guards.
* Un **Module File** (<feature>.module.ts) important pour l’import de dépendances uniquement nécessaires à la feature.
* Un **Fichier de Styles** propre (optionnel) pour la customisation locale.

**2.2.4 App Module**

Le AppModule constitue le point d’entrée, orchestrant :

* L’import des modules CoreModule et SharedModule en forRoot() si nécessaire.
* La configuration du RouterModule avec PreloadAllModules pour précharger les modules les plus utilisés.
* La déclaration des providers globaux (TranslateLoader, ErrorHandler).
* L’intégration des plugins Capacitor au bootstrap.

**2.3 Choix technologiques détaillés**

|  |  |  |  |
| --- | --- | --- | --- |
| Couche | Technologie | Version ciblée | Avantages clés |
| UI | Ionic Framework | 6.x | Composants natifs iOS/Android, thèmes adaptatifs, animations fluides |
| Front-end | Angular | 16 (Standalone) | Bundle optimisé, CLI riche, RxJS intégré, support long terme |
| Back-end Proxy | NestJS (avec Express) | 10.x | DI, Guards, Interceptors, WebSockets, facilité de tests unitaires et e2e |
| Base de données | Firebase Firestore | Platform stable | Realtime sync, offline caching, sécurité via rules, scalabilité automatique |
| Authentification | Firebase Auth | Latest | OAuth, Email/Password, SSO Google/Apple, MFA, gestion des tokens |
| Packaging mobile | Capacitor | 6.x | Accès API natives, gestion plugins, recettage simplifié |
| CI/CD | GitHub Actions | - | Versionnable, Marketplace d’actions, parallélisme, matrices de jobs |
| i18n & l10n | ngx-translate | 14.x | Format ICU, pluralisation, chargement dynamique, hot reload |
| Monitoring | Sentry (JS & NodeJS) | 7.x | Trace des erreurs, performance monitoring, alerts configurables |
| Cache du Proxy | Redis | 7.x | Mise en cache LRU, expiration TTL, réduction des appels vers API externes |

**2.4 Patterns et bonnes pratiques de développement**

1. **Facade Pattern** :
   * Création de facades (FlightFacade, TripFacade) regroupant appels API, cache et logging sous une interface unique.
2. **Reactive Programming** :
   * Usage intensif de RxJS (Observable, Subject, BehaviorSubject) pour gérer les flux de données en temps réel.
   * Évitement des fuites de mémoire avec takeUntil dans les subscriptions.
   * Combinaison de switchMap, mergeMap et shareReplay pour optimiser les appels réseau.
3. **Lazy Loading & Preloading Strategy** :
   * Configuration de PreloadAllModules pour précharger en arrière-plan.
   * Découpage de gros modules en sous-modules lazy afin de réduire le temps de boot.
4. **Immutabilité et State Management** :
   * Encouragement de patterns immuables pour les entités (utilisation de readonly, Record, structures gelées).
   * Option d’intégration de NgRx ou Akita si les états deviennent trop complexes.
5. **Clean Code et Standardisation** :
   * ESLint et Prettier pour garantir la cohérence du style.
   * Convention de nommage SCREAMING\_SNAKE pour les constantes et UPPER\_CAMEL pour les classes.
   * Documentation inline via TSDoc et génération automatique via Compodoc.
6. **Sécurité et Performance** :
   * Configuration stricte de CSP (Content Security Policy) et sanitization des entrées utilisateur.
   * Lazy loading des images (ion-img), compression WebP, utilisation de virtual scroll pour les listes longues.

**2.5 Organisation des assets, styles et internationalisation**

* **SCSS** :
  + Dossiers : src/app/shared/styles/ pour global, src/app/features/\*/styles/ pour local.
  + Fichiers clés :
    - \_variables.scss : couleurs, typographies, espacements.
    - \_mixins.scss : mixins responsive, grid helper, clearfix.
    - \_animations.scss : variables de timing, fonctions ease-in/out.
* **Assets** :
  + assets/images/icons/, assets/images/backgrounds/, assets/images/markers/.
  + **Optimisation** : génération automatisée de sprites, lazy load, srcset pour supports Retina.
* **Internationalisation (i18n)** :
  + Structure des fichiers : assets/i18n/en.json, fr.json, es.json, etc.
  + **Organisation des clés** : modulaires (<feature>.<component>.<key>).
  + **Chargement** : pré-chargement des principales langues, chargement à la demande des secondaires.
  + **Outils** : script de validation de clés manquantes, extraction automatique via ngx-translate-extract.

**Chapitre 3 : Structure du projet et conventions de nommage**

Ce chapitre présente en détail l’organisation physique et logique de l’ensemble du code source de l’application BOB, ainsi que les conventions de nommage et les bonnes pratiques à respecter pour garantir la cohérence, la lisibilité et la maintenabilité du projet. Il est essentiel que chaque contributeur s’aligne sur ces règles pour faciliter la collaboration, la revue de code et l’évolution du code base.

**3.1 Arborescence générale du projet**

Le répertoire src/ constitue le cœur de l’application et est organisé selon une séparation claire des responsabilités :

src/

├── app/ # Code applicatif principal

│ ├── core/ # Services globaux, interceptors, guards, interfaces de base

│ ├── shared/ # Composants, directives, pipes réutilisables, types et utilitaires génériques

│ ├── features/ # Modules métier (lazy loaded) regroupant pages, composants, services

│ │ ├── flight/ # Suivi de vol et données géospatiales

│ │ │ ├── pages/ # Pages Angular (Home, Map, Details, History)

│ │ │ ├── components/ # Composants spécifiques (FlightMap, AltitudeChart, FlightInfoCard)

│ │ │ ├── services/ # FlightService, DataTransformers, NotificationService

│ │ │ ├── models/ # Entités TypeScript, interfaces et value objects

│ │ │ ├── styles/ # SCSS locaux, thèmes spécifiques

│ │ │ ├── flight.module.ts

│ │ │ └── flight-routing.module.ts

│ │ ├── trip/ # Gestion de voyages, itinéraires et points d’intérêt

│ │ ├── chat/ # Messagerie en temps réel et historique des conversations

│ │ ├── game/ # Mini-jeux embarqués (quiz, puzzles, leaderboard)

│ │ └── admin/ # Interface d’administration et modération du contenu

│ ├── assets/ # Ressources statiques référencées par l’application

│ │ ├── images/ # Icons, backgrounds, markers (formats WebP, SVG optimisés)

│ │ ├── i18n/ # Fichiers de traduction JSON organisés par langue et module

│ │ └── styles/ # Styles globaux (themes, mixins, fonctions utils)

│ ├── environments/ # Configurations d’environnement (development, staging, production)

│ ├── app.module.ts # Module racine de l’application (import Core, Shared, AppRouting)

│ └── app.routes.ts # Définition des routes principales et stratégie de preloading

├── assets/ # Fichiers statiques non transformés (favicon, meta tags)

├── environments/ # Configurations globales pour CI/CD et scripts de déploiement

├── capacitor.config.ts # Paramètres Capacitor pour les plateformes mobiles

├── ionic.config.json # Configuration du CLI Ionic (capacitor, live reload)

├── angular.json # Architect configuration pour Angular CLI (build, serve, test)

├── package.json # Déclarations de dépendances, scripts de build, lint et tests

├── tsconfig.json # Configuration TypeScript (strict mode, alias de chemins)

├── .eslintrc.json # Règles ESLint pour la cohérence du code

├── prettier.config.js # Configuration Prettier pour le formatage automatique

└── README.md # Présentation du projet, guide d’installation et de contribution

**3.1.1 Dossier app/**

* Centralise tout le code Angular et Ionic.
* Sépare les éléments globaux (core, shared) des fonctionnalités métier (features).

**3.1.2 Dossier assets/**

* Contient des ressources statiques qui ne sont pas transformées par Webpack.
* Sépare les images, traductions et styles pour une gestion claire et versionnée.

**3.1.3 Fichiers racine**

* **angular.json / tsconfig.json** : configurent la compilation TypeScript et les builds.
* **package.json** : inclut les scripts clés :
  + npm run start: lancement en dev.
  + npm run build: compilation prod.
  + npm run lint: vérification du style de code.
  + npm run test: exécution des tests unitaires et E2E.
* **capacitor.config.ts / ionic.config.json** : définissent les paramètres de packaging mobile.

**3.2 Conventions de nommage**

Des règles strictes garantissent la clarté des imports et la découverte rapide des fichiers :

**3.2.1 Composants Angular**

|  |  |  |
| --- | --- | --- |
| **Catégorie** | **Convention** | **Exemple** |
| Composant simple | <feature>-<component>.component.ts | flight-map.component.ts |
| Composant page | <feature>-<page>.page.ts | chat-conversation.page.ts |
| Composant dialogue | <feature>-<name>-dialog.component.ts | trip-edit-dialog.component.ts |

* Les **templates HTML** utilisent <feature>-<component>.component.html.
* Les **styles SCSS** associés suivent le même préfixe.

**3.2.2 Services**

|  |  |  |
| --- | --- | --- |
| Type | Convention | Exemple |
| Service domaine | <feature>.service.ts | flight-data.service.ts |
| Service de façade | <feature>-facade.service.ts | trip-facade.service.ts |
| Service utilitaire | <name>.util.service.ts | date-format.util.service.ts |

**3.2.3 Modules et routing**

|  |  |  |
| --- | --- | --- |
| Élément | Convention | Exemple |
| Module | <feature>.module.ts | admin.module.ts |
| Routing | <feature>-routing.module.ts | game-routing.module.ts |

**3.2.4 Models, interfaces et enums**

|  |  |  |
| --- | --- | --- |
| Élément | Convention | Exemple |
| Interface | I<Feature><Entity>.ts | ITripDetails.ts |
| Type alias | <feature><Name>Type.ts | chatMessageType.ts |
| Enum | <Name>Enum.ts | UserRoleEnum.ts |

**3.2.5 Pipes et directives**

|  |  |  |
| --- | --- | --- |
| Élément | Convention | Exemple |
| Pipe | <feature>-<name>.pipe.ts | flight-duration.pipe.ts |
| Directive | <name>.directive.ts | click-outside.directive.ts |

**3.2.6 Styles SCSS**

* **Global** : \_variables.scss, \_mixins.scss, \_functions.scss dans src/app/assets/styles/.
* **Local** : <feature>-<component>.component.scss à côté des composants.
* **Themes** : theme-light.scss, theme-dark.scss chargés via variables.scss.

**3.2.7 Tests**

|  |  |  |
| --- | --- | --- |
| Test | Convention | Exemple |
| Unitaire | <file-to-test>.spec.ts | flight-data.service.spec.ts |
| E2E | <feature>-<scenario>.e2e-spec.ts | trip-creation.e2e-spec.ts |

**3.3 Dossiers clés détaillés**

**3.3.1 Dossier core/**

Le dossier core/ héberge les éléments transverses qui s’appliquent à l’ensemble de l’application :

* **Services globaux** :
  + AuthService : gestion avancée de l’auth (tokens, MFA, SSO).
  + LoggerService : intégration Sentry, niveaux de log personnalisés.
  + NetworkService : détection de la connectivité et fallback offline.
  + StorageService : abstraction et fallback entre LocalStorage, IndexedDB, Capacitor Storage.
* **Interceptors HTTP** :
  + JwtInterceptor : injection automatique du token.
  + ErrorInterceptor : traitement global des erreurs et notifications.
* **Guards & Resolvers** :
  + AuthGuard, RoleGuard, UnsavedChangesGuard, DataResolver.
* **Interfaces et modèles de base** :
  + BaseEntity, ApiResponse<T> pour uniformiser le traitement des réponses.

**3.3.2 Dossier shared/**

Ce dossier contient toutes les briques réutilisables et indépendantes de tout domaine métier :

* **UI Components** :
  + ButtonPrimary, ButtonSecondary, CardInfo, SkeletonLoader, AvatarComponent.
* **Pipes** :
  + FormatDatePipe, TruncateTextPipe, SafeHtmlPipe.
* **Directives** :
  + AutoFocusDirective, ClickOutsideDirective, LazyLoadImageDirective.
* **Types utilitaires** :
  + PaginatedResponse<T>, Option<T>, SelectOption.
* **Helpers** :
  + Fonctions utilitaires TS dans utils/ (formatters, validators).

**3.3.3 Dossier features/**

Chaque fonctionnalité est isolée dans son propre module pour faciliter l’évolution et le déploiement progressif :

src/app/features/<feature>/

├── pages/ # Pages Angular (component.ts + HTML + SCSS)

├── components/ # Widgets réutilisables au sein du module

├── services/ # Services de domaine (Data, Facade)

├── models/ # Interfaces, enums, value objects

├── styles/ # Styles spécifiques à la feature

├── <feature>.module.ts # Déclaration du module et imports nécessaires

├── <feature>-routing.module.ts # Définition des routes et guards associés

└── index.ts # Exports publics pour faciliter l’import externes

* **TripModule** : gère la création, l’édition, la suppression et l’affichage des trips.
* **ChatModule** : encapsule la logique de messagerie, les threads et la synchronisation.
* **GameModule** : intègre plusieurs jeux : quiz, puzzles, et tableau de scores.

**3.3.4 Dossier assets/**

Ce répertoire contient toutes les ressources non compilées :

* **images/** : organisation en sous-dossiers (icons/, backgrounds/, markers/), optimisées pour le web et le mobile.
* **i18n/** : fichiers JSON nommés par code langue, structurés par feature (ex. flight.json, chat.json).
* **styles/** : polices (fonts/), thèmes, animations Lottie, sprites.
* **docs/** : guides PDF, maquettes Figma exportées, chartes graphiques.

**3.4 Recommandations pour les contributeurs**

1. **Adhérer aux conventions** de nommage et d’architecture pour tout nouveau code.
2. **Placer les tests** (unitaires et E2E) dans un dossier \_\_tests\_\_ ou juste à côté du fichier testé.
3. **Mettre à jour la documentation** (README, CHANGELOG, Compodoc) pour chaque fonctionnalité majeure.
4. **Effectuer une Pull Request** claire, avec description précise, capture d’écran et checklist de test.
5. **Utiliser les alias de chemins** (@core, @shared, @features) pour simplifier et uniformiser les imports.
6. **Vérifier la performance** et l’accessibilité : light DOM, ARIA labels, tests Lighthouse.
7. **Respecter la sécurité** : ne jamais exposer de clés dans le code, utiliser les Firestore Rules et CSP.

**Chapitre 4 : Description détaillée des modules, services et pages principales**

Ce chapitre offre une description exhaustive de chaque **Feature Module**, de leurs **services**, **composants**, **pages**, **modèles** et **patterns** associés. Il vise à accompagner les développeurs dans la compréhension fine de la structure, des interactions, des flux de données et des tests à mettre en place pour chaque fonctionnalité majeure.

**4.1 FlightModule**

**Objectif :** proposer un suivi de vol complet et immersif, combinant cartographie interactive, données altimétriques, timeline des phases, alertes de proximité et analytics de vol.

**4.1.1 Structure du module**

src/app/features/flight/

├── pages/ # Pages Angular pour l’interface principale

│ ├── flight-home.page.\* # Accueil du suivi de vol (résumé, notifications)

│ ├── flight-map.page.\* # Carte en plein écran avec trajectoire et waypoints

│ ├── flight-altitude.page.\* # Graphique altimétrique interactif

│ ├── flight-timeline.page.\* # Timeline des phases (boarding, cruise, descent)

│ ├── flight-alerts.page.\* # Notifications et alertes (turbulence, arrivées)

├── components/ # Composants réutilisables internes

│ ├── flight-map.component.\* # Wrapper de la carte avec clustering

│ ├── altitude-chart.component.\* # Graph linéaire custom avec zoom et tooltips

│ ├── flight-info-card.component.\* # Cards résumant vitesse, altitude, distance

│ ├── flight-alert.component.\* # UI pour une alerte spécifique

│ └── flight-status-badge.component.\*# Badge d’état (on time, delayed)

├── services/ # Services et façades métier

│ ├── flight.service.ts # Appels HTTP au proxy, gestion du cache et des tokens

│ ├── flight-facade.service.ts # Interface simplifiée combinant plusieurs services

│ ├── flight-data.transformer.ts # Conversion pure des payloads externes

│ ├── flight-alert.service.ts # Détection des événements critiques et dispatch d’alertes

│ └── flight-metrics.service.ts # Collecte et envoi d’événements analytics

├── models/ # Types, interfaces et enums

│ ├── flight.model.ts # IFlight, IPosition, IAltitudePoint

│ ├── flight-state.enum.ts # boarding, taxi, takeoff, cruise, descent, landed

│ ├── flight-event.enum.ts # turbulence, altitudeChange, waypointReached

│ └── flight-metrics.interface.ts # IMetricsEvent, IAnalyticsPayload

├── styles/ # Styles SCSS dédiés au module

│ └── flight-theme.scss # Variables couleurs, mixins, animations spécifiques

├── flight.module.ts # Déclaration, imports, providers spécifiques

└── flight-routing.module.ts # Routes (home, map, altitude, timeline, alerts)

**4.1.2 Services détaillés**

* **FlightService** :
  + **getFlightStatus()** : HTTP GET pour le statut, met en cache la réponse pendant 30s.
  + **streamPositionUpdates()** : renvoie un Observable<IPosition> en push (WebSocket ou Firestore).
  + **getAltitudeHistory(range: TimeRange)** : GET historique, applique un pipeline RxJS (debounceTime, map).
  + **getNearbyAirports()** : appel optionnel pour afficher les aéroports proches, caching local.
* **FlightFacadeService** :
  + **initializeFlight(flightId: string)** : orchestration de l’auth, du chargement initial, du prefetch.
  + **subscribeToUpdates()** : expose un BehaviorSubject<AppFlightState> pour l’état global.
  + **refreshData(force: boolean)** : forcage de rafraîchissement manuel ou périodique.
* **FlightAlertService** :
  + Détecte les événements (turbulence, seuils d’altitude) via combineLatest de streams de données.
  + Émet des notifications push locales avec **LocalNotifications** de Capacitor.
* **FlightMetricsService** :
  + Enregistre les événements utilisateur (navigation, refresh) et les performances (temps de chargement) via Google Analytics.
  + Fournit un strap pour les tests A/B sur différentes vues de carte.

**4.1.3 Pages et composants approfondis**

* **flight-home.page** :
  + **Layout** : header avec l’état, section cartes miniatures, section notifications.
  + **Interactions** : scroll infini pour historique des statuts récents, swipe horizontal pour changer de module.
* **flight-map.page** :
  + **Plugins** : @capacitor/google-maps, clustering marker, custom overlays (weather icons).
  + **Performance** : ion-virtual-scroll pour la liste de waypoints, ChangeDetectionStrategy.OnPush.
* **flight-altitude.page** :
  + **Librairie** : ngx-charts-line-chart, configurations dynamiques, export PNG.
  + **UX** : zoom avec pinch-to-zoom, tooltip show on long-press.
* **flight-timeline.page** :
  + **UI** : timeline horizontale scrollable, responsive, animations via @angular/animations.
  + **Accessibilité** : ARIA roles, focus management.
* **flight-alerts.page** :
  + Listing des alertes, possibilité de marquer comme lues, filtrage par type.

**4.2 TripModule**

**Objectif :** gérer la création, la modification et la visualisation des voyages (trips), avec support de médias, géocodage et partage.

**4.2.1 Structure du module**

src/app/features/trip/

├── pages/

│ ├── trip-list.page.\* # Liste des trips avec filtrage avancé

│ ├── trip-detail.page.\* # Vue détaillée avec cartes, notes, galerie

│ ├── trip-edit.page.\* # Formulaire dynamique + validation avancée

│ ├── trip-map.page.\* # Vue carte fullscreen dédiée

│ └── trip-share.page.\* # Interface de partage (réseaux sociaux, email)

├── components/

│ ├── trip-card.component.\* # Rendu résumé pour chaque trip

│ ├── poi-list.component.\* # Liste des POI avec drag-and-drop

│ ├── photo-gallery.component.\* # Galerie optimisée, lazy load des images

│ └── trip-form.component.\* # Sous-composant du formulaire d’édition

├── services/

│ ├── trip.service.ts # CRUD Firestore, transactions multi-documents

│ ├── trip-facade.service.ts # Orchestration business, validation DDD

│ ├── photo-upload.service.ts # Upload avec gestion d’erreurs et de compression

│ ├── geocoding.service.ts # Auto-complete et reverse-geocoding

│ └── sharing.service.ts # Préparation des liens et interactions natives

├── models/

│ ├── trip.model.ts # ITrip, IUserTripLink

│ ├── poi.model.ts # IPointOfInterest, IGeoLocation

│ ├── trip-status.enum.ts # planned, active, completed, archived

│ └── share-options.interface.ts # IShareOptions (platforms, message template)

├── styles/

│ └── trip-theme.scss # Variables SCSS (couleurs, typographie, espacement)

├── trip.module.ts # Import des dépendances et exports publics

└── trip-routing.module.ts # Definition des routes et guards (AuthGuard, OwnerGuard)

**4.2.2 Services détaillés**

* **TripService** :
  + **createTrip(trip: ITrip)**, **updateTrip(id, changes)**, **deleteTrip(id)**.
  + **getTrips(userId, queryOpts)** : utilise collectionGroup pour requêtes multi-utilisateurs.
  + Transaction supportée via runTransaction pour op. atomiques.
* **TripFacadeService** :
  + **loadTrips(params)** : combine filtre, pagination et caching.
  + **getTripDetail(tripId)** : orchestration de l’appel Firestore et préchargement des images.
* **PhotoUploadService** :
  + Convertit l’image en WebP, redimensionne avant upload, expose uploadProgress$.
  + Gestion locale du cache des images et purge périodique.
* **GeocodingService** :
  + **autocomplete(query)** : debounceTime(300ms), distinctUntilChanged, switchMap vers API.
  + **reverseGeocode(coords)** : mapping précis des coordonnées en adresse.
* **SharingService** :
  + Crée des liens dynamiques via Firebase Dynamic Links.
  + Intègre les partages natifs avec **Share API** ou plugins Capacitor.

**4.2.3 Pages et composants approfondis**

* **trip-list.page** :
  + **UI** : barre de recherche, filtre par date/status, tri par distance.
  + **Performance** : ion-virtual-scroll, OnPush, caching local.
* **trip-detail.page** :
  + **Sections** : carte interactive, galerie, notes, box de partage.
  + **Interactions** : pinch-to-zoom, slide carousel, lazy loading.
  + **Sécurité** : masquer les données privées selon les règles Firestore.
* **trip-edit.page** :
  + **Form** : Reactive Forms avec **FormArray** pour POI, validateurs custom asynchrones.
  + **UX** : sauvegarde automatique partielle, draft mode.
* **trip-share.page** :
  + **Options** : partager par email, SMS, réseaux sociaux.
  + **Prévisualisation** : vignette du trip et message template.

**4.3 ChatModule**

**Objectif :** fournir une messagerie en temps réel, modérable, avec support d’images, emojis, notifications push et archivage.

**4.3.1 Structure du module**

src/app/features/chat/

├── pages/

│ ├── chat-list.page.\* # Liste des conversations et canaux publics/privés

│ ├── chat-room.page.\* # Vue d’une conversation, messages et input composer

│ ├── chat-settings.page.\* # Paramètres du chat (notifications, blocklist)

├── components/

│ ├── message-item.component.\* # Affichage d’un message (texte, image, emoji)

│ ├── message-list.component.\* # Liste virtualisée des messages

│ ├── chat-input.component.\* # Zone de saisie avec attachments et emojis picker

│ └── typing-indicator.component.\* # Affiche qui est en train d’écrire

├── services/

│ ├── chat.service.ts # Firestore/WebSocket, gestion des rooms et permissions

│ ├── chat-facade.service.ts # Orchestration, modération, caching partiel

│ ├── emoji.service.ts # Chargement et gestion des emojis custom

│ └── notifications.service.ts # Gestion des notifications push locales et serveur

├── models/

│ ├── chat-message.model.ts # IMessage, IUserReference, IMediaAttachment

│ ├── chat-room.model.ts # IChatRoom, IParticipant

│ └── chat-event.enum.ts # messageSent, userJoined, userLeft, messageEdited

├── styles/

│ └── chat-theme.scss # Couleurs, polices, espacements spécifiques

├── chat.module.ts # Import et export des dépendances chat

└── chat-routing.module.ts # Routes (list, room/:id, settings)

**4.3.2 Points clés**

* **Realtime** : utilisation de Firestore snapshots et onSnapshot pour synchro rapide.
* **Scalabilité** : pagination infinie, limitToLast, indexation des collections.
* **Sécurité** : règles Firestore pour modération, blocage, suppression.
* **UX** : typing indicators, read receipts, messages épinglés.
* **Notifications** : push via Firebase Cloud Messaging et Capacitor PushNotifications plugin.

**4.4 GameModule**

**Objectif :** divertir les passagers avec des mini-jeux embarqués, avec score, leaderboard et options premium.

**4.4.1 Structure du module**

src/app/features/game/

├── pages/

│ ├── game-list.page.\* # Sélection des jeux disponibles

│ ├── game-4096.page.\* # Puzzle 4096 intégration complète

│ ├── game-quiz.page.\* # Quiz géo basé sur la carte du vol

│ └── game-leaderboard.page.\* # Classement global et par vol

├── components/

│ ├── game-board.component.\* # Plateau de jeu abstrait pour puzzles

│ ├── quiz-question.component.\* # Interface question/réponse

│ ├── score-display.component.\* # Affichage du score et historique

│ └── premium-banner.component.\* # Promotion de fonctionnalités payantes

├── services/

│ ├── game.service.ts # Logique métier des jeux, state management

│ ├── leaderboard.service.ts # Stockage et récupération des scores

│ ├── quiz.service.ts # Chargement dynamique des questions via API

│ └── purchase.service.ts # Gestion des achats in-app via Capacitor InAppPurchase

├── models/

│ ├── game-state.interface.ts # IGameState, IScoreEntry

│ ├── quiz-model.ts # IQuizQuestion, IQuizAnswer

│ └── purchase-status.enum.ts # purchased, pending, failed

├── styles/

│ └── game-theme.scss # Animations, transitions, palette spécifique

├── game.module.ts # Déclaration du module et routes children

└── game-routing.module.ts # Routage interne avec guards premium

**4.4.2 Particularités**

* **State Management** : usage de BehaviorSubject pour chaque jeu, sauvegarde automatique du state local.
* **Performance** : Web Workers pour calculs intensifs (4096), NgZone.runOutsideAngular pour animations.
* **Monétisation** : gated features via purchase.service, affichage conditionnel des bannières.
* **Leaderboard** : stockage en Firestore, règles de sécurité pour éviter triche.

**4.5 AdminModule**

**Objectif :** fournir un back-office embarqué pour modération, gestion des contenus, analyse des données et déploiement de sondages.

**4.5.1 Structure du module**

src/app/features/admin/

├── pages/

│ ├── admin-dashboard.page.\* # Vue d’ensemble KPIs (users, trips, chats, games)

│ ├── user-management.page.\* # CRUD utilisateurs, rôles, statuts

│ ├── content-moderation.page.\* # Modération des messages et uploads

│ └── polls.page.\* # Création et suivi de sondages en vol

├── components/

│ ├── kpi-card.component.\* # Affichage d’une métrique clé

│ ├── user-table.component.\* # Table interactive avec filtres et pagination

│ ├── moderation-item.component.\* # UI pour un élément à modérer

│ └── poll-creator.component.\* # Formulaire de création de sondage

├── services/

│ ├── admin.service.ts # Appels Firestore et fonctions Cloud (moderation)

│ ├── kpi.service.ts # Récupération de métriques via Cloud Functions

│ ├── user.service.ts # Gestion avancée des rôles et permissions

│ └── poll.service.ts # CRUD sondages, collecte de réponses

├── models/

│ ├── user.model.ts # IUser, IUserRoleEnum

│ ├── poll.model.ts # IPoll, IPollResponse

│ └── moderation-event.enum.ts # approve, reject, flag

├── styles/

│ └── admin-theme.scss # Palette, typographie et spacing du back-office

├── admin.module.ts # Import des dépendances admin et providers

└── admin-routing.module.ts # Routes sécurisées (AdminGuard, RoleGuard)

**4.5.2 Fonctionnalités clés**

* **Sécurité renforcée** : AdminGuard, RoleGuard exécutés avant l’accès aux pages.
* **Analytique** : dashboards temps réel via chart.js ou recharts.
* **Modération** : actions batch, log des décisions via Cloud Functions.
* **Sondages** : distribution en vol, collecte instantanée, export CSV.

**Chapitre 5 : Flux de navigation et UI/UX globale**

Ce chapitre décrit en détail la **navigation** et l’**expérience utilisateur** (UX) de l’application BOB, en couvrant la configuration du **routing** Angular, la structure des vues, les **gestures**, les **animations**, la **conception responsive**, l’**accessibilité**, ainsi que les mécanismes de **notification** et de **feedback** pour assurer un parcours utilisateur fluide, performant et agréable.

**5.1 Zones de navigation et arborescence des routes**

L’application est segmentée en trois espaces principaux, chaque zone disposant de son propre layout et de ses guards :

1. **Zone non authentifiée** :
   * **Routes** : /auth/login, /auth/signup, /auth/forgot-password, /auth/reset/:token.
   * **Pages** : composant de connexion (LoginPage), d’inscription (SignupPage), de récupération de mot de passe (ForgotPasswordPage) et de réinitialisation.
   * **Layout** : écran centré avec card de formulaire, logo en haut, footer avec liens RGPD et CGU.
   * **Guard** : GuestGuard empêche l’accès aux pages d’auth lorsqu’une session est active, redirige vers /app/flight.
2. **Zone Utilisateur** :
   * **Tabbar principale** :
     + **Flight** (/app/flight)
     + **Trip** (/app/trip)
     + **Chat** (/app/chat)
     + **Game** (/app/game)
   * **Layout** : <ion-tabs> pour navigation persistante, <ion-header> avec bouton profil et notifications, <ion-footer> optionnel pour actions rapides.
   * **Guard** : AuthGuard vérifie presence d’un token Firebase valide.
   * **Sub-routes** : chaque tab utilise son propre RouterOutlet pour conserver l’état lors des switches.
3. **Zone Administrateur** :
   * **Menu latéral** : <ion-menu side="start" contentId="main"> contenant les liens vers Dashboard, Users, Content, Polls, Settings.
   * **Routes** : sous /app/admin/\*, lazy loading vers AdminModule.
   * **Guards** : AuthGuard + RoleGuard assurant uniquement aux utilisateurs admin l’accès.
   * **Layout** : interface desktop-like avec <ion-split-pane> sur tablettes, tables responsives, chart panels.

**5.2 Configuration avancée du RouterModule**

La configuration du RouterModule utilise :

* **Lazy loading** pour optimiser la taille initiale du bundle.
* **PreloadingStrategy** (PreloadAllModules ou custom) pour précharger les modules après le chargement initial.
* **Routes paramétrées** pour deep linking vers des entités spécifiques (ex. trip/detail/:id).

const routes: Routes = [

{

path: 'auth',

loadChildren: () => import('./auth/auth.module').then(m => m.AuthModule),

canActivate: [GuestGuard]

},

{

path: 'app',

component: AppShellComponent,

canActivate: [AuthGuard],

children: [

{

path: 'flight',

loadChildren: () => import('./features/flight/flight.module')

.then(m => m.FlightModule)

},

{

path: 'trip',

loadChildren: () => import('./features/trip/trip.module')

.then(m => m.TripModule)

},

{

path: 'chat',

loadChildren: () => import('./features/chat/chat.module')

.then(m => m.ChatModule)

},

{

path: 'game',

loadChildren: () => import('./features/game/game.module')

.then(m => m.GameModule)

},

{

path: 'admin',

loadChildren: () => import('./features/admin/admin.module')

.then(m => m.AdminModule),

canActivate: [RoleGuard]

},

{ path: '', redirectTo: 'flight', pathMatch: 'full' }

]

},

{ path: '', redirectTo: 'auth/login', pathMatch: 'full' }

];

@NgModule({

imports: [RouterModule.forRoot(routes, {

preloadingStrategy: PreloadAllModules,

scrollPositionRestoration: 'enabled',

anchorScrolling: 'enabled'

})],

exports: [RouterModule]

})

export class AppRoutingModule {}

**Astuce** : scrollPositionRestoration permet de mémoriser la position de défilement entre changements de routes.

**5.3 Gestures et interactions tactiles**

BOB exploite le **GestureController** d’Ionic pour offrir des interactions naturelles :

* **Swipe Tabs** :
  + Détection du mouvement horizontal pour changer d’onglet instantanément.
  + Seuil ajusté pour éviter les conflits avec ion-slides.
* **Pull-to-refresh** :
  + Composant <ion-refresher> placé en tête des listes longues (Trip, Chat), déclenche ionRefresh.
* **Zoom et Drag** :
  + Pinch-to-zoom sur cartes (flight-map, trip-map) via plugin **Capacitor Gestures**.
  + Drag-to-move des marqueurs POI dans le TripModule.
* **Long press** :
  + Ouverture de menus contextuels (ContextMenu) sur messages Chat et items de liste.

// Configuration d'un long press

const longPress = this.gestureCtrl.create({

el: element,

gestureName: 'long-press',

threshold: 0,

holdThreshold: 500,

onStart: () => this.openContextMenu()

});

longPress.enable(true);

**5.4 Animations et transitions poussées**

Pour rendre la navigation plus engageante et fluide :

* **Trigers Angular** :
  + fadeInRoute, slideInUp, zoomIn, définis globalement et appliqués sur <router-outlet>.
  + Transitions de liste (@listStagger) pour afficher les éléments un à un.
* **NavController d’Ionic** :
  + Méthodes navigateForward, navigateBack avec options de animationDirection.
* **Modals et Popovers** :
  + Paramètres enterAnimation et leaveAnimation custom pour transitions spécifiques.
  + Intégration de l’API **Animation** de Web Animations pour micro-animations.

// Exemple d'animation de modal

const modal = await this.modalController.create({

component: CustomModalComponent,

enterAnimation: modalEnter,

leaveAnimation: modalLeave

});

await modal.present();

**5.5 Design responsive et adaptatif**

La conception UI s’adapte à tous les écrans grâce :

* **SCSS Variables & Mixins** :
  + Breakpoints dans \_variables.scss, mixin respond-to pour media queries.
  + Utility mixins (flex-center, responsive-grid) pour layouts flexibles.
* **Split-Pane** :
  + <ion-split-pane> pour afficher simultanément la liste et le détail sur tablette/desktop.
* **Component Variants** :
  + ion-menu-toggle pour mode side-menu adaptatif.
  + ion-grid pour mise en page responsive des tableaux et dashboards.

**Note** : tester régulièrement avec Chrome DevTools Device Mode et outils comme BrowserStack.

**5.6 Accessibilité (a11y) et internationalisation (i18n)**

BOB respecte les standards WCAG 2.1 AA et prend en charge plusieurs langues :

* **Attributs ARIA** :
  + Utilisation de aria-label, role, aria-live pour annonces dynamiques.
  + Décoratifs marqués aria-hidden="true".
* **Navigation Keyboard** :
  + tabindex sur éléments interactifs, gestion du focus via FocusService.
  + Raccourcis claviers (ESC pour fermer modals).
* **Contraste et Mode sombre** :
  + Thèmes Light & Dark dans variables.scss, respect des ratios de contraste.
* **Internationalisation** :
  + ngx-translate avec fichiers JSON modulaires (en.json, fr.json).
  + Interpolation ICU pour pluralisation et genres.
  + Chargement lazy des fichiers de langue.

**5.7 Notifications, toasts, toasts persistants et feedback**

Pour informer et guider l’utilisateur de manière non intrusive :

* **Push Notifications** :
  + **Capacitor PushNotifications** + FCM, configuration des channels pour Android.
  + Gestion des permissions et deep linking depuis notification.
* **Toasts et Alerts** :
  + ion-toast pour retours courts, persistance configurable.
  + ion-alert pour confirmations critiques, choix multiples.
  + Personnalisation des styles (CSS variables, shadow DOM overrides).
* **Persistent Notifications** :
  + Section dédiée dans le layout pour messages système (offline, updates).
* **Haptic & Sound Feedback** :
  + **Haptics** plugin pour vibrations, **NativeAudio** pour sons courts.

// Example push registration

import { PushNotifications } from '@capacitor/push-notifications';

PushNotifications.register();

PushNotifications.addListener('registration', token => {

console.log('Push token: ', token.value);

});

PushNotifications.addListener('pushNotificationReceived', notification => {

this.handleNotification(notification);

});

**5.8 Suivi analytics et tests de navigation**

* **Analytics** :
  + **Firebase Analytics** pour tracking des écrans, événements custom (tab change, form submissions).
  + **Sentry** pour monitoring des performances de navigation et erreurs JS.
* **E2E Tests** :
  + **Cypress** scripts pour flows majeurs (login, create trip, chat send, game play).
  + Environnements headless pour CI (GitHub Actions).
* **Audit** :
  + **Lighthouse** CI pour performance, accessibilité, SEO.

**Chapitre 6 : Infrastructure Firebase & Capacitor**

Ce chapitre présente en détail l’infrastructure serverless, les configurations et les best practices pour déployer, maintenir et sécuriser l’application BOB. Il couvre :

1. La mise en œuvre de Firebase Authentication et l’intégration avancée avec Angular.
2. La modélisation, la sécurité, l’optimisation et la réplication de données dans Cloud Firestore.
3. Le stockage de médias avec Firebase Storage, incluant la transformation à la volée et la gestion des coûts.
4. La création et le déploiement de Cloud Functions, pour orchestrer les tâches backend et assurer la scalabilité.
5. L’hébergement web pour la PWA, la configuration de Service Workers et les optimisations SEO.
6. La configuration de Capacitor, l’installation/gestion des plugins et l’optimisation des builds natifs.
7. Les workflows CI/CD, la gestion de secrets, le monitoring, la résilience et la surveillance des coûts.

**6.1 Firebase Authentication**

**6.1.1 Stratégies d’identification**

* **Providers classiques** :
  + **Email/Password** : via createUserWithEmailAndPassword() avec vérification d’adresse (sendEmailVerification()).
  + **OAuth 2.0** : Google, Apple, Facebook, GitHub – configuration OAuth dans Google Cloud Console et Firebase.
* **Utilisateurs anonymes** :
  + Permettent un onboarding rapide, conversion transparente en compte permanent.
* **Multi-Factor Authentication (MFA)** :
  + Ajout d’un second facteur via SMS (Twilio) ou email.
  + Exemple : const mfaInfo = PhoneMultiFactorGenerator.assertion(phoneAuthCredential);
* **SSO Enterprise** :
  + Intégration SAML/OpenID Connect pour clients Corporate via Firebase Identity Platform.

**6.1.2 Flux, gestion des tokens et sécurité**

* **Cycle de vie du token** :
  + Storage sécurisé dans IndexedDB ou Capacitor Secure Storage.
  + Renouvellement automatique via onIdTokenChanged().
  + Gestion des erreurs : interceptor global pour code 401, redirection vers /auth/login.
* **Revocation et invalidation** :
  + auth().revokeRefreshTokens(uid) pour forcer la déconnexion sur tous les devices.
* **Segmentation des utilisateurs** :
  + Claims personnalisés : setCustomUserClaims(uid, {role: 'admin'}), exploités ensuite dans authState.
* **Audit et conformité** :
  + Activation des audit logs dans Google Cloud pour suivre chaque action critique.

**6.1.3 Integration Angular & Checkpoints**

// Exemple de AuthService avec AngularFire

@Injectable({providedIn: 'root'})

export class AuthService {

user$ = this.afAuth.authState;

constructor(private afAuth: AngularFireAuth) {}

login(email:string, pass:string) { return this.afAuth.signInWithEmailAndPassword(email, pass); }

logout() { return this.afAuth.signOut(); }

}

* **Guards** : AuthGuard pour pages protégées, RoleGuard pour rôles spécifiques.
* **Resolvers** : préchargement de profile utilisateur via ProfileResolver avant activation de route.

**6.2 Cloud Firestore**

**6.2.1 Conception de la base de données**

* **Collections principales** :
  + users/{uid}, trips/{tripId}, chats/{roomId}, games/{gameId}, admin/{settingId}.
* **Sous-collections** : pour éviter les documents trop volumineux :
  + trips/{tripId}/pois, chats/{roomId}/messages, games/{gameId}/scores.
* **Pagination et sharding** :
  + Curseurs startAt, startAfter et limit pour pagination efficace.
  + Sharding de listes volumineuses en collections horodatées (ex. messages\_2025\_07).
* **Indexation** :
  + Déclaration dans firestore.indexes.json et synchronisation via CI.
  + Index composites pour filtres multi-chiens (status+timestamp, owner+visibility).

**6.2.2 Règles de sécurité et validation**

rules\_version = '2';

service cloud.firestore {

match /databases/{db}/documents {

match /users/{uid} {

allow read, write: if request.auth.uid == uid;

}

match /trips/{tripId} {

allow create: if request.auth.uid != null;

allow read: if resource.data.public == true || resource.data.owner == request.auth.uid;

allow update, delete: if resource.data.owner == request.auth.uid;

validateName();

}

function validateName() {

allow write: if request.resource.data.name is string && request.resource.data.name.size() <= 100;

}

// autres collections...

}

}

* **Validation de schéma** : vérifier la présence, le type et la taille des champs.
* **Protection contre les requêtes excessives** : limitation via Cloud Functions et counters Firestore pour throttling.

**6.2.3 Optimisation offline et sync**

* **Caching client** :
  + AngularFirestoreModule.enablePersistence({ synchronizeTabs: true, experimentalTabMultiWindow: true }).
* **Conflits de synchronisation** :
  + Politique lastWriteWins ou usage de champs \_updatedAt pour merges manuels.
* **Batch Writes & Transactions** :
  + Regroupement d’opérations critiques via batch.commit().
  + Transactions pour garantir l’intégrité des mises à jour multi-documents.

**6.3 Firebase Storage**

**6.3.1 Structure et conventions**

* **Arborescence** :
  + Utilisateurs : users/{uid}/avatars/avatar\_{timestamp}.webp.
  + Voyages : trips/{tripId}/photos/photo\_{uuid}.webp.
  + Chat : chat/{roomId}/attachments/{attachmentId}.
* **Nomination** : UUID, timestamp, slash-lowercase pour cohérence.

**6.3.2 Règles de sécurité**

service firebase.storage {

match /b/{bucket}/o {

match /trips/{tripId}/photos/{file} {

allow read: if request.auth.uid != null;

allow write: if request.auth.uid == resource.metadata.ownerId && request.resource.size < 5 \* 1024 \*1024;

}

// autres règles...

}

}

* **MIME type** : restreindre à image/jpeg, image/png, image/webp, video/mp4.
* **Quotas** : contrôler le volume de stockage et la bande passante.

**6.3.3 Traitement et CDN**

* **Compression client** : conversion en WebP, réduction de la résolution avant upload.
* **Cloud Functions onFinalize** :
* exports.generateThumbnail = functions.storage.object().onFinalize(async (object) => {
* // Télécharger l'image, générer thumbnail via Sharp
* // Uploader dans folder thumbs/

});

* **Serving via CDN** :
  + Configurer les réécritures dans firebase.json pour utiliser imageHosting CDN.
  + Headers Cache-Control pour assets statiques.

**6.4 Cloud Functions & Firebase Hosting**

**6.4.1 Cloud Functions orchestrées**

* **HTTP Functions** :
  + Proxy Flight Data : functions.https.onRequest(app) (Express + CORS).
  + Geocoding API wrapper : functions.https.onRequest(geocodeHandler).
* **Background Functions** :
  + Firestore triggers (onCreate, onUpdate) pour notifications, analytics.
  + Auth triggers : functions.auth.user().onCreate() pour initialiser le profil Firestore.
* **Configuration avancée** :
  + runWith({memory: '256MB', timeoutSeconds: 30, minInstances: 1}) pour cold starts.

**6.4.2 Déploiement et Hosting PWA**

// firebase.json

{

"hosting": {

"public": "www",

"ignore": ["firebase.json","\*\*/.\*","\*\*/node\_modules/\*\*"],

"rewrites": [

{"source": "/api/\*\*","function":"apiProxy"},

{"source": "/\*\*","destination":"/index.html"}

],

"headers": [{

"source": "\*\*/\*.{js,css}",

"headers": [{"key":"Cache-Control","value":"public,max-age=31536000,immutable"}]

}]

}

}

* **Service Worker** : généré par ng add @angular/pwa, stratégies performance-first pour assets pré-cache.
* **SEO & Prerender** : utilisation de @nguniversal/express-engine pour SSR, scripts npm run prerender.
* **Custom Domain & SSL** : configuration automatique via Firebase Hosting, support de redirections 301.

**6.5 Capacitor : Packaging natif et plugins**

**6.5.1 Configuration de base**

// capacitor.config.ts

import { CapacitorConfig } from '@capacitor/cli';

const config: CapacitorConfig = {

appId: 'com.company.bob',

appName: 'BoredOnBoard',

webDir: 'www',

bundledWebRuntime: false,

server: { url: 'http://<local-ip>:8100', cleartext: true }

};

export default config;

* **Options supplémentaires** : android.minSdkVersion, ios.deploymentTarget, backgroundColor.

**6.5.2 Plugins clés et pratiques**

* **@capacitor/geolocation** : getCurrentPosition(), watchPosition(), configuration de l’accuracy.
* **@capacitor/camera** : choix de la source (camera, gallery), paramètres de qualité, retour base64 vs fileUri.
* **@capacitor/push-notifications** : inscription, gestion des tokens, channels Android.
* **@capacitor/local-notifications** : planification de rappels contextuels (embarquement, alerte altitude).
* **@capacitor/haptics** : feedback différencié (impact({style: Heavy}), notification({type: Success})).

**6.5.3 Personnalisation iOS & Android**

* **iOS** :
  + Info.plist : demandes d’autorisation pour NSLocationWhenInUseUsageDescription, NSCameraUsageDescription.
  + Assets : LaunchImage, App Icon via Xcode Asset Catalog.
* **Android** :
  + AndroidManifest.xml : permissions (ACCESS\_FINE\_LOCATION, WRITE\_EXTERNAL\_STORAGE), exported flags.
  + styles.xml : thème Material, splash screen via SplashScreen.show().

**6.5.4 CI/CD Mobile & Tests**

* **Live Reload** : ionic cap run android -l --external, proxy config.
* **Debug natif** : via Android Studio, Xcode, inspection DOM via Safari Web Inspector.
* **E2E Mobile** : Appium + Cypress + BrowserStack pour tests cross-device.

**6.6 Workflows CI/CD et gestion des secrets**

**6.6.1 Pipeline Web (GitHub Actions)**

name: CI Web

on: [push]

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: actions/setup-node@v3

- run: npm ci

- run: npm run lint

- run: npm test

- run: npm run build -- --prod

deploy:

needs: build

runs-on: ubuntu-latest

if: github.ref == 'refs/heads/main'

steps:

- uses: actions/checkout@v3

- run: npm ci

- run: npm run build -- --prod

- uses: FirebaseExtended/action-hosting-deploy@v0

with:

repoToken: ${{ secrets.GITHUB\_TOKEN }}

firebaseServiceAccount: ${{ secrets.FIREBASE\_SA\_KEY }}

channelId: live

* **Variables d’environnement** : pass via GitHub Secrets (FIREBASE\_SA\_KEY, API\_URL).
* **Quality Gates** : SonarCloud, Codecov pour la couverture.

**6.6.2 Pipeline Mobile (Fastlane / EAS)**

* **iOS** :
  + fastlane match type: 'appstore', fastlane beta, fastlane release.
  + Gestion des certificats via match (Git Crypto).
* **Android** :
  + eas build --platform all, eas submit --platform android --key ${{ secrets.GOOGLE\_PLAY\_KEY }}.
* **Tests** : npm test, npm run e2e:ci, Appium tests sur émulateurs.

**6.7 Surveillance, monitoring et gestion des coûts**

**6.7.1 Monitoring & alerting**

* **Sentry** : JavaScript, native iOS/Android, Cloud Functions, configuration de releases.
* **Firebase Crashlytics** : suivi des crashes mobiles, associable à Analytics.
* **Cloud Monitoring** : dashboards Grafana/Prometheus, alerts sur erreurs 5XX, latences.

**6.7.2 Optimisation des coûts**

* **Firestore** : contrôler le nombre de lectures/écritures, utiliser onSnapshot uniquement quand nécessaire.
* **Functions** : minimiser le cold start, limiter l’allocation mémoire.
* **Storage** : appliquer la compression, purge automatique des fichiers obsolètes.
* **Hosting** : tirer parti des headers cache, invalidation manuelle vs TTL.

**6.7.3 Backup & résilience**

* **Export automatique** : Firestore backup quotidien vers Cloud Storage via gcloud.
* **Recovery** : scripts gcloud firestore import pour restauration.
* **Geo-replication** : activer multi-région pour Firestore et Storage selon SLA.

**Chapitre 7 : Processus CI/CD, tests automatisés et stratégie de publication**

Ce chapitre fournit un panorama complet des pratiques de **Continuous Integration** (CI) et **Continuous Deployment** (CD) mises en œuvre pour l’application BOB, détaillant les workflows, les outils, les étapes de validation, ainsi que la stratégie de tests et de publication pour les environnements web (PWA) et mobiles (iOS/Android). Ce chapitre traite également de la gouvernance, de l’observabilité et des procédures de gestion des versions, rollbacks et hotfixes.

**7.1 Modèle Git et gestion des branches**

**7.1.1 Stratégie de branching**

* **Trunk-Based Development** :
  + Une seule branche « main » toujours déployable.
  + Branches de courte durée (< 3 jours) nommées selon : feature/nom-court, bugfix/ID-ticket, hotfix/vX.Y.Z.
* **Conventional Commits** :
  + Préfixes (feat:, fix:, chore:, docs:) pour générer automatiquement les changelogs.
  + Corps de commit décrit le « quoi » et le « pourquoi ».
* **Pull Requests (PR)** :
  + Doivent inclure : description, référence de ticket, captures d’écran éventuelles.
  + Reviews obligatoires (au moins deux approbations) et validation des CI pipelines.
  + Labels automatiques : ready-for-review, needs-changelog, breaking-change.

**7.2 Pipelines CI/CD pour la PWA web**

**7.2.1 Phases du pipeline**

1. **Checkout & Setup** : actions/checkout, setup-node (Node 18 LTS).
2. **Installation & Audit** :
   * npm ci pour installer précisément les dépendances.
   * npm audit --audit-level=moderate pour détecter vulnérabilités sévères.
3. **Linting & Style** :
   * ESLint pour TypeScript, Stylelint pour SCSS, Prettier pour le formatage.
4. **Tests unitaires** :
   * Jest + Testing Library (Angular), couverture minimale 85 %.
   * Rapports uploadés comme artefacts pour revue.
5. **Tests d’accessibilité (a11y)** :
   * Axe-core intégré via npm run test:a11y, seuil > 90.
6. **Tests end-to-end (E2E)** :
   * Cypress headless dans un container Docker pour simuler un utilisateur complet.
7. **Performance & SEO** :
   * Lighthouse CI avec budgets configurés : load < 3 s, TTI < 2 s, accessibilité > 90, SEO > 80.
8. **Build production** :
   * npm run build -- --prod, analyse de bundle (source-map-explorer) pour détection de poids anormal.
9. **Déploiement staging** :
   * Firebase Hosting: canal staging pour validation manuelle.
10. **Promotion en production** :

* Merge sur main déclenche déploiement automatique sur canal live.

1. **Notification** :

* Slack/Teams via webhook pour succès/échec, liens vers rapports et environnements preview.

**7.2.2 Exemple de workflow GitHub Actions**

name: CI/CD PWA

on:

push:

branches: [main, staging]

pull\_request:

branches: [main]

jobs:

validate:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- uses: actions/setup-node@v3

with:

node-version: '18'

- run: npm ci

- run: npm audit --audit-level=moderate

- run: npm run lint

- run: npm run test:unit

- run: npm run test:a11y

- run: npm run test:e2e

- uses: sonarsource/sonarcloud-github-action@v1

- uses: actions/upload-artifact@v3

with:

name: coverage

path: coverage/

build\_and\_deploy:

needs: validate

if: github.ref == 'refs/heads/main'

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- run: npm ci

- run: npm run build -- --prod

- uses: FirebaseExtended/action-hosting-deploy@v0

with:

repoToken: ${{ secrets.GITHUB\_TOKEN }}

firebaseServiceAccount: ${{ secrets.FIREBASE\_SA\_KEY }}

channelId: live

**7.3 Pipelines CI/CD pour applications mobiles**

**7.3.1 iOS – Fastlane**

* **match** : gestion des certificats et provisioning profiles (types AppStore, AdHoc).
* **lint** : swiftlint et validation CocoaPods.
* **test** : scan exécute XCTest, génère rapport JUnit.
* **build** : gym assemble une archive IPA signée, versionnée via agvtool.
* **deploy\_beta** : upload\_to\_testflight pour distribution TestFlight.
* **release** : upload\_to\_app\_store, publication publique.
* **post\_actions** : notification Slack + mise à jour du changelog.

Example lane iOS :

lane :beta do

match(type: 'adhoc')

lint

scan

gym(scheme: 'BoredOnBoard')

upload\_to\_testflight(skip\_waiting\_for\_build\_processing: true)

end\```

#### 7.3.2 Android – EAS / Fastlane

- \*\*eas.json\*\* : profils `development` et `production` avec credentials sécurisés.

- \*\*lint\*\* : `./gradlew lint` et `detekt` pour Kotlin.

- \*\*test\*\* : `./gradlew testDebugUnitTest`.

- \*\*build\*\* : `./gradlew bundleRelease`, signature via Keystore stocké dans CI.

- \*\*deploy\_beta\*\* : `supply(track: 'beta')`.

- \*\*release\*\* : `supply(track: 'production')`.

- \*\*notify\*\* : publication de liens vers APK/AAB sur slack.

### 7.4 Tests automatisés approfondis

#### 7.4.1 Tests unitaires

- \*\*Web\*\* : Jest + Angular Testing Library, services isolés via spies/mocks.

- \*\*Mobile\*\* : XCTest (iOS) & JUnit (Android), injection de dépendances pour tests unitaires.

- \*\*Coverage\*\* : minimum 85 %, badge dans README.

#### 7.4.2 Tests end-to-end (E2E)

- \*\*Web\*\* : Cypress, scénario de bout-en-bout couvrant auth, CRUD trips, chat et jeux.

- \*\*Mobile\*\* : Appium + WebDriverIO, exécution sur émulateurs et BrowserStack pour real devices.

- \*\*Reporting\*\* : capture d’écran automatique en cas d’échec, vidéo des sessions.

#### 7.4.3 Tests de performance et sécurité

- \*\*Lighthouse CI\*\* : audits quotidiens et sur chaque PR pour performance, accessibilité, SEO.

- \*\*axe-core\*\* : intégration a11y checks dans Cypress.

- \*\*SAST/DAST\*\* : SonarQube pour code smells et vulnérabilités, ZAP ou OWASP CLI pour sécurité API.

### 7.5 Versioning sémantique et release management

#### 7.5.1 Versioning et changelog

- \*\*SemVer\*\* : versions MAJOR.MINOR.PATCH, automatique via `standard-version`.

- \*\*CHANGELOG.md\*\* : mis à jour à chaque release, publication sur GitHub Releases avec notes et assets.

#### 7.5.2 Cadence de publication

- \*\*Web\*\* : déploiement continu sur main, version mineure toutes les 2 semaines, version majeure trimestrielle.

- \*\*iOS/Android\*\* : cycles bimensuels en beta, publication mensuelle en production.

#### 7.5.3 Rollback et hotfix

- \*\*Rollback\*\* : `firebase hosting: revert --version X` pour web, `fastlane pilot distribute` vers une ancienne version pour iOS, `supply` pour Android.

- \*\*Hotfix\*\* : branche dédiée `hotfix/vX.Y.Z` pour correction critique, merge dans main puis backport vers développement.

### 7.6 Observabilité, monitoring et alerting

- \*\*Logs\*\* : Sentry pour erreurs front & mobile, Google Cloud Logging pour Functions.

- \*\*Metrics\*\* : Firebase Performance Monitoring, Prometheus + Grafana pour Functions, dashboards sur Data Studio.

- \*\*Alertes\*\* : seuils configurés pour erreurs 5XX, pannes de pipeline, régressions de performance, violations de sécurité.

### 7.7 Documentation, gouvernance et support

- \*\*Documentation\*\* : Compodoc (Angular), Jazzy/DocC (iOS), Dokka (Android), hébergée sur un site interne.

- \*\*Governance\*\* : définition des rôles (DevOps, QA, Release Manager, PO), process on-call, SLA de réponse à incidents (4 h).

- \*\*Support\*\* : tickets via Jira/GitHub Issues, workflow d’escalade, rétrospective post-mortem pour incidents majeurs.

---

\*Fin de la documentation\*

**Chapitre 8 : Bonnes pratiques de développement, sécurité et maintenance**

Ce chapitre compile un ensemble de **best practices** et de **procédures opérationnelles** destinées à garantir que l’application BOB reste :

* **Fiable** : code robuste, tests exhaustifs, monitoring continu.
* **Sécurisée** : protection contre les menaces front et back, gestion rigoureuse des accès.
* **Maintenable** : architecture claire, documentation vivante, workflows efficaces.
* **Évolutive** : modularité, refactoring régulier, adoption progressive de nouvelles technologies.
* **Durable** : gestion de la dette technique, cycle de vie maîtrisé, support long terme.

**8.1 Qualité du code et workflows de revue**

**8.1.1 Outils de lint, formatage et hooks Git**

Mettre en place un **pipeline local** garantissant un code formaté et conforme avant chaque commit :

1. **Husky** : hooks Git pré-commit et pré-push.
2. **Lint-Staged** : exécute uniquement sur fichiers modifiés.
3. **ESLint** :
   * Règles partagées spécifiques à Angular, Ionic et TypeScript.
   * Plugins : @typescript-eslint/recommended, eslint-plugin-import, eslint-plugin-jsdoc, eslint-plugin-rxjs.
   * Objectif : zero warnings.
4. **Prettier** :
   * Standardisation du style (indentation 2, singleQuote:true, trailingComma: all).
   * Intégration avec ESLint (eslint-config-prettier).
5. **Stylelint** :
   * Validation SCSS, enforcement de variables (stylelint-config-recommended-scss).
6. **Automatisation** :
   * npm run prepare pour installer Husky.
   * npm run lint et npm run format:check dans CI.

**8.1.2 Conventions, patterns et code metrics**

Adopter des **patterns éprouvés** et mesurer la qualité :

* **Nommage** :
  + Classes & interfaces : PascalCase.
  + Services : feature-name.service.ts.
  + Composants : feature-name.component.ts/html/scss.
  + Constants & enums : SCREAMING\_SNAKE\_CASE.
* **Architecture** :
  + **Domain-Driven Design** : séparation claire des contexts (Auth, Flight, Trip, Chat, Admin).
  + **Feature folders** : chaque module autonome.
* **Design Patterns** :
  + **Facade** : simplifier l’API externe.
  + **Factory** : instanciation de services dynamiques.
  + **Decorator** : enrichir le comportement sans modifier le code existant.
* **TypeScript strict** :
  + Activation de toutes les options (strict, noImplicitAny, strictNullChecks, noUncheckedIndexedAccess).
* **Code Metrics** :
  + **SonarCloud** : complexité cyclomatique < 10, duplication < 3%, couverture > 90%.
  + **Codecov** : badge de couverture dans README.
  + **Bundle analysis** : garder le bundle < 500 kB gzippé.

**8.1.3 Pull Requests, revues et pair programming**

Les **pull requests** doivent :

1. **Décrire** l’objectif, le ticket associé, l’impact fonctionnel.
2. **Inclure** captures d’écran ou GIF pour UI, capture console pour back.
3. **Passer** le pipeline CI (lint, tests, security scan).
4. **Obtenir** au moins deux approbations, dont un membre QA.
5. **Mettre à jour** le CHANGELOG.md (entrée ### [Unreleased]).

**Pair programming** :

* Recommandé pour les fonctionnalités critiques (> 2 jours de dev).
* Sessions guidées pour les nouveaux entrants.
* Documentation de session (résumé, décisions architecturales).

**8.2 Sécurité applicative et gestion des accès**

**8.2.1 Sécurité front-end (Angular / Ionic)**

1. **Content Security Policy (CSP)** :
   * Définir un en-tête CSP restrictif en production.
   * Utiliser nonce ou hash pour scripts inline autorisés.
2. **Sanitization** :
   * DomSanitizer pour tous les contenus dynamiques.
   * Bannissement de tout innerHTML sans contrôle.
3. **HTTP Interceptors** :
   * Gestion centralisée des erreurs 401, 403.
   * Rejection des réponses avec en-têtes non sécurisés.
4. **Hardening Angular** :
   * Activation de enableProdMode() en prod.
   * Désactivation de JIT compiler.

**8.2.2 Sécurité backend (Node.js / Firebase Functions)**

1. **En-têtes de sécurité** :
   * Helmet pour HSTS, XSS-Protection, X-Frame-Options, X-Content-Type-Options.
2. **Validation d’entrées** :
   * class-validator pour DTOs, Joi ou Yup pour Express.
3. **Protection NoSQL** :
   * Règles Firestore robustes, validation schématique via request.resource.data.is(...).
4. **Rate Limiting & DDoS** :
   * express-rate-limit, Google Cloud Armor pour functions.
5. **Logging & Auditing** :
   * Logs d’accès, audit des modifications de schéma.

**8.2.3 Gestion des secrets et accès**

* **Secret Manager** :
  + Stockage centralisé des clés API, tokens OAuth.
  + Récupération dynamique en CI/CD.
* **IAM & RBAC** :
  + Custom claims Firebase pour rôles.
  + Principe de moindres privilèges.
* **Rotation** :
  + Automatiser la rotation trimestrielle, envoyer alertes 2 semaines avant expiration.

**8.3 Gestion des dépendances et versions**

**8.3.1 Mise à jour et audit**

* **Dependabot / Renovate** :
  + PR automatiques pour patch/minor, revue requise pour major.
* **Audit vulnérabilités** :
  + npm audit, Snyk daily, intégration dans CI.
* **Rapports** :
  + Dashboard mensuel, alertes Slack pour vulnérabilités critiques.

**8.3.2 Verrouillage et registry privée**

* **Lockfiles** : package-lock.json, yarn.lock commités.
* **Registry interne** : GitHub Packages, Nexus pour paquets privés.
* **Semantic Versioning** : MAJOR.MINOR.PATCH, version bump automatique via standard-version.

**8.4 Refactoring, dette technique et évolutivité**

**8.4.1 Suivi de la dette technique**

* **Issues GitHub** : label technical-debt.
* **Metrics** : duplication, complexité, coverage via SonarCloud.
* **Revues trimestrielles** : audit de dette, priorisation.

**8.4.2 Planification et exécution**

* **Capacité dédiée** : 15% du sprint.
* **Tâches atomiques** : < 1 jour de travail.
* **Test de non-régression** : retester suite complète avant merge.

**8.4.3 Adoption progressive**

* **Proof-of-Concept** : projet isolé.
* **Feature Flags** : basculer progressivement.
* **Monitoring** : mesurer impact post-déploiement.

**8.5 Documentation et onboarding**

**8.5.1 Guides et diagrammes**

* **Style Guide** : conventions, patterns, C4, DDD.
* **Diagrammes** : navigation, CI/CD, architecture microservice.

**8.5.2 Documentation générée**

* **Compodoc** : Angular.
* **DocC / Jazzy** : iOS.
* **Dokka** : Android.
* **Hébergement** : GitHub Pages, ReadTheDocs.

**8.5.3 Onboarding**

* **Templates repo** : clonable, setup.sh.
* **Guides** : Getting Started, FAQ.
* **Pairing** : mentor first week.

**8.6 Maintenance, support, DRP et observabilité**

**8.6.1 Monitoring et alerting**

* **Sentry** : front & mobile.
* **Crashlytics** : mobile natif.
* **Prometheus & Grafana** : Firestore, Functions.
* **Alertes** : Slack, pager duty, seuils personnalisés.

**8.6.2 Sauvegardes et reprise après sinistre**

* **Backups** : Firestore daily, Storage snapshots.
* **Multi-région** : haute dispo.
* **Runbooks** : RTO < 1h, RPO < 15min.

**8.6.3 Cycle de vie des versions**

* **Support** : 2 dernières versions majeures.
* **EOL** : annoncé 2 mois avant.
* **Hotfix** : workflow dédié, merge back.

**Chapitre 9 : Perspectives, améliorations futures et modèle Freemium**

Alors que BOB continue de se positionner comme le compagnon de voyage incontournable, il est essentiel d’anticiper les **évolutions stratégiques** et de proposer des **innovations à forte valeur ajoutée**. Ce dernier chapitre esquisse une **feuille de route ambitieuse** articulée autour :

1. D’un **modèle Freemium** robuste, adapté aux besoins variés des utilisateurs, avec un contraste clair entre l’offre gratuite et les services Premium.
2. Du développement de **fonctionnalités avancées** et exclusives pour les abonnés Pro, intégrant la cartographie offline, l’IA et la collaboration.
3. De nouvelles **sources de revenus** via la publicité intelligente, les partenariats stratégiques et la monétisation de datas anonymisées.
4. D’une **roadmap détaillée** sur 4 phases de 0 à 12 mois avec des jalons précis et des **indicateurs clés** de performance (KPI).
5. D’une réflexion sur l’**expansion** du périmètre fonctionnel et géographique, incluant de nouveaux segments de mobilité et des intégrations cross-industrielles.

**9.1 Modèle Freemium : différencier Free vs Pro**

Le modèle Freemium de BOB doit offrir une **gratuité convaincante** tout en donnant une **justification forte** à l’abonnement. La matrice suivante illustre la granularité des services :

|  |  |  |
| --- | --- | --- |
| **Fonctionnalité** | **Gratuit (Free)** | **Premium (Pro)** |
| **Suivi de vol** | Mise à jour position toutes les 5 min | Tracking toutes les 30 s, historique complet des précédents vols, analyses comparatives |
| **Cartographie** | Vue simple en ligne | Téléchargement de zones GPS, cartes offline vectorielles, recherche POI sans connexion |
| **Données météo** | Conditions basiques (température, vent) | METAR/TAF en temps réel, alertes turbulence, prévisions horaires détaillées |
| **Gestion de voyages** | CRUD trips basique | Partage et édition collaborative, synchronisation multi-appareils, workflows personnalisés |
| **Messagerie & social feed** | Chat public, likes | Salles privées, sondages integrés, bots d’assistance basés sur IA |
| **Jeux embarqués** | Accès à 2 jeux gratuits | Catalogue complet (+5 jeux exclusifs), tournois hebdomadaires, leaderboards mondiaux |
| **Notifications Push** | Alerts simples (départ/arrivée) | Rappels intelligents (check-in, gate change), notifications géolocalisées |
| **Streaming audio** | Radio FM intégrée | Podcasts sélectionnés, audio-guide de l’aéroport, playlists premium |
| **Personnalisation UI** | Light/Dark mode | Thèmes sur-mesure, adaptatifs (couleur compagne, logo), animations exclusives |
| **Téléchargement de médias** | Photos et notes | Vidéos Travelogue HD, PDF de journal de bord |
| **Export & rapports** | — | Génération de rapports PDF/Excel détaillés et partageable |
| **Publicité** | Bandeaux génériques | Zéro pub |
| **Support & assistance** | Support Forum | Support Live Chat 24/7, SLA < 1 h |
| **Tarification** | Gratuit | 6.99 €/mois ou 69 €/an avec essai gratuit 14 jrs |

**Cible** : convertir progressivement 5 % des utilisateurs Free en abonnés Pro d’ici la fin de la Phase 1, puis stabiliser à 8 %.

**9.2 Fonctionnalités Premium à développer**

**9.2.1 Cartographie Offline et navigation enrichie**

* **Pack Tuiles Dynamiques** : sélection intelligente des tuiles à télécharger selon l’itinéraire.
* **Guidage Indoor/Outdoor** : orientation pas-à-pas dans les terminaux via BLE ou ARKit/ARCore.
* **Recherches Avancées** : filtres par catégories (lounges, duty-free, restaurants), évaluations et horaires.

**9.2.2 Smart Alerts & Analytics avancés**

* **Tableau de bord personnalisé** : OUI basées sur la durée, retards cumulés, vitesse moyenne, altitude max.
* **Notifications Proactives** : alertes de porte modifiée, consignes sécurité, porte d’urgence.
* **Analyse prédictive** : Machine Learning pour anticiper retards et recommander des alternatives de vol.

**9.2.3 Collaboration et Social / Pro**

* **Espace Équipe** : workflow de trip partagé, assignation de tâches (packaging, check-in).
* **Modération et Sécurité** : modération automatique via NLP, filtres anti-spam.
* **Événements Live** : quiz gamifiés synchronisés avec le vol, sondages d’ambiance, intégration Twitch.

**9.2.4 Branding, fidélité et marketing intégré**

* **Thèmes co-brandés** : personnalisation UI pour chaque compagnie aérienne (branding unique).
* **Programme de Récompenses** : points cumulés à chaque utilisation, échanges contre des services (fast-track, lounge).
* **Offres Partner** : accès à des promotions exclusives (hotels, location de voitures), intégration API loyalty.

**9.2.5 Intelligence Artificielle et innovation**

* **Assistant Conversationnel** : support vocal/textuel via chatbot intelligent intégré, guidance en vol.
* **Recommandations Contextuelles** : contenus, restaurants et services basés sur profil et historique.
* **Prédiction du flux passagers** : modèles IA pour anticiper embouteillages et recommander des plages horaires.

**9.3 Monétisation publicitaire et partenariats**

**9.3.1 Publicité native et reward-based**

* **Formats natifs** : bandeaux discrets, cards sponsorisées insérées dans le feed.
* **Rewarded Ads** : visionnage volontaire de courtes vidéos (10–15 s) pour débloquer une ressource Premium (e.g. accès offline tempor.)
* **Data-driven Targeting** : segmentation par destination, classe de vol, centre d’intérêt (shopping, gastronomie).

**9.3.2 Partenariats stratégiques B2B**

* **Co-branding compagnies aériennes** : application aux couleurs de l’opérateur, intégration CRM pour upsell.
* **Tourisme local** : guides payants, packs excursions in-app, réservation directe avec commissions.
* **Retail & Duty-Free** : bons de réduction dynamiques, paiement in-app, click-and-collect.

**9.3.3 Services Data et insights pour partenaires**

* **API Insights Premium** : accès limité aux agrégats anonymes sur flux passagers, points chauds, taux d’engagement.
* **Portail Partenaire** : dashboard temps réel, export CSV/JSON, alerting sur spikes de trafic.
* **White-label Solutions** : SDK et API pour intégration de BOB dans applications tierces.

**9.4 Roadmap et jalons clés**

La mise en œuvre se décline en phases ciblées avec **livrables opérationnels** et **KPI mesurables** :

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Durée | Livrables clés | KPI visés |
| **1** | 0–3 mois | Lancement Freemium, bilan UX, billing impl. | Conv. Pro ≥ 5 %, churn < 10 % |
| **2** | 3–6 mois | Offline maps, Smart Alerts, rapports PDF | Offline usage ≥ 25 %, alert opt-in ≥ 40 % |
| **3** | 6–9 mois | Collaboration Pro, Social Live, AR Indoor | Engagem. social ≥ 30 sessions/mois |
| **4** | 9–12 mois | API B2B, partenariats, IA avancée | Revenus B2B ≥ 20 % CA total |

**Suivi trimestriel** : ajustement des priorités selon feedback NPS, data analytics et performance financière.

**9.5 Expansion opportunités et futurs segments**

* **Nouveaux modes de transport** : adaptation BOB pour trains (TGV, ICE) et croisières, en exploitant la même base modulable.
* **Internationalisation** : localisation complète (20+ langues), intégration réglementaire par région.
* **Écosystème IoT** : intégration de wearables (watch notifications), beacons pour indoor positioning.
* **Communauté et écosystème ouvert** : plugins, marketplace d’extensions (guide linguistique, jeux créés par la communauté).