



Migration Flutter & Roadmap Production

POC Web → Application Mobile Flutter

Date: 26 décembre 2025

■ Table des matières

- 1. Vue d'ensemble du projet
- 2. État actuel du POC Web
- 3. Architecture Flutter cible
- 4. Phase 1: Migration POC de base
- 5. Phase 2: Multi-voyages
- 6. Phase 3: Authentification
- 7. Phase 4: Backend Firebase
- 8. Phase 5: Features avancées
- 9. Checklist de migration
- 10. Timeline & Priorités

1. Vue d'ensemble du projet

Objectif

Migrer le POC web Tripline (HTML/CSS/JS) vers une application mobile Flutter native avec backend Firebase, authentification, et architecture multi-voyages.

Pourquoi Flutter ?

- **Cross-platform:** iOS + Android avec un seul codebase
- **Performance native:** Compilation en code natif (ARM/x64)
- **Hot reload:** Développement ultra-rapide
- **Material Design:** Composants UI modernes intégrés
- **Écosystème riche:** Packages pour tout (Firebase, Maps, etc.)
- **Maintenance:** Plus simple qu'une app React Native ou native

Stack technique cible

Composant	Technologie
Frontend	Flutter 3.x (Dart)
State Management	Provider ou Riverpod
Backend	Firebase (Firestore + Auth + Storage)
Base de données	Cloud Firestore
Authentification	Firebase Auth (Google, Apple, Email)
Storage	Firebase Storage (photos)
Analytics	Firebase Analytics
Notifications	Firebase Cloud Messaging
Maps	Google Maps Flutter

2. État actuel du POC Web

Features implémentées ■

- **CRUD complet:** Ajouter/Modifier/Supprimer restaurants & activités
- **Système isDone:** Marquer comme fait/pas fait avec tri automatique
- **Priorités:** Must-do, Haute, Normale, Basse, Optionnel
- **Timeline carousel:** Vue par jour avec swipe (style Instagram)
- **Search & Filters:** Recherche + filtres par ville + tri
- **Export/Import:** JSON pour backup
- **Dark mode:** Thème clair/sombre
- **Design Apple:** Minimaliste avec Lucide Icons
- **Mobile-first:** Responsive avec bottom nav
- **localStorage:** Persistence locale des données

Limitations actuelles ■■

- **Un seul voyage:** Pas de multi-voyages
- **Local only:** Données uniquement sur l'appareil
- **Pas de sync:** Impossible de partager entre appareils
- **Pas d'auth:** Pas de compte utilisateur
- **Pas de backup cloud:** Risque de perte de données
- **Web uniquement:** Pas d'app mobile native

Modèle de données actuel

Restaurant:

```
{ id, name, city, cuisine, priceRange, googleMapsUrl, photoUrl, notes,  
isReserved, reservationDate, priority, bookingUrl, isDone }
```

Activity:

```
{ id, name, city, category, date, time, duration, cost, googleMapsUrl, photoUrl,  
notes, isBooked, reservationDate, priority, bookingUrl, isDone }
```

3. Architecture Flutter cible

Structure de dossiers

```
lib/
└── main.dart
└── models/
    ├── restaurant.dart
    ├── activity.dart
    └── trip.dart
└── screens/
    ├── home_screen.dart
    ├── trip_list_screen.dart
    ├── trip_detail_screen.dart
    ├── timeline_screen.dart
    ├── add_item_screen.dart
    └── auth/
        ├── login_screen.dart
        └── register_screen.dart
└── widgets/
    ├── item_card.dart
    ├── timeline_day.dart
    ├── priority_badge.dart
    └── bottom_nav.dart
└── services/
    ├── firebase_service.dart
    ├── auth_service.dart
    └── storage_service.dart
└── providers/
    ├── trip_provider.dart
    ├── auth_provider.dart
    └── theme_provider.dart
└── utils/
    ├── constants.dart
    └── helpers.dart
```

Packages Flutter requis

- **firebase_core**: Firebase SDK
- **firebase_auth**: Authentification
- **cloud_firestore**: Base de données
- **firebase_storage**: Stockage photos
- **provider**: State management
- **google_maps_flutter**: Cartes
- **intl**: Formatage dates/nombres
- **image_picker**: Photos depuis caméra/galerie
- **url_launcher**: Ouvrir liens externes
- **shared_preferences**: Cache local

4. Phase 1: Migration POC de base

Durée estimée: 2-3 semaines

Objectifs

Reproduire toutes les fonctionnalités actuelles du POC web dans Flutter, sans authentification ni backend (données locales uniquement).

Étapes détaillées

1. Setup projet Flutter

- Créer nouveau projet: `flutter create tripline`
- Configurer iOS et Android
- Ajouter packages de base dans pubspec.yaml
- Setup assets (fonts, images, icons)

2. Créer les models

- Restaurant model avec toJSON/fromJSON
- Activity model avec toJSON/fromJSON
- Ajouter validation des champs

3. Interface utilisateur de base

- Bottom navigation (4 onglets)
- App bar avec logo et actions
- Theme provider (light/dark)
- Color scheme Apple (primary: #007AFF)

4. CRUD local avec shared_preferences

- Service de stockage local
- Méthodes save/load/delete
- Provider pour state management
- Liste des restaurants et activités

5. Cards & Timeline

- ItemCard widget (photo + info)
- Timeline carrousel avec PageView
- Indicateurs ronds (page indicators)
- Animations de transition

6. Formulaires

- Form d'ajout restaurant/activité
- Validation des champs
- Date/time pickers

→ Image picker pour photos

→ Priority selector

7. Search & Filters

→ Search bar avec debounce

→ Filtres par ville (dropdown)

→ Tri (price, priority, name)

→ Affichage résultats filtrés

8. Features supplémentaires

→ Marquer comme fait/pas fait

→ Export/Import JSON

→ Dark mode toggle

→ Settings screen

Livrables Phase 1

■ App Flutter fonctionnelle (iOS + Android)

■ Toutes les features du POC web reproduites

■ Stockage local (shared_preferences)

■ Design Apple minimaliste

■ Timeline carrousel avec indicateurs

■ Dark mode fonctionnel

5. Phase 2: Multi-voyages

Durée estimée: 1-2 semaines

Objectifs

Transformer l'app en gestionnaire multi-voyages avec écran d'accueil listant tous les trips et navigation entre voyages.

Nouveau modèle de données

Trip model:

```
class Trip {  
    String id;  
    String name;  
    String? destination;  
    DateTime startDate;  
    DateTime endDate;  
    String currency;  
    double? budget;  
    List<Restaurant> restaurants;  
    List<Activity> activities;  
    DateTime createdAt;  
    DateTime updatedAt;  
}
```

Étapes détaillées

1. Refactorisation du modèle

- Créer Trip model
- Migrer restaurants/activities dans Trip
- Ajouter metadata (dates, budget, currency)
- Migration des données existantes

2. Trip List Screen

- Écran d'accueil avec liste des voyages
- Card pour chaque trip (photo, dates, stats)
- FAB 'Créer un voyage'
- Swipe to delete
- Empty state si aucun voyage

3. Trip Management

- Formulaire création voyage
- Édition nom/dates/budget
- Suppression avec confirmation
- Duplication de voyage

4. Navigation

- Switch entre voyages
- Breadcrumb: Voyages > Japon 2026
- Deep linking vers un trip
- Retour à la liste

5. Stockage multi-trips

- Adapter shared_preferences pour liste de trips
- currentTripId pour trip actif
- Sauvegarde/chargement optimisés

Livrables Phase 2

- Multi-voyages fonctionnel
- Écran d'accueil avec liste de trips
- CRUD complet sur les voyages
- Navigation fluide entre trips
- Données migrées automatiquement

6. Phase 3: Authentification

Durée estimée: 1 semaine

Objectifs

Ajouter l'authentification Firebase pour créer des comptes utilisateurs et préparer la synchronisation cloud.

Méthodes d'authentification

- **Google Sign-In:** Login rapide avec compte Google
- **Apple Sign-In:** Obligatoire pour iOS (Apple requirement)
- **Email/Password:** Authentification classique
- **Anonymous:** Pour tester sans créer de compte

Étapes détaillées

1. Setup Firebase

- Créer projet Firebase
- Ajouter iOS app (Bundle ID)
- Ajouter Android app (Package name)
- Télécharger google-services.json et GoogleService-Info.plist
- Configurer firebase_core dans main.dart

2. Firebase Auth Setup

- Activer Email/Password dans Firebase Console
- Activer Google Sign-In
- Activer Apple Sign-In (iOS)
- Créer AuthService avec méthodes login/logout/register

3. UI d'authentification

- Login screen avec email/password
- Register screen
- Boutons Google/Apple Sign-In
- Forgot password flow
- Loading states et error handling

4. Auth State Management

- AuthProvider avec StreamBuilder
- Écoute du auth state (connecté/déconnecté)
- Redirection auto si non connecté
- Persist auth state

5. Profile & Settings

- Écran profil utilisateur
- Affichage email/nom
- Bouton logout
- Delete account option

Livrables Phase 3

- Firebase Auth configuré
- Login/Register fonctionnels
- Google + Apple Sign-In
- Auth state management
- Profile screen
- Gestion des erreurs auth

7. Phase 4: Backend Firebase

Durée estimée: 2-3 semaines

Objectifs

Migrer du stockage local vers Cloud Firestore pour sync multi-devices et backup automatique dans le cloud.

Structure Firestore

```
users/  
  {userId}/  
    email: string  
    displayName: string  
    photoURL: string  
    createdAt: timestamp  
    trips (subcollection)/  
      {tripId}/  
        name: string  
        destination: string  
        startDate: timestamp  
        endDate: timestamp  
        currency: string  
        budget: number  
        createdAt: timestamp  
        updatedAt: timestamp  
        restaurants (subcollection)/  
          {restaurantId}/  
            name, city, cuisine, etc.  
        activities (subcollection)/  
          {activityId}/  
            name, city, category, etc.
```

Étapes détaillées

1. Firestore Setup

- Activer Cloud Firestore dans Firebase Console
- Configurer Security Rules (users can only access their data)
- Créer indexes pour queries optimisées
- Setup Firebase Storage pour photos

2. Firebase Service

- FirebaseService avec CRUD methods
- createTrip, updateTrip, deleteTrip
- addRestaurant, updateRestaurant, deleteRestaurant
- addActivity, updateActivity, deleteActivity
- Queries avec filters et sorting

3. Sync temps réel

- StreamBuilder pour écouter les changements
- Auto-refresh quand données changent
- Conflict resolution (last write wins)
- Optimistic updates pour UX fluide

4. Upload photos

- Compression d'images avant upload
- Upload vers Firebase Storage
- Génération d'URL publique
- Suppression des anciennes photos

5. Offline support

- Enable Firestore offline persistence
- Cache local avec shared_preferences
- Queue des opérations offline
- Sync automatique au retour online

6. Migration des données

- Script de migration local → Firestore
- Import des trips existants
- Vérification de l'intégrité
- Cleanup local après migration

Security Rules Firestore

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
    match /trips/{tripId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
    match /{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

Livrables Phase 4

- Firestore configuré avec security rules
- Sync temps réel multi-devices
- Upload photos vers Firebase Storage
- Offline support avec cache
- Migration automatique des données

■ Backup cloud automatique

8. Phase 5: Features avancées

Durée estimée: 3-4 semaines

Features à implémenter

Budget Tracker ■

- Calcul automatique total dépenses
- Progress bar visuelle
- Alerte dépassement budget
- Répartition par catégorie (food, transport, activities)
- Export CSV des dépenses

Map View ■■

- Google Maps intégration
- Markers pour chaque lieu
- Clustering si beaucoup de points
- Directions entre 2 lieux
- Street View preview

Itinéraire optimisé ■

- Calcul meilleur ordre de visite
- Affichage temps de trajet
- Détection conflits horaires
- Suggestions d'optimisation
- Export vers Google Maps

Météo ■■

- API météo (OpenWeather ou WeatherAPI)
- Prévisions pour dates du voyage
- Affichage dans timeline par jour
- Alertes température/pluie

Partage de voyage ■

- Génération lien de partage
- Mode read-only pour invités
- Collaboration temps réel (optionnel)
- Export PDF avec itinéraire complet

Packing List ■

- Checklist de bagages
- Suggestions automatiques selon destination

- Check/uncheck items
- Catégories (vêtements, tech, docs)

Notifications ■

- Rappels avant départ
- Notifications réservations
- Push notifications via FCM
- Scheduling local

Analytics ■

- Firebase Analytics
- Tracking des features utilisées
- Crashlytics pour bugs
- Performance monitoring

9. Checklist de migration complète

Phase 1 - POC Base

- Setup projet Flutter
- Models (Restaurant, Activity)
- UI de base (Bottom nav, AppBar)
- CRUD local
- ItemCard widget
- Timeline carousel
- Formulaires ajout/édition
- Search & Filters
- isDone feature
- Export/Import JSON
- Dark mode
- Tests iOS et Android

Phase 2 - Multi-voyages

- Trip model
- Trip List Screen
- CRUD trips
- Navigation entre trips
- Migration données
- Breadcrumb navigation

Phase 3 - Auth

- Firebase project setup
- Firebase Auth configuration
- Login/Register screens
- Google Sign-In
- Apple Sign-In
- Auth state management
- Profile screen
- Logout & delete account

Phase 4 - Backend

- Firestore setup
- Security rules
- Firebase Service
- Sync temps réel
- Upload photos (Firebase Storage)
- Offline support
- Migration local → Firestore

Phase 5 - Features avancées

- Budget tracker
- Map view
- Itinéraire optimisé
- Météo
- Partage de voyage
- Packing list
- Notifications
- Analytics

QA & Deployment

- Tests unitaires
- Tests d'intégration
- Tests UI
- Performance testing
- iOS build & TestFlight
- Android build & Play Store beta
- App Store submission
- Play Store submission

10. Timeline & Priorités

Planning recommandé

Phase	Durée	Priorité	Status
Phase 1: POC Base	2-3 semaines	CRITIQUE	À faire
Phase 2: Multi-voyages	1-2 semaines	CRITIQUE	À faire
Phase 3: Auth	1 semaine	HAUTE	À faire
Phase 4: Backend	2-3 semaines	HAUTE	À faire
Phase 5: Features avancées	3-4 semaines	MOYENNE	À faire
QA & Deployment	1-2 semaines	HAUTE	À faire
TOTAL	10-15 semaines		

Ordre de priorité

- Phase 1 (POC):** Reproduire POC web → App fonctionnelle standalone
- Phase 2 (Multi-voyages):** Architecture scalable avec plusieurs trips
- Phase 3 (Auth):** Comptes utilisateurs pour préparer le cloud
- Phase 4 (Backend):** Sync cloud + backup automatique
- Phase 5 (Features):** Différentiateurs (budget, map, météo, etc.)

Stratégie de développement

Approche **itératrice et incrémentale**: chaque phase produit une app fonctionnelle et déployable. Cela permet de tester régulièrement avec de vrais utilisateurs et d'ajuster les priorités selon les retours.

- **MVP First:** Phase 1+2 = MVP déployable (mono-user, multi-trips)
- **Cloud Second:** Phase 3+4 = Version cloud (auth + sync)
- **Premium Last:** Phase 5 = Features premium pour monétisation
- **Beta Testing:** TestFlight/Play Store beta dès la Phase 2
- **Feedback Loop:** Itérations rapides basées sur retours users

■ Résumé Exécutif

- **POC Web validé**: Design Apple, Timeline carousel, isDone, Search/Filters
- **Migration Flutter**: 10-15 semaines pour app production-ready
- **5 Phases progressives**: POC → Multi-voyages → Auth → Backend → Features
- **Stack moderne**: Flutter + Firebase + Firestore
- **Deployment**: App Store + Play Store en beta testing dès Phase 2

Ce roadmap permet de transformer le POC web en une application mobile complète, scalable et production-ready, tout en itérant rapidement avec de vraies données utilisateurs.

Prochaine étape: Démarrer la Phase 1 avec setup du projet Flutter et reproduction des features de base.