AGENT-READY PRODUCT SPECIFICATION

Travel Companion App

Team	Team 11 — Darius · Varun · Allen · Kapil
Version	1.0
Stack	React Native (Expo) + ElysiaJS (Bun) + PostgreSQL + SQLite
Monorepo	Turborepo + pnpm workspaces
Timeline	9 Weeks | Feb 27 → Apr 30, 2025 | 5 Phases
Auth	Email/password + Google OAuth (Android) + Apple OAuth (iOS)
AI	Qwen 2.5 fine-tuned — runs on-device via llama.rn (no server inference)
Analytics	PostHog (Phase 5)
Last Updated	February 2025

Read this entire specification before writing a single line of code.

This document is the single source of truth. When the spec and your judgment conflict, implement the spec and add a `// SPEC QUESTION:` comment.

Table of Contents

1. Product Overview

Travel Companion is a mobile application that centralizes everything a traveler needs into a single, lightweight, offline-first app: itinerary planning, interactive offline maps, curated local recommendations, a local language cheat sheet, and an on-device AI travel assistant. Travelers no longer need to juggle multiple disconnected services or worry about poor connectivity abroad.

The core loop: a user plans a trip, downloads an offline city pack, and then explores with full access to their itinerary, map, recommendations, and AI chat — with or without internet. When connectivity is restored the app syncs all changes back to the server automatically.

1.1 Target Users

- International travelers in areas with limited or expensive mobile data
- Domestic tourists visiting unfamiliar cities
- Solo travelers who need a personal guide on demand
- Groups who want to share and coordinate itineraries (Phase 4+)

1.2 Core Value Proposition

- Offline-first — all core features work without internet after a city pack is downloaded
- All-in-one — itinerary, map, recommendations, language guide, and AI chat in one app
- On-device AI — Travel LLM runs locally on the user's phone, no server calls, works offline
- Social layer — optionally connect with other travelers at the same destination

Attribute	Details
Platform	Mobile App — React Native (Expo). iOS and Android.
Auth	Email/password · Google OAuth (Android only) · Apple OAuth (iOS only)
Offline	Offline-first. Core features work after city pack download. Auto-syncs on reconnect.
AI	Qwen 2.5 fine-tuned on travel data. Runs on-device via llama.rn (llama.cpp bindings). No backend inference. Works offline.
Analytics	PostHog — event tracking and feature flags. Added in Phase 5.
Environments	Development (local) · Staging (cloud preview) · Production

2. Tech Stack

All technology decisions below are fixed. Do not substitute libraries without adding a // ARCH DECISION: comment.

Layer	Technology	Notes / Constraints
Monorepo	Turborepo + pnpm workspaces	apps/mobile · apps/backend · packages/database · packages/shared-types. Use pnpm for all installs.
Frontend	React Native + TypeScript (Expo SDK)	Expo managed workflow. Strict TypeScript. No class components.
Routing	expo-router	File-based routing in apps/mobile/src/app/. Use Expo Router stack + tabs.
Styling	NativeWind v4 (Tailwind for RN)	No StyleSheet.create() except where NativeWind cannot achieve the result.
Animations	React Native Reanimated v3	All transitions and gesture-driven animations.
Gestures	React Native Gesture Handler	Required Reanimated peer dep. Handles swipe-to-delete, drag-to-reorder.
Forms	React Hook Form + Zod	All forms use RHF. Validation schemas in Zod (client). TypeBox (server). Mirror schemas where possible.
UI Components	react-native-reusables	Accessible headless RN component primitives. Styled with NativeWind.
State	Zustand	Client-side: auth, trip, offline, chat (LLM), UI state.
Storage	MMKV (react-native-mmkv)	Replaces AsyncStorage for all key-value persistence. Synchronous, significantly faster.
HTTP Client	eden-tanstack-react-query	Single package combining Eden Treaty (type-safe client) + TanStack Query (caching/mutations). Generated from backend App type.
Date/Time	date-fns	Never use moment.js or dayjs.
Local DB	SQLite (expo-sqlite)	Offline city pack data: places, phrases, itinerary items. Schema in apps/mobile/src/db/.
Maps	Mapbox SDK (react-native-mapbox-gl)	Offline tile packs downloaded as part of city pack.
On-Device LLM	llama.rn (llama.cpp React Native bindings)	Runs Qwen 2.5 GGUF model locally on device. No server inference endpoint. Works offline after model download.
Backend	ElysiaJS + TypeScript (Bun runtime)	Bun as runtime (not Node). All backend scripts: bun run, bun test, etc.
Validation	TypeBox (ElysiaJS native)	All incoming API data validated server-side. One model.ts per module.

ORM	Prisma v5 (packages/database)	Schema-first. Shared across all backend services.
Relational DB	PostgreSQL (Neon)	Separate DB per environment. String: DATABASE_URL.
Cache	Upstash Redis	Rate limiting, recommendation/phrase cache (1hr TTL).
Auth	JWT (access + refresh tokens)	httpOnly cookies for refresh. Google OAuth via @react-native-google-signin/google-signin (Android). Apple via expo-apple-authentication (iOS).
CDN	Cloudflare CDN	Phase 5. City pack assets behind Cloudflare. Cache busted on packVersion change.
Analytics	PostHog React Native	Phase 5. Event tracking, screen views, feature flags. Never send PII.
Testing	Jest + React Native Testing Library	Min 70% coverage for utilities and API routes.
Crash Reporting	Sentry	Phase 5. Unhandled errors captured with userId + screen name. No PII beyond userId.

3. Project File Structure

Create files in this exact structure. Do not reorganize without a // ARCH DECISION: comment. The backend follows the Elysia module pattern: each feature lives in src/modules/<feature>/ with three files — index.ts (controller), service.ts (business logic + Prisma), model.ts (TypeBox schemas).

```
travel-companion/
├── apps/
│   ├── mobile/                        # React Native (Expo) app
│   │   ├── src/
│   │   │   ├── components/
│   │   │   │   ├── auth/              # LoginForm, RegisterForm, OAuthButton
│   │   │   │   ├── itinerary/         # ItineraryBuilder, DayCard, ItemRow
│   │   │   │   ├── map/               # MapView, OfflineMapBanner, PlacePin
│   │   │   │   ├── recommendations/   # RecommendationList, PlaceCard
│   │   │   │   ├── language/          # PhraseCard, CategoryPicker
│   │   │   │   ├── chat/              # ChatWindow, MessageBubble, TypingIndicator
│   │   │   │   ├── social/            # NearbyTravelers, ConnectionCard,
MessageThread
│   │   │   │   └── shared/            # Button, Modal, Toast, Skeleton, Badge
│   │   │   ├── app/                   # expo-router file-based routes
│   │   │   │   ├── (auth)/
│   │   │   │   │   ├── login.tsx
│   │   │   │   │   └── register.tsx
│   │   │   │   ├── (tabs)/
│   │   │   │   │   ├── index.tsx      # Home
│   │   │   │   │   ├── map.tsx
│   │   │   │   │   ├── recommendations.tsx
│   │   │   │   │   ├── chat.tsx
│   │   │   │   │   └── social.tsx
│   │   │   │   ├── trip/
│   │   │   │   │   ├── [id].tsx       # Trip detail + itinerary
│   │   │   │   │   └── new.tsx
│   │   │   │   ├── place/[id].tsx
│   │   │   │   ├── settings.tsx
│   │   │   │   └── _layout.tsx
│   │   │   ├── store/                 # Zustand stores
│   │   │   │   ├── authStore.ts
│   │   │   │   ├── tripStore.ts
│   │   │   │   ├── offlineStore.ts
│   │   │   │   ├── chatStore.ts       # on-device LLM state
│   │   │   │   └── uiStore.ts
│   │   │   ├── api/                   # eden-tanstack-react-query — one file per
resource
│   │   │   │   ├── client.ts          # Eden Treaty instance + queryClient
│   │   │   │   ├── trips.ts
│   │   │   │   ├── itinerary.ts
│   │   │   │   ├── places.ts
│   │   │   │   ├── social.ts
│   │   │   │   └── sync.ts
│   │   │   ├── llm/                   # On-device LLM (llama.cpp via llama.rn)
│   │   │   │   ├── llamaContext.ts    # LlamaContext singleton
```

```
│   │   │   │   ├── modelManager.ts      # Download, cache, load GGUF model
│   │   │   │   ├── promptBuilder.ts     # Assemble trip-context system prompt
│   │   │   │   └── chatEngine.ts        # Streaming inference + message management
│   │   │   ├── hooks/
│   │   │   │   ├── useTrip.ts
│   │   │   │   ├── useOfflineSync.ts
│   │   │   │   ├── useLocation.ts
│   │   │   │   └── useLLM.ts            # LLM loading + inference hook
│   │   │   ├── types/                   # Shared TypeScript interfaces and enums
│   │   │   ├── db/                      # SQLite (expo-sqlite) offline schema +
queries
│   │   │   │   ├── schema.ts
│   │   │   │   └── offlineQueries.ts
│   │   │   └── utils/                   # formatDate.ts, distanceCalc.ts, syncDiff.ts
│   │   ├── app.json
│   │   ├── babel.config.js
│   │   ├── tailwind.config.js
│   │   └── package.json
│   │
│   └── backend/                         # ElysiaJS API (Bun runtime)
│       ├── src/
│       │   ├── modules/
│       │   │   ├── auth/
│       │   │   │   ├── index.ts         # Elysia route controller
│       │   │   │   ├── service.ts       # Business logic
│       │   │   │   └── model.ts         # TypeBox schemas
│       │   │   ├── user/
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
│       │   │   │   └── model.ts
│       │   │   ├── trip/
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
│       │   │   │   └── model.ts
│       │   │   ├── itinerary/
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
│       │   │   │   └── model.ts
│       │   │   ├── destination/
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
│       │   │   │   └── model.ts
│       │   │   ├── place/
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
│       │   │   │   └── model.ts
│       │   │   ├── offline/
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
│       │   │   │   └── model.ts
│       │   │   ├── social/              # Phase 4
│       │   │   │   ├── index.ts
│       │   │   │   ├── service.ts
```

```
│   │   │   │   │   └── model.ts
│   │   │   │   └── message/              # Phase 4
│   │   │   │       ├── index.ts
│   │   │   │       ├── service.ts
│   │   │   │       └── model.ts
│   │   │   ├── utils/
│   │   │   │   ├── logger/
│   │   │   │   │   └── index.ts          # Structured logger (no console.log in app
code)
│   │   │   │   ├── config/
│   │   │   │   │   └── index.ts          # TypeBox-validated env vars — fail fast on
missing
│   │   │   │   └── redis/
│   │   │   │       └── index.ts          # Upstash Redis client
│   │   │   ├── middleware/
│   │   │   │   ├── auth.ts
│   │   │   │   ├── errorHandler.ts
│   │   │   │   └── rateLimiter.ts
│   │   │   ├── cron/
│   │   │   │   └── packGenerator.ts      # Nightly: rebuild offline city packs
│   │   │   └── app.ts                    # Elysia app + plugin registration
│   │   ├── server.ts                     # Bun entry point
│   │   └── package.json
│
├── packages/
│   ├── database/                         # Shared Prisma schema + client
│   │   ├── prisma/
│   │   │   ├── schema.prisma
│   │   │   └── migrations/
│   │   └── package.json
│   └── shared-types/                     # Types shared between mobile + backend
│       ├── src/index.ts
│       └── package.json
│
├── turbo.json
├── pnpm-workspace.yaml
└── package.json
```

3.1 Elysia Module Pattern

Every backend feature is a self-contained Elysia plugin registered in app.ts. Follow this pattern exactly for every module. All Prisma calls live exclusively in service.ts — never in index.ts.

```
// apps/backend/src/modules/auth/

// ── model.ts  ── TypeBox schemas (request + response types) ──────────
import { t } from 'elysia'

export const RegisterBody = t.Object({
  email:    t.String({ format: 'email' }),
  password: t.String({ minLength: 8 }),
  name:     t.String({ minLength: 1, maxLength: 100 }),
})

export const AuthResponse = t.Object({
  user:        t.Object({ id: t.String(), email: t.String(), name: t.String() }),
  accessToken: t.String(),
})

// ── service.ts  ── Business logic + all Prisma calls ─────────────────
import { db } from '@repo/database'
import bcrypt from 'bcryptjs'

export const authService = {
  async register(email: string, password: string, name: string) {
    const user = await db.user.create({ data: { email, name } })
    await db.password.create({
      data: { userId: user.id, passwordHash: await bcrypt.hash(password, 12) }
    })
    return user
  },
}

// ── index.ts  ── Elysia controller ──────────────────────────────────
import Elysia from 'elysia'
import { RegisterBody, AuthResponse } from './model'
import { authService } from './service'

export const authModule = new Elysia({ prefix: '/auth' })
  .post('/register', async ({ body }) => {
    const user = await authService.register(body.email, body.password, body.name)
    return { user, accessToken: generateToken(user.id) }
  }, { body: RegisterBody, response: AuthResponse })
```

3.2 eden-tanstack-react-query Client Pattern

The mobile app uses the eden-tanstack-react-query package — a single integration that wires Eden Treaty's type-safe client directly into TanStack Query's hook system. Import the generated api object and call .useQuery() or .useMutation() directly on any route. No manual useQuery wrappers are needed.

```
// apps/mobile/src/api/client.ts
// Uses the 'eden-tanstack-react-query' package for type-safe,
// query-cached API calls generated from the ElysiaJS App type.

import { createEdenTreatyReactQuery } from 'eden-tanstack-react-query'
import type { App } from 'backend'              // backend exports App type

export const api = createEdenTreatyReactQuery<App>(
  process.env.EXPO_PUBLIC_API_BASE_URL!,
)

// ─────────────────────────────────────────────────────────────────
// apps/mobile/src/api/trips.ts
// All queries + mutations use the generated api object directly.
// No manual useQuery/useMutation wrappers needed.

// In component:
const { data, isLoading } = api.trips.get.useQuery()

const createTrip = api.trips.post.useMutation({
  onSuccess: () => api.trips.get.invalidate(),
})

// Calling a mutation:
createTrip.mutate({
  destinationId: 'dest_123',
  title: 'Tokyo Adventure',
  startDate: '2025-04-01T00:00:00Z',
  endDate:   '2025-04-10T00:00:00Z',
})
```

4. Data Models

All models defined in packages/database/prisma/schema.prisma. The agent must implement these exactly. Fields marked Phase 4 should be added via migration at the start of Phase 4 only.

```
// packages/database/prisma/schema.prisma
//
// Auth is split into two tables:
//   User         — core profile (no credentials)
//   OAuthAccount — one row per OAuth provider per user
//   Password     — one row per email/password user (optional)
//
// This allows one user to link multiple OAuth providers without
// nullable fields cluttering the User model.
```

4.1 User

```
model User {
  id           String   @id @default(cuid())
  email        String   @unique
  passwordHash String?
  name         String
  avatarUrl    String?
  bio          String?  @db.VarChar(300)
  socialOptIn  Boolean  @default(false) // Phase 4: nearby traveler visibility
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt

  // Relations
  oauthAccounts       OAuthAccount[]
  trips               Trip[]
  chatMessages        ChatMessage[]
  sentMessages        Message[]      @relation("SentMessages") // Phase 4
  receivedMessages    Message[]      @relation("ReceivedMessages") // Phase 4
  connections         Connection[]   @relation("ConnectionRequester") // Phase 4
  receivedConnections Connection[]   @relation("ConnectionReceiver") // Phase 4
}
```

4.2 OAuthAccount (one row per provider per user)

```
// One row per OAuth provider per user.
// A user can have both a Google and Apple account linked.
model OAuthAccount {
  id           String        @id @default(cuid())
  userId       String
  provider     OAuthProvider
  providerUserId String              // Google sub / Apple sub
  createdAt    DateTime      @default(now())

  user         User          @relation(fields: [userId], references: [id], onDelete:
Cascade)

  @@unique([provider, providerUserId])
  @@index([userId])
}

enum OAuthProvider {
  GOOGLE   // Android only
  APPLE    // iOS only
}
```

4.3 Trip

```
model Trip {
  id             String          @id @default(cuid())
  userId         String
  destinationId  String
  title          String
  startDate      DateTime
  endDate        DateTime
  coverImageUrl  String?
  isPublic       Boolean         @default(false)  // Phase 4
  createdAt      DateTime        @default(now())
  updatedAt      DateTime        @updatedAt

  user           User            @relation(fields: [userId], references: [id])
  destination    Destination     @relation(fields: [destinationId], references: [id])
  itineraryItems ItineraryItem[]
  chatMessages   ChatMessage[]

  @@index([userId])
}
```

4.4 Destination

```
model Destination {
  id          String   @id @default(cuid())
  name        String
  country     String
  countryCode String               // ISO 3166-1 alpha-2
  latitude    Float
  longitude   Float
  timezone    String
  packVersion Int      @default(0) // increments when offline pack is regenerated
  createdAt   DateTime @default(now())

  trips       Trip[]
  places      Place[]
  phrases     Phrase[]

  @@index([countryCode])
}
```

4.5 Place

```
model Place {
  id            String        @id @default(cuid())
  destinationId String
  name          String
  category      PlaceCategory
  description   String?       @db.VarChar(1000)
  latitude      Float
  longitude     Float
  address       String?
  imageUrl      String?
  isCurated     Boolean       @default(false)  // appears in 'Must See' list
  rating        Float?                         // 0–5

  destination   Destination   @relation(fields: [destinationId], references: [id])

  @@index([destinationId, category])
  @@index([destinationId, isCurated])
}

enum PlaceCategory {
  ATTRACTION
  RESTAURANT
  CAFE
  HOTEL
  TRANSPORT
  SHOPPING
  NATURE
  NIGHTLIFE
  OTHER
}
```

4.6 ItineraryItem

```
model ItineraryItem {
  id         String    @id @default(cuid())
  tripId     String
  placeId    String?                   // null = custom item with no linked Place
  title      String
  notes      String?   @db.VarChar(500)
  date       DateTime                  // calendar date of the activity
  startTime  String?                   // 'HH:MM' 24-hour format, optional
  endTime    String?                   // 'HH:MM' 24-hour format, optional
  order      Int                       // display order within the day
  isDone     Boolean   @default(false)
  updatedAt  DateTime  @updatedAt

  trip       Trip      @relation(fields: [tripId], references: [id], onDelete: Cascade)

  @@index([tripId, date])
}
```

4.7 Phrase

```
model Phrase {
  id              String         @id @default(cuid())
  destinationId   String
  category        PhraseCategory
  originalText    String         // text in the destination's local language
  transliteration String?        // phonetic pronunciation hint
  englishText     String

  destination     Destination    @relation(fields: [destinationId], references: [id])

  @@index([destinationId, category])
}

enum PhraseCategory {
  GREETINGS
  DIRECTIONS
  FOOD_AND_DRINK
  EMERGENCIES
  SHOPPING
  NUMBERS
  TRANSPORT
  ACCOMMODATION
  GENERAL
}
```

4.8 ChatMessage (on-device chat history synced to server)

```
// ChatMessage stores the on-device AI chat history server-side.
// The LLM runs entirely on-device (llama.rn + GGUF model).
// Messages are synced to the server for cross-device persistence.
model ChatMessage {
  id        String    @id @default(cuid())
  userId    String
  tripId    String?                    // null = general travel advice
  role      ChatRole
  content   String    @db.VarChar(4000)
  createdAt DateTime  @default(now())

  user      User      @relation(fields: [userId], references: [id])
  trip      Trip?     @relation(fields: [tripId], references: [id])

  @@index([userId, createdAt])
}

enum ChatRole {
  USER
  ASSISTANT
}
```

4.9 Connection (add in Phase 4)

```
// Add in Phase 4
model Connection {
  id          String           @id @default(cuid())
  requesterId String
  receiverId  String
  status      ConnectionStatus @default(PENDING)
  createdAt   DateTime         @default(now())

  requester User @relation("ConnectionRequester", fields: [requesterId], references:
[id])

  receiver User @relation("ConnectionReceiver", fields: [receiverId], references: [id])

  @@unique([requesterId, receiverId])
  @@index([receiverId])
}

enum ConnectionStatus {
  PENDING
  ACCEPTED
  REJECTED
}
```

4.10 Message (add in Phase 4)

```
// Add in Phase 4
model Message {
  id         String    @id @default(cuid())
  senderId   String
  receiverId String
  content    String    @db.VarChar(2000)
  readAt     DateTime?
  createdAt  DateTime  @default(now())

  sender User @relation("SentMessages", fields: [senderId], references: [id])

  receiver User @relation("ReceivedMessages", fields: [receiverId], references: [id])

  @@index([senderId, receiverId, createdAt])
}
```

PHASE 1

Foundation: Auth · Itinerary · Offline Core

5. Phase 1 — Foundation: Auth, Itinerary & Offline Core

Dates	Feb 27 – Mar 19 (3 weeks)
Goal	A user can register, create a trip, build an itinerary, and access all data offline after downloading a city pack.
Complete When	App runs on iOS + Android simulator. New user completes the full loop: register → create trip → add itinerary items → download offline pack → view itinerary offline. All R-1xx–R-4xx requirements pass with >70% test coverage.

F-101: Authentication

Platform Note OAuth Split

Google OAuth is Android-only. Use @react-native-google-signin/google-signin on the client; the backend receives an idToken and verifies it server-side.Apple OAuth is iOS-only. Use expo-apple-authentication on the client; the backend receives an identityToken and verifies it server-side.Both flows upsert into the OAuthAccount table with the appropriate provider enum. A single User can have both an APPLE and GOOGLE OAuthAccount row.

ID	Requirement	Acceptance Criteria
R-101	Register with email + password	React Hook Form + Zod validation. Email format + min 8-char password enforced. Creates User + Password rows. Issues JWT. Navigates to Home.
R-102	Sign in with Google OAuth (Android)	Google Sign-In button (Android only). Client gets idToken. POST /auth/google with idToken. Backend verifies with Google, upserts User + OAuthAccount(GOOGLE). Returns JWT. Never shown on iOS.
R-103	Sign in with Apple OAuth (iOS)	Apple Sign-In button (iOS only). Client gets identityToken via expo-apple-authentication. POST /auth/apple. Backend verifies, upserts User + OAuthAccount(APPLE). Returns JWT. Never shown on Android.
R-104	Sign in with email + password	Backend looks up Password row by user email. Failed login: generic 'Invalid email or password.' — never reveal if email exists. Rate limit: 10 req/min/IP.

R-105	Protected route enforcement	Unauthenticated navigation to any protected screen redirects to Login. JWT middleware returns 401 for missing/expired tokens.
R-106	Sign out	Clears access token from Zustand + MMKV. Clears refresh token cookie. Navigates to Onboarding.
R-107	Silent token refresh	On 401, client calls POST /auth/refresh once. If refresh succeeds, retry original request. If refresh fails, clear state and navigate to Login.

F-102: Trip Management

ID	Requirement	Acceptance Criteria
R-201	Create a trip	Required fields: destination (autocomplete, GET /destinations?q=), title (1–100 chars), start date, end date. endDate >= startDate enforced by Zod. POST /trips on save.
R-202	View trip list on Home screen	Sorted by startDate DESC. Each card: destination name + country code, title, date range, cover image or placeholder. Skeleton loader during fetch.
R-203	Edit a trip	Tap trip → trip detail. Edit opens pre-filled form. PATCH /trips/:id on save.
R-204	Delete a trip	Long-press or swipe-left → confirmation modal. DELETE /trips/:id. Hard delete, cascades to ItineraryItems.
R-205	Destination autocomplete	Debounced 300ms. GET /destinations?q= (min 2 chars). Max 10 results with name + countryCode flag. Tap to select.

F-103: Itinerary Builder

ID	Requirement	Acceptance Criteria
R-301	Itinerary organized by day	ItineraryScreen: one collapsible section per calendar day between trip start and end dates. Section header: date + item count.
R-302	Add an itinerary item	Tap '+' on a day. React Hook Form + Zod modal: title (required, 1–100), optional linked place, optional startTime/endTime ('HH:MM'), optional notes (max 500). POST /trips/:id/itinerary-items.
R-303	Reorder items within a day	Drag handle per row. Gesture Handler + Reanimated for smooth drag. On release: PATCH /trips/:id/itinerary-items/reorder.

R-304	Mark item done	Checkbox toggles isDone. PATCH /itinerary-items/:id. Done items: strikethrough + reduced opacity via Reanimated layout animation.
R-305	Delete an item	Swipe-left reveals Delete (Gesture Handler). Confirmation modal. DELETE /itinerary-items/:id.
R-306	Edit an item	Tap row opens pre-filled modal. PATCH /itinerary-items/:id on save.

F-104: Offline City Pack

ID	Requirement	Acceptance Criteria
R-401	Download a city pack	Trip detail: 'Download Offline Pack' button. GET /offline-pack/:destinationId. Response: destination, places[], phrases[], signed Mapbox tile URL, packVersion. Progress indicator (%).
R-402	Store pack data in SQLite	Places + phrases written to expo-sqlite tables. Mapbox SDK manages tile cache. Pack metadata (destinationId, packVersion, downloadedAt) stored in MMKV via offlineStore.
R-403	Fall back to local data offline	No network: all place/phrase/itinerary reads fall back to SQLite. Persistent offline banner shown on relevant screens.
R-404	Queue offline itinerary writes	Writes while offline (add/edit/delete/reorder/toggle-done) are written to SQLite immediately and queued in an outbox table. On reconnect, sync engine replays queue in order. Conflict: server wins on deletes; client wins on edits.
R-405	Detect pack updates	On reconnect: GET /destinations/:id/pack-version. If packVersion > local: banner 'Updated city pack available. Tap to update.'
R-406	Delete a pack	Settings: list of downloaded packs with size. Swipe to delete removes SQLite rows + Mapbox tile cache for that destination.

Phase 1 — Out of Scope

- Interactive map (Phase 2)
- Recommendations + Language guide (Phase 2)
- Travel LLM chat (Phase 3)
- Social features, push notifications (Phase 4)
- PostHog analytics (Phase 5)

Phase 1 — Build Order

1. Turborepo + pnpm workspace scaffold. Create apps/mobile, apps/backend, packages/database, packages/shared-types.
2. packages/database: Prisma schema (User, Password, OAuthAccount, Trip, Destination, Place, ItineraryItem, Phrase — Phase 1 fields only). Run initial migration.
3. Seed data: Destination + Place + Phrase for at least 5 cities.
4. Auth module (backend): register, login, /auth/google (Android), /auth/apple (iOS), refresh, logout.
5. Auth UI (mobile): Onboarding screen, Login, Register. React Hook Form + Zod. Google Sign-In button (Android only). Apple Sign-In button (iOS only).
6. Trip module (backend) + UI (mobile): CRUD + destination autocomplete.
7. Itinerary module (backend) + UI (mobile): day sections, item CRUD, drag-to-reorder, swipe-to-delete.
8. Offline module (backend): GET /offline-pack/:id, GET /destinations/:id/pack-version.
9. Offline UI + SQLite schema + outbox sync engine + offline banner.
10. Tests for all routes and utility functions.

PHASE 2

Discovery: Map · Recommendations · Language Guide

6. Phase 2 — Discovery: Map, Recommendations & Language Guide

Dates	Mar 20 – Apr 2 (2 weeks)
Prerequisite	All Phase 1 requirements pass and test coverage is above 70%.
Goal	Users can explore on an interactive offline-capable map, browse curated local recommendations, and access a full language cheat sheet — all offline-capable.

F-201: Interactive Map

ID	Requirement	Acceptance Criteria
R-501	Map renders place pins	MapScreen: Mapbox map centered on active trip's destination. Place records shown as color-coded pins by PlaceCategory. Tap pin → Reanimated bottom sheet: name, category, description, 'Add to Itinerary' button.
R-502	Map works offline	Mapbox offline tile pack (from city pack) used when no network. Pins rendered from SQLite. Offline banner shown.
R-503	User location dot	With permission: GPS blue dot via expo-location. 'My Location' button re-centers map.
R-504	Category filter chips	Chips at top toggle pin categories. Active chips highlighted. At least one category must remain active.
R-505	Itinerary mode toggle	Toggle switches between 'All Places' and 'My Itinerary'. Itinerary mode: only places linked to active trip's items.
R-506	Add to itinerary from map	Tapping 'Add to Itinerary' on a pin bottom sheet opens the itinerary item modal pre-filled with place name + placeId.

F-202: Recommendations

ID	Requirement	Acceptance Criteria
R-601	Curated places list	RecommendationsScreen: all Places where isCurated = true, organized into category tabs. Each card: name, image, description snippet, rating.

R-602	Offline recommendations	Curated places served from SQLite after pack download. 'Offline' badge on cards when disconnected.
R-603	Add to itinerary from card	Each card: 'Add to Itinerary' button → itinerary item modal pre-filled.
R-604	Place detail screen	Tap card → PlaceDetailScreen: full description, address, rating, mini-map preview, photos, add-to-itinerary button.
R-605	Client-side search	Search bar filters cards in real-time (no API call). Filters on name + description. Shows result count and 'No results' empty state.

F-203: Language Guide

ID	Requirement	Acceptance Criteria
R-701	Phrases by category	LanguageScreen: Phrase records for active trip's destination, organized by PhraseCategory tabs.
R-702	Phrase card layout	Original local-language text (large, bold) · transliteration in italics (muted) · English translation.
R-703	Client-side search	Search bar filters across all three text fields. Case-insensitive. Shows result count.
R-704	Fully offline	All phrase data from SQLite after pack download. Offline banner when disconnected.
R-705	Favorite phrases	Heart icon per card. Favorites persisted in MMKV (not synced to server). 'Favorites' tab filters to saved phrases.

Phase 2 — Build Order

11. Wire Mapbox offline tile download into Phase 1 city pack flow.
12. MapScreen: pin rendering, category filter chips, Reanimated bottom sheet, user location dot.
13. Map ↔ itinerary integration: itinerary mode toggle, add-to-itinerary from map.
14. RecommendationsScreen + PlaceDetailScreen + SQLite offline fallback.
15. LanguageScreen + search + MMKV favorites.
16. Tests for all new screens and interactions.

PHASE 3

Intelligence: On-Device Travel LLM Chat

7. Phase 3 — Intelligence: On-Device Travel LLM Chat

Dates	Mar 27 – Apr 9 (2 weeks, overlaps end of Phase 2)
Prerequisite	All Phase 2 requirements pass. Fine-tuned Qwen 2.5 GGUF checkpoint available and hosted on CDN for download.
Goal	Users can have a contextual AI conversation about their trip. The LLM runs entirely on-device via llama.rn. No server inference endpoint exists. Works fully offline after the model is downloaded.

⚠️ The Travel LLM runs ENTIRELY on the user's device. There is NO backend inference endpoint. The model is downloaded from a CDN URL once and stored locally. All inference is performed by llama.rn (React Native llama.cpp bindings). Chat history is synced to the server only for cross-device persistence — not for inference.

F-301: Model Download & Management

ID	Requirement	Acceptance Criteria
R-801	Model download flow	On first visit to ChatScreen: if model not present in FileSystem.documentDirectory, show 'Download AI Model (~800MB)' prompt with estimated size. User must explicitly confirm. Download via expo-file-system with progress indicator (%). Model stored at documents/models/<filename>.gguf.
R-802	Download resumes on interruption	Use FileSystem.createDownloadResumable. If download is interrupted, user can resume from where it stopped. Download state persisted in MMKV.
R-803	Model loading state	After download, model is loaded into llama.rn context (LlamaContext). Loading indicator shown. Loaded context cached as singleton for the session (getLlamaContext()).
R-804	User can delete the model	Settings → 'AI Model' section shows model size + download date. 'Delete Model' button removes the GGUF file and clears the LlamaContext. Frees device storage.

F-302: Chat User Experience

ID	Requirement	Acceptance Criteria
R-901	Chat accessible from tab bar	ChatScreen in bottom tabs. Message thread oldest-to-newest. Input bar pinned above keyboard (KeyboardAvoidingView).
R-902	Trip context in system prompt	System prompt includes: user name, destination name, trip dates, summarized itinerary (item titles + dates). Max 2000 tokens of context assembled by promptBuilder.ts. Injected into every completions call.
R-903	On-topic guardrail	System prompt instructs model to only discuss travel, destinations, food, culture, logistics, trip planning. Off-topic: 'I'm your travel companion — I'm best equipped to help with travel questions.'
R-904	Streaming response bubble	Inference is streamed token-by-token via llama.rn's completion callback. Each token appended to the in-progress bubble in real time using Reanimated. Typing indicator shown while awaiting first token.
R-905	Chat history persisted locally	Messages stored in MMKV (immediate, synchronous). Synced to server via POST /chat/messages in batches after each assistant response. Last 10 messages from MMKV included in every new LLM context window.
R-906	Clear chat history	Settings → 'Clear Chat History'. Confirmation modal. Clears MMKV chat cache + calls DELETE /chat/history on server.
R-907	Model not downloaded state	If model not downloaded: show download prompt instead of chat UI. Do not show an empty or broken chat interface.
R-908	Works fully offline	After model is downloaded, the entire chat feature works with no network. Sync to server is deferred until reconnect. Offline banner shown but chat input remains enabled.

Phase 3 — Build Order

17. Fine-tune Qwen 2.5 on travel dataset (ML engineer deliverable: GGUF Q4_K_M model file hosted on CDN).
18. modelManager.ts: download, resume, progress, delete.
19. llamaContext.ts: LlamaContext singleton initialization.
20. promptBuilder.ts: assemble trip-context system prompt (≤2000 token budget).
21. chatEngine.ts: streaming inference via llama.rn completion callback.
22. Backend: POST /chat/messages (batch sync), GET /chat/messages, DELETE /chat/history.
23. ChatScreen UI: download prompt, model loading state, message list, input bar, streaming bubble, offline banner.
24. useLLM.ts hook: exposes model status (not-downloaded | downloading | loading | ready | error) and streamChat().
25. Tests for chat sync routes (backend). Tests for promptBuilder and modelManager utilities.

PHASE 4

Social: Connect with Travelers · Push Notifications · Sync

8. Phase 4 — Social: Connect with Travelers & Sync

Dates	Apr 10 – Apr 23 (2 weeks)
Prerequisite	All Phase 3 requirements pass. Run Phase 4 DB migrations before writing any feature code.
Migrations	Add Connection table · Add Message table · Add isPublic + socialOptIn fields to User/Trip.
Goal	Users can opt in to see other travelers at the same destination, send connection requests, and message each other. Background sync ensures offline edits replay correctly.

F-401: Nearby Travelers

ID	Requirement	Acceptance Criteria
R-1001	Social discovery opt-in	Settings toggle: 'Show me to other travelers nearby'. Default OFF. When OFF, user never appears in results. Persisted in User.socialOptIn via PATCH /users/me.
R-1002	Nearby travelers list	SocialScreen: opt-in users whose active trip destination matches the caller's. Shows: avatar, name, bio snippet, days remaining.
R-1003	Send connection request	Tap traveler card → profile modal → 'Connect' → POST /connections. Button changes to 'Pending' while PENDING.
R-1004	Accept / reject requests	'Requests' tab: incoming PENDING connections. Accept → PATCH /connections/:id {ACCEPTED}. Reject → REJECTED. Rejected users not shown in nearby list again.
R-1005	View accepted connections	'Connections' tab: ACCEPTED connections with a 'Message' button each.

F-402: Messaging

ID	Requirement	Acceptance Criteria
R-1101	Send messages to connections	Tap 'Message' → thread screen. Input bar + message list. POST /messages. Both parties must have ACCEPTED connection (403 otherwise).
R-1102	Read receipts	Opening a thread calls PATCH /messages/mark-read. Sender sees 'Read' label once readAt is set.

R-1103	Poll for new messages	Thread polls GET /messages?connectionId=:id every 10 seconds when open. // SPEC QUESTION: evaluate WebSocket upgrade if polling UX is unacceptable.
R-1104	Push notifications for messages	New message + app backgrounded → push notification via Expo Notifications. Shows sender name + 60-char preview. Tap opens thread.

F-403: Background Sync

ID	Requirement	Acceptance Criteria
R-1201	Auto-sync on reconnect	On network restore, sync engine replays outbox queue. Banner: 'Syncing...' → 'All changes saved.'
R-1202	Handle server-side deletions	GET /sync?lastSyncAt= returns items deleted server-side since lastSyncAt. SQLite removes those records.
R-1203	Conflict resolution is transparent	Server wins on structural data. Client wins on itinerary item edits. Toast: 'Some changes were updated by another device.'

Phase 4 — Build Order

26. Run Phase 4 migrations.
27. User module: PATCH /users/me (socialOptIn). Social module: GET /social/nearby, POST /connections, PATCH /connections/:id, GET /connections.
28. SocialScreen UI: Nearby, Requests, Connections tabs.
29. Message module: POST/GET /messages, PATCH /messages/mark-read.
30. Message thread screen + 10s polling.
31. Push notification setup: Expo Notifications + backend trigger on new message.
32. Enhanced sync: GET /sync endpoint + server-side deletion handling.
33. Tests for all social and sync routes.

PHASE 5

Polish: PostHog Analytics · Performance · CDN · QA

9. Phase 5 — Polish: Analytics, Performance & QA

Dates	Apr 24 – Apr 30 (1 week)
Prerequisite	All Phase 4 requirements pass.
Goal	PostHog analytics integrated, app meets all performance targets, city pack downloads are fast via CDN, crash reporting active, all known bugs fixed, demo-ready.

F-501: PostHog Analytics

Privacy Note PostHog Integration

Never send PII (name, email, address) to PostHog. Only send userId (as a distinct ID), event names, and non-identifying properties such as feature flags and boolean context. PostHog is initialized after the user explicitly accepts the privacy policy.

ID	Requirement	Acceptance Criteria
R-1301	PostHog initialized on app start	posthog-react-native SDK initialized in App root. EXPO_PUBLIC_POSTHOG_API_KEY and EXPO_PUBLIC_POSTHOG_HOST from env. Identify user with userId as distinct ID (no email or name sent).
R-1302	Screen view tracking	Every screen transition calls analytics.screenView(screenName). Implemented in expo-router layout using usePathname().
R-1303	Key events tracked	All events defined in apps/mobile/src/utils/analytics.ts wrapper. Never call posthog.capture() directly in components. Tracked events: user_registered (method), trip_created, trip_deleted, itinerary_item_added, offline_pack_downloaded, llm_model_downloaded, chat_message_sent, connection_request_sent, connection_accepted.
R-1304	Server-side events	Backend emits PostHog server-side events for: pack generation completed, sync conflicts detected. Uses POSTHOG_API_KEY env var (server-side only).
R-1305	Feature flags	PostHog feature flags used to gate experimental features in future releases. posthog.isFeatureEnabled() checked in

		relevant components. // ARCH DECISION: flags evaluated client-side to avoid blocking app startup.

F-502: Performance Optimization

ID	Requirement	Acceptance Criteria
R-1401	Cold launch under 3 seconds	Measure on mid-range Android (Pixel 4a equivalent). Lazy-load non-critical screens via expo-router dynamic imports. Defer heavy SQLite queries past initial render.
R-1402	API response times under 300ms	Non-AI endpoints < 300ms at p95. Profile slow Prisma queries. Add missing DB indexes. Cache recommendations + phrases in Upstash Redis (1hr TTL).
R-1403	City pack CDN	City pack JSON + assets served from S3 behind Cloudflare CDN. Cache-Control: max-age=3600. Edge cache invalidated when packVersion increments.
R-1404	Virtualized lists	All long lists use FlashList instead of FlatList. No full re-renders on scroll.

F-503: Error Handling & QA

ID	Requirement	Acceptance Criteria
R-1501	Sentry crash reporting	All unhandled errors captured with userId + screen name. Never send PII beyond userId. Sentry initialized before PostHog.
R-1502	Error boundaries on all screens	React Native error boundaries wrap each top-level screen. Crash shows 'Something went wrong' fallback + 'Reload Screen' button.
R-1503	70% test coverage maintained	Full test suite. Coverage report must show >= 70% for routes and utilities. Fix any Phase 4 regressions.
R-1504	User acceptance testing	UAT with at least 3 non-team testers. Document findings. Fix P0 and P1 bugs before final demo.
R-1505	Non-functional requirements verified	Verify: cold launch < 3s, API p95 < 300ms, offline mode fully functional, sync replays correctly, on-device LLM works after airplane-mode, no MapScreen memory leaks.
R-1506	Runs on iOS and Android	Full feature set tested on iOS 16+ and Android 12+. Platform-specific issues documented.

Phase 5 — Build Order

34. PostHog SDK integration: analytics.ts wrapper, screen tracking, identify on login.
35. Server-side PostHog events in backend.
36. Sentry integration + error boundary wrappers.
37. Cloudflare CDN setup for S3 city pack assets + Cache-Control headers.
38. Upstash Redis caching for recommendations + phrases.
39. DB query profiling + missing index additions.
40. FlashList migration for all long lists.
41. Full test suite + coverage report.
42. UAT sessions + P0/P1 bug fixes.
43. Final production deployment + demo prep.

10. API Reference (Full)

Base URL: /api/v1

All responses JSON

Protected routes require: Authorization: Bearer <accessToken>

Timestamps: ISO 8601 UTC.

10.1 Auth Routes

```
// ── POST /auth/register ──────────────────────────────────────────
// Body:    { email: string, password: string, name: string }
// Success: 201  { user: { id, email, name }, accessToken: string }
//          Sets httpOnly refreshToken cookie (7 days)
// Error:   409  Email already registered
//          422  Validation failure — field-level errors in details

// ── POST /auth/login ─────────────────────────────────────────────
// Body:    { email: string, password: string }
// Success: 200  { user: { id, email, name }, accessToken: string }
// Error:   401  'Invalid email or password.'  (never reveal if email exists)
// Limit:   10 requests / min / IP  →  429

// ── POST /auth/google  (Android only) ────────────────────────────
// Body:    { idToken: string }  (from @react-native-google-signin/google-signin)
// Flow:    Client obtains idToken from Google Sign-In SDK on Android.
//          Backend verifies idToken with Google, upserts User + OAuthAccount
//          (provider=GOOGLE), issues JWT.
// Success: 200  { user, accessToken }  — sets refreshToken cookie
// Error:   401  Invalid Google ID token

// ── POST /auth/apple  (iOS only) ─────────────────────────────────
// Body:    { identityToken: string, fullName?: string }
//          (from expo-apple-authentication on iOS)
// Flow:    Client performs Apple Sign-In, sends identityToken to backend.
//          Backend verifies token with Apple, upserts User + OAuthAccount
//          (provider=APPLE), issues JWT.
// Success: 200  { user, accessToken }  — sets refreshToken cookie
// Error:   401  Invalid Apple identity token

// ── POST /auth/refresh ───────────────────────────────────────────
// No body. Reads refreshToken cookie.
// Success: 200  { accessToken: string }
// Error:   401  Missing / expired — client must re-login

// ── POST /auth/logout ────────────────────────────────────────────
// Protected. Clears cookie. Removes refresh token hash from DB.
// Success: 200  { message: 'Logged out' }
```

10.2 Trip Routes (all protected)

```
// All routes protected — require Authorization: Bearer <accessToken>

// ── GET  /trips ──────────────────────────────────────────────────────
// Returns all trips for the authenticated user, sorted by startDate DESC.
// Includes: destination name, countryCode, coverImageUrl.
// Success: 200  { trips: Trip[] }

// ── POST /trips ──────────────────────────────────────────────────────
// Body:    { destinationId: string, title: string (1-100),
//            startDate: ISO8601, endDate: ISO8601 }
// Constraint: endDate ≥ startDate
// Success: 201  { trip: Trip }

// ── GET  /trips/:id ──────────────────────────────────────────────────
// Returns single trip + all itinerary items (sorted by date ASC + order ASC).
// Error:   403 if authenticated user does not own the trip.

// ── PATCH /trips/:id ─────────────────────────────────────────────────
// Body:    Partial<{ title, startDate, endDate, coverImageUrl }>
// Success: 200  { trip: Trip }

// ── DELETE /trips/:id ────────────────────────────────────────────────
// Hard delete. Cascades to all ItineraryItems.
// Success: 200  { message: 'Trip deleted' }

// ── GET  /destinations?q= ────────────────────────────────────────────
// Autocomplete. Min 2 chars. Returns max 10 Destination records.
// Success: 200  { destinations: [{ id, name, country, countryCode }] }

// ── GET  /destinations/:id/pack-version ──────────────────────────────
// Returns { packVersion: number, updatedAt: ISO8601 }
// Client polls this on reconnect to detect stale offline packs.
```

10.3 Itinerary Routes (all protected)

```
// All routes protected.

// ── GET  /trips/:id/itinerary-items ──────────────────────────────
// Success: 200  { items: ItineraryItem[] }  (sorted: date ASC, order ASC)

// ── POST /trips/:id/itinerary-items ──────────────────────────────
// Body:    { title: string (1-100),
//            date: ISO8601,
//            placeId?:   string,
//            startTime?: string ('HH:MM'),
//            endTime?:   string ('HH:MM'),
//            notes?:     string (max 500 chars) }
// Success: 201  { item: ItineraryItem }

// ── PATCH /itinerary-items/:id ───────────────────────────────────
// Body:    Partial<ItineraryItem fields + isDone>
// Must own the item via trip ownership check.
// Success: 200  { item: ItineraryItem }

// ── DELETE /itinerary-items/:id ──────────────────────────────────
// Hard delete. Must own item.
// Success: 200  { message: 'Item deleted' }

// ── PATCH /trips/:id/itinerary-items/reorder ─────────────────────
// Body:    { items: [{ id: string, order: number, date: ISO8601 }] }
// Bulk-updates order (and optionally date) for a set of items.
// All items must belong to the trip.
// Success: 200  { updated: number }
```

10.4 Offline Pack Routes (all protected)

```
// ── GET  /offline-pack/:destinationId ────────────────────────────
// Protected. Returns complete city pack JSON.
// Response: {
//   destination: Destination,
//   places:      Place[],
//   phrases:     Phrase[],
//   mapTileUrl:  string   // signed S3/CDN URL for Mapbox .db tile pack
//   packVersion: number
// }
// Headers: Content-Encoding: gzip
// Cache:   Cloudflare CDN (Phase 5). Busted when packVersion increments.

// ── GET  /sync?lastSyncAt= ───────────────────────────────────────
// Protected.
// Query:   lastSyncAt: ISO8601 — timestamp of last successful sync
// Returns: { deletedItems: [{ type: 'ItineraryItem' | ... , id: string }] }
// Client removes matching SQLite records after replaying outbox.
```

10.5 Place / Recommendation Routes (all protected)

```
// ── GET  /destinations/:id/places ──────────────────────────────
// Query params:
//   category?:  PlaceCategory  (filter by category)
//   isCurated?: boolean        (true = recommendations list only)
// Success: 200  { places: Place[] }

// ── GET  /places/:id ───────────────────────────────────────────
// Returns single Place with full details.
// Success: 200  { place: Place }
```

10.6 Chat Sync Routes (all protected, Phase 3)

```
// The Travel LLM runs on-device — no inference API routes exist.
// These routes only handle SYNCING chat history to the server
// for cross-device persistence.

// ── GET  /chat/messages ────────────────────────────────────────
// Protected.
// Query:  tripId? — filter to a specific trip's chat thread
// Returns last 100 ChatMessage rows for user, oldest first.
// Success: 200  { messages: ChatMessage[] }

// ── POST /chat/messages ────────────────────────────────────────
// Protected. Stores a batch of messages after an on-device chat session.
// Body:   { messages: [{ role: ChatRole, content: string, tripId?: string,
//                        createdAt: ISO8601 }] }
// Success: 201  { synced: number }
// Constraint: max 50 messages per batch. Content max 4000 chars.

// ── DELETE /chat/history ───────────────────────────────────────
// Protected. Hard deletes all ChatMessage rows for the user.
// Also clears on-device MMKV chat cache (handled client-side).
// Success: 200  { message: 'Chat history cleared' }
```

10.7 Social + Messaging Routes (all protected, Phase 4)

```
// All routes protected. Phase 4 only.

// ── GET  /social/nearby ──────────────────────────────────────────
// Returns users who:
//   - Have socialOptIn = true
//   - Have an active trip at the same destination as the caller
//   - Are NOT the caller
//   - Have no REJECTED connection with the caller
// Success: 200  { travelers: [{ id, name, avatarUrl, bio, daysRemaining }] }

// ── POST /connections ────────────────────────────────────────────
// Body:    { receiverId: string }
// Creates a PENDING Connection.
// Error:   409 if Connection between these two users already exists.
// Success: 201  { connection: Connection }

// ── PATCH /connections/:id ───────────────────────────────────────
// Body:    { status: 'ACCEPTED' | 'REJECTED' }
// Only the receiver may call this endpoint (403 otherwise).
// Success: 200  { connection: Connection }

// ── GET  /connections ────────────────────────────────────────────
// Returns accepted connections for the authenticated user.
// Success: 200  { connections: [{ id, user: { id, name, avatarUrl } }] }

// ── POST /messages ───────────────────────────────────────────────
// Body:    { receiverId: string, content: string (1-2000 chars) }
// Both parties must have an ACCEPTED Connection (403 otherwise).
// Success: 201  { message: Message }

// ── GET  /messages?connectionId= ─────────────────────────────────
// Returns messages in thread (both directions), oldest first.
// Success: 200  { messages: Message[] }

// ── PATCH /messages/mark-read ────────────────────────────────────
// Body:    { connectionId: string }
// Sets readAt = now() on all unread messages where receiverId = caller.
// Success: 200  { updated: number }
```

10.8 PostHog Server Events Reference (Phase 5)

```
// apps/mobile/src/utils/analytics.ts
// Wrapper around PostHog React Native SDK.
// All events defined here — never call posthog.capture() directly in components.

import PostHog from 'posthog-react-native'

export const analytics = {
  // Auth
  register:         (method: 'email' | 'google' | 'apple') ⇒
    posthog.capture('user_registered', { method }),

  // Trips
  tripCreated:      (destinationId: string) ⇒
    posthog.capture('trip_created', { destinationId }),
  tripDeleted:      () ⇒ posthog.capture('trip_deleted'),

  // Itinerary
  itemAdded:        (hasPlace: boolean) ⇒
    posthog.capture('itinerary_item_added', { hasPlace }),

  // Offline
  packDownloaded:   (destinationId: string) ⇒
    posthog.capture('offline_pack_downloaded', { destinationId }),

  // LLM Chat
  modelDownloaded:  () ⇒ posthog.capture('llm_model_downloaded'),
  chatMessageSent:  (hasTripContext: boolean) ⇒
    posthog.capture('chat_message_sent', { hasTripContext }),

  // Social (Phase 4)
  connectionSent:   () ⇒ posthog.capture('connection_request_sent'),
  connectionAccepted: () ⇒ posthog.capture('connection_accepted'),

  // Screens
  screenView:       (screen: string) ⇒
    posthog.screen(screen),
}
```

11. UI/UX Guidelines

Travel-inspired palette. All UI in React Native with NativeWind. No StyleSheet.create() except where NativeWind cannot achieve the result. Animations via Reanimated v3. Gestures via Gesture Handler.

11.1 Navigation Structure

- Bottom tab bar (5 tabs): Home · Map · Recommendations · Chat · Social
- Stack navigation within each tab — implemented via expo-router file-based routes
- Global modals (trip creation, itinerary item creation) at root layout level
- All confirmation prompts use a custom Modal component from react-native-reusables — never Alert.alert() for destructive actions

11.2 Color System (Light Mode)

Token	Hex	Usage
primary	#3B82F6	Bright travel-blue used for buttons and highlights
accent	#2563EB	Softer secondary blue (badges, subtle highlights)
accent2	#1E3A8A	Deep navy-blue for gradients / elevated surfaces
gold	#F5C542	Used for weather sun icon / premium highlights
page-bg	#0B1220	Main dark background
card-bg	#111827	Cards, tiles, elevated containers
border	#1F2937	Subtle outline on cards and tiles
text-primary	#F3F4F6	Main white text
text-muted	#9CA3AF	Secondary text (labels, metadata)
success	#22C55E	Offline ready / positive indicator green
warning	#F59E0B	Used for attention states
danger	#EF4444	Deep red — delete actions, errors, destructive states
code-bg	#0F172A	Code block background

11.3 Component Rules

- Use react-native-reusables Modal for ALL confirmation dialogs — never Alert.alert() for destructive actions
- Use Toast for all transient feedback: saved, deleted, synced, error, offline
- All destructive actions require a Modal with explicit Cancel and Confirm options
- Form validation: inline below each field via React Hook Form fieldState.error — never as screen-level banners
- All data-fetching states show Skeleton loaders — no blank screens or raw spinners
- Color is never the sole means of conveying state — always pair with text label or icon
- All interactive elements: min 44×44pt touch target, accessibilityLabel + accessibilityHint
- Animated transitions use Reanimated layout animations — never the core RN Animated API
- The ChatScreen model-download prompt must clearly state the download size before the user confirms

12. Error Handling Standards

12.1 API Error Response Format

Every API error must return this exact JSON shape. No exceptions.

```
{
  "error": {
    "code":     string,   // machine-readable, e.g. "VALIDATION_ERROR"
    "message":  string,   // human-readable, safe to display to users
    "details"?: any       // optional: field-level errors for form validation
  }
}
```

12.2 Standard Error Codes

Code	HTTP	Meaning
VALIDATION_ERROR	422	Input failed TypeBox / Zod validation. Include per-field details in details.
UNAUTHORIZED	401	Missing, invalid, or expired access token.
FORBIDDEN	403	Token valid but user does not own this resource.
NOT_FOUND	404	Resource does not exist or belongs to another user.
CONFLICT	409	e.g. Email already registered, connection already exists.
RATE_LIMITED	429	Too many requests. Include Retry-After header.
INTERNAL_ERROR	500	Unexpected server error. Log full error. Never expose stack traces.

12.3 Frontend Error Handling

- Network / 500: Toast 'Something went wrong. Please try again.'
- 401: auto-attempt token refresh once. If refresh fails ⟶ clear state ⟶ navigate to Login.
- 403: navigate to a 403 screen with a 'Go Home' button.
- VALIDATION_ERROR: render details field-by-field inline via React Hook Form setError.
- Never expose raw error objects, stack traces, or internal codes in the UI.
- Log all errors to Sentry in production. console.error in development only.

13. Non-Functional Requirements

13.1 Performance

- Cold launch < 3 seconds on mid-range Android (Pixel 4a equivalent).
- API response < 300ms for all non-LLM endpoints at p95 under normal load.
- City pack download for a medium city (e.g. Kyoto) < 30 seconds on 4G.
- On-device LLM first-token latency < 5 seconds on iPhone 12 / Pixel 6 equivalent.
- Required DB indexes: Trip(userId) · ItineraryItem(tripId, date) · Place(destinationId, category) · Place(destinationId, isCurated) · ChatMessage(userId, createdAt) · Message(senderId, receiverId, createdAt) · OAuthAccount(userId).
- Recommendation + phrase data cached in Upstash Redis. TTL: 1 hour.

13.2 Security

- Passwords hashed with bcrypt, cost factor 12. Password row separate from User — never on the same model.
- Access tokens expire in 15 minutes. Refresh tokens expire in 7 days.
- Refresh token hash stored in DB for server-side invalidation.
- All API routes verify the authenticated userId matches the resource owner. No IDOR vulnerabilities.
- Rate limit: login 10 req/min/IP · chat sync 50 req/user/hour.
- Sanitize all user-provided strings. Strip HTML from notes, bio, message content.
- CORS: allow only FRONTEND_URL origin in production.
- Mapbox public token restricted to map-read scopes only.
- No inference server endpoint exists. The model CDN URL (EXPO_PUBLIC_MODEL_CDN_URL) is public by design — it points to a static file download, not an inference API.

13.3 Offline & Sync

- Core features (itinerary, map, recommendations, language guide, LLM chat) work offline after city pack + model download.
- Offline writes queued and replayed in order on reconnect.
- Sync is idempotent — replaying the same operation twice must not create duplicates.
- Conflict: server wins on structural data; client wins on personal itinerary edits.

13.4 Accessibility

- All interactive elements: descriptive accessibilityLabel and accessibilityHint.
- Minimum touch target: 44×44 points.
- Color is never the sole means of conveying information — always pair with text or icon.
- All TextInput components have an accessibilityLabel.

13.5 Code Quality

- TypeScript strict mode everywhere. No implicit any.
- No console.log in production code — use the logger utility.
- All env vars accessed through the TypeBox-validated config object in apps/backend/src/utils/config/.
- MMKV for all mobile key-value storage. Never use AsyncStorage.
- Minimum 70% test coverage for utility functions and API routes.
- Three environments: development (local), staging (preview), production.

14. Environment Variables

Create .env.example with all keys and no values. Backend config validated via TypeBox at Bun startup — fail fast on any missing required variable. Never commit real values.

```
# ── Backend  (apps/backend/.env) ──────────────────────────────
DATABASE_URL=              # PostgreSQL (Neon) connection string
JWT_ACCESS_SECRET=
JWT_REFRESH_SECRET=

# Google OAuth — Android
GOOGLE_CLIENT_ID=          # Web client ID for token verification
GOOGLE_CLIENT_SECRET=

# Apple OAuth — iOS
APPLE_CLIENT_ID=           # Apple Sign-In service ID (bundle ID)
APPLE_TEAM_ID=
APPLE_KEY_ID=
APPLE_PRIVATE_KEY=         # Base64-encoded .p8 key

PORT=
FRONTEND_URL=
NODE_ENV=                  # development | staging | production

UPSTASH_REDIS_URL=
UPSTASH_REDIS_TOKEN=

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_S3_BUCKET=
AWS_REGION=

MAPBOX_SECRET_TOKEN=       # Server-side only — tile pack generation

# Phase 4+
EXPO_PUSH_ACCESS_TOKEN=    # Expo push notification credentials

# Phase 5
POSTHOG_API_KEY=           # PostHog server-side events

# ── Mobile App  (apps/mobile/.env) ────────────────────────────
EXPO_PUBLIC_API_BASE_URL=
EXPO_PUBLIC_MAPBOX_ACCESS_TOKEN=  # Public token — map rendering only
                                  # Restrict to map-read scopes

# Android Google Sign-In (client-side SDK only, not a secret)
EXPO_PUBLIC_GOOGLE_CLIENT_ID=     # Android OAuth client ID

# Phase 5
EXPO_PUBLIC_POSTHOG_API_KEY=      # PostHog client-side events
EXPO_PUBLIC_POSTHOG_HOST=         # e.g. https://app.posthog.com

# On-device model
EXPO_PUBLIC_MODEL_CDN_URL=        # CDN URL to download GGUF model file
```

15. Glossary

These terms have precise meanings in this codebase. Use them consistently in code, variable names, and comments.

Term	Definition
Destination	A city or region travelers visit. Has Places and Phrases. Seeded by the team — users cannot create destinations.
Trip	A user-created travel plan for one Destination with start and end dates.
ItineraryItem	A single scheduled activity within a Trip. Optionally linked to a Place.
Place	A physical location within a Destination. Seeded by the team.
Curated Place	A Place where isCurated = true. Appears in the Recommendations 'Must See' list.
Phrase	A translated text entry for a Destination's local language.
Offline Pack / City Pack	Downloadable bundle: Destination + Place + Phrase data for one city + Mapbox tile cache. Stored in SQLite.
packVersion	Integer counter on Destination incremented when pack data changes. Client uses this to detect stale packs.
Outbox	Local SQLite queue of writes made while offline. Replayed in order on reconnect.
OAuthAccount	One DB row per OAuth provider per user. Allows a single User to have both GOOGLE and APPLE accounts linked.
ChatMessage	A message in the on-device AI travel assistant chat. Distinct from Message (user-to-user DMs).
Message	A direct message between two connected users (Phase 4). Not AI-related.
Connection	Social relationship between two users. PENDING → ACCEPTED or REJECTED.
Nearby Travelers	Opt-in users at the same destination. In code: nearbyTravelers or connections — never just 'social'.
llama.rn	React Native bindings for llama.cpp. Used to run the GGUF model on-device.
GGUF	File format for quantized LLM weights used by llama.cpp / llama.rn.
LlamaContext	The singleton llama.rn inference context. Initialized once after model load, reused for all chat sessions.
eden-tanstack-react-query	Single npm package combining Eden Treaty (type-safe HTTP client from ElysiaJS) with TanStack Query hooks. Used for all API calls from mobile.
MMKV	react-native-mmkv. Used for all key-value storage on mobile. Replaces AsyncStorage entirely.
PostHog	Product analytics platform. Added in Phase 5. All events defined in analytics.ts wrapper — never called directly from components.

16. Agent Instructions & Constraints

Read this section before writing any code. These rules override any other instinct or default behavior.

16.1 Phase Gates

- Do NOT start Phase 2 until all Phase 1 requirements pass and test coverage is above 70%.
- Do NOT start Phase 3 until all Phase 2 requirements pass.
- Do NOT start Phase 4 until all Phase 3 requirements pass. Run Phase 4 DB migrations before writing feature code.
- Do NOT start Phase 5 until all Phase 4 requirements pass.

16.2 Build Order Within Each Phase

- Always: DB schema → API module (model + service + controller) → API tests → Frontend screen/component → Frontend tests.
- Complete each feature (backend + frontend + tests) before moving to the next feature in the same phase.
- After completing a feature, verify every requirement ID listed for it passes before moving on.

16.3 Decision Rules

- When the spec is ambiguous: implement the simpler interpretation and add `// SPEC QUESTION: [description]`
- When making an architectural decision not covered by this spec: add `// ARCH DECISION: [explanation]`
- If you believe the spec has an error: implement it as written and add `// SPEC ERROR: [concern]`
- Do not add libraries not listed in Section 2. If you must: add a `// ARCH DECISION:` comment.
- Do not implement any feature listed as Out of Scope for the current phase.

16.4 Security Rules (Non-Negotiable)

- Never hardcode secrets, user IDs, or URLs. All config from env vars validated in utils/config/index.ts.
- All DB queries scope to the authenticated userId. Never query across users.
- The model CDN URL is public (users download the GGUF file). No inference server URL exists.
- Never expose stack traces to users. All production errors return generic messages.
- MMKV for all mobile key-value storage. Never use AsyncStorage.
- Google OAuth button is Android-only. Apple OAuth button is iOS-only. Never show the wrong button on the wrong platform.

16.5 Handling Gaps

If this spec does not cover a situation, apply in order:

44. Check if a similar pattern exists elsewhere in the spec. Follow it.
45. Choose the approach most consistent with the existing architecture.
46. Choose the simpler of two reasonable approaches.
47. Leave a comment explaining your choice.

End of Specification — Travel Companion App v1.0

Team 11 · Feb 27 – Apr 30, 2025