ChatGPT

Aplicație Mobilă Turism (React Native + Expo + AI)

Scopul aplicației: Această aplicație mobilă listează locații turistice (restaurante, cafenele) dintr-un fișier JSON și oferă două moduri de vizualizare – listă și hartă – cu navigație intuitivă. Utilizatorul explorează locații, poate vizualiza detaliile fiecărei locații și primește o descriere „vibe" generată automat cu ajutorul unui API AI. Aplicația include un buton rapid către WhatsApp pentru fiecare locație (de ex. rezervări). Se implementează trei teme vizuale (Light, Dark, Pastel Mov), iar schimbarea temei se face dinamic. Documentația de față acoperă complet **user journey** (fluxul utilizatorului) și toate funcționalitățile esențiale, arhitectura tehnică, configurările și pașii de dezvoltare pentru hackathon.

User Journey (Exemplu): Utilizatorul deschide aplicația și se află într-un ecran „Explore" cu listă de locații (titlu, scurtă descriere, iconițe). El poate schimba modul de vizualizare în hartă (Map View) pentru a vedea locațiile pe o hartă OSM. Prin apăsarea pe o locație, ajunge pe un ecran de detaliu care afișează informații despre locație, precum și o descriere detaliată generată de AI. Există și un meniu jos (Bottom Tab Navigator) cu două tab-uri: „Explore" (lista/harta locațiilor) și „Profil" (setările utilizatorului). În timp ce aplicația apelează serviciul AI pentru descriere, se afișează un indicator de încărcare (spinner) [1].

Funcționalități cheie: Navigare prin file (explorare/profil) [2], afișare listă de locații cu `FlatList` (randare eficientă) [3], harta interactivă (`react-native-maps` cu tile-uri OpenStreetMap), selectare locație cu ecran detaliu, generare descriere de către API AI (de ex. OpenAI) cu sistem de prompt, buton WhatsApp, teme vizuale (Light/Dark/Pastel) cu design tokens și toggle la runtime [4]. Aplicația este implementată în TypeScript, cu standarde moderne de cod (ESLint/Prettier) [5] și rulează sub Expo Managed Workflow.

Arhitectura aplicației pe module

- **UI / Components:** Componente vizuale reutilizabile (ex. `LocationItem`, `MapView`, `ThemeSwitcher`, `LoadingIndicator`) și ecrane (paginile aplicației: `ExploreScreen`, `DetailsScreen`, `ProfileScreen` etc). Vom folosi concepte de design system (design tokens pentru culori, fonturi) [6] [7].
- **Store / State Management:** Folosim un store global (de ex. Context API sau Redux Toolkit) care păstrează lista locațiilor încărcată din JSON și starea aplicației (temă curentă, date user). Datele din JSON pot fi încărcate local sau de pe un endpoint specificat în app config.
- **API / Networking:** Modul pentru apeluri de rețea – include un serviciu care face HTTP GET pentru fișierul JSON de locații și un serviciu AI care face POST către un endpoint de generare descriere (ex: `/vibe`). Pentru apelul AI, folosim `axios` (sau fetch); de ex. un `axios.create()` configurat cu base URL OpenAI [8] și trimitem promptul de bază.
- **Map Module:** Componente legate de hartă: un ecran/map view ce folosește `react-native-maps` (via Expo) și suprapunem tile-uri OSM cu `<MapView><UrlTile urlTemplate="...osmpl/{z}/{x}/{y}.png" /></MapView>`. Se configurează corespunzător pentru a arăta harta corect (vezi [17]).
- **Themes:** Module de teme și design tokens. Vom defini un `ThemeContext` cu culori și stiluri pentru teme Light, Dark și Pastel Mov. Schimbarea temei se face la runtime prin context și un hook (ex.

`<ThemeProvider>` cu `toggleTheme`) [4] [9]. Culorile și fonturile sunt stocate în tokens (de ex. un fișier `theme.ts` cu paletă de culori, fonturi, spațieri) [6] [7].

- **AI Module:** Interacțiunea cu serviciul AI. De exemplu, un hook `useVibeGenerator` sau un serviciu `AIService` care primește o descriere scurtă și returnează un text generat. Se realizează un POST (sau `fetch`) către endpoint-ul AI cu payload (detalii în secțiunea Contract API) și un loader activat până la primirea răspunsului [10] [1].

Structura fișierelor și directoare

Organizarea poate arăta astfel:

```
/App.tsx                # Intrarea principală a aplicației (wrapping navigation
& providers)
/src
  /assets               # Imagini, fonturi, etc.
  /components           # Componente reutilizabile (ex. LocationItem, Button,
MapViewWithMarker etc.)
  /screens              # Ecrane mari (ExploreScreen, DetailsScreen,
ProfileScreen, SettingsScreen etc.)
  /navigation           # Fișiere de navigare (BottomTabsNavigator,
StackNavigator)
  /contexts             # Context API (ThemeContext, AuthContext etc.)
  /hooks                # Hooks personalizate (useLocations, useAI, useTheme,
useGeo etc.)
  /store                # Logică de state global (de ex. Redux slices sau
context)
  /services             # Servicii de rețea/API (LocationService, AIService,
WhatsAppService)
  /utils                # Funcții ajutătoare, formatare data, validări
  /theme                # Fișiere teme (design tokens, palette, themes.ts,
ThemeProvider)
  /config               # Config expo, EAS (eas.json), tsconfig, ESLint
```

Fișiere de configurare: - `app.json` / `app.config.js` – setări Expo (slug, nume, scheme pentru deep linking etc.). - `eas.json` – configurări EAS Build (profiluri de build, taste, profiles). - `tsconfig.json` – extinde `expo/tsconfig.base` [11], cu `strict: true` pentru tipizare strictă. - `.eslintrc.js` sau `eslint.config.js` – configurarea ESLint. Putem rula `npx expo lint` pentru inițializare [5]. - `.prettierrc` – configurare Prettier. - `.env` – variabile de mediu (cheie API OpenAI, etc.). - `babel.config.js` – config Babel (inclusiv plugin `react-native-dotenv` pentru variabile de mediu dacă folosim expo build).

Exemplu `tsconfig.json` (extinde expo base) [11]:

```
{
  "extends": "expo/tsconfig.base",
```

```
  "compilerOptions": {
    "strict": true,
    "baseUrl": ".",
    "paths": { "@/*": ["src/*"] }
  },
  "exclude": ["node_modules", "babel.config.js", "metro.config.js"]
}
```

Exemplu de script în `package.json`:

```
"scripts": {
  "start": "expo start",
  "android": "expo start --android",
  "ios": "expo start --ios",
  "lint": "expo lint",
  "tsc": "tsc --noEmit"
}
```

Teme vizuale și tokens de design

Vom implementa **3 teme**: Light, Dark și Pastel Mov (temă personalizată cu nuanțe de violet pastel). Fiecare temă are propriul set de tokeni: - Culori primare/secundare/neutral pentru fundal, text, accent, butoane etc. - Seturi de fonturi și mărimi de text (ex. `fontSizes.body`, `heading`, etc.). - Spațieri (padding, margin) standardizate. - Alte valori de design (e.g. radius, shadow).

Un exemplu de fișier `theme.ts` (design tokens) poate arăta astfel [6] [7]:

```
export const lightPalette = {
  background: "#FFFFFF",
  text: "#222222",
  primary: "#736DFF",   // un mov pastel
  secondary: "#A09BFF",
  accent: "#3EC55F",    // de ex. verde accent
  // ...alte culori
};
export const darkPalette = {
  background: "#121212",
  text: "#E8E7FF",
  primary: "#524DA0",
  secondary: "#8570CC",
  accent: "#3EC55F",
  // ...
};
export const pastelPalette = {
```

```
  background: "#E8E7FF",
  text: "#1D1C25",
  primary: "#A09BFF",
  secondary: "#DCDAFF",
  accent: "#3EC55F",
  // ...
};

export const fontSizes = {
  xxl: 32,
  xl: 28,
  lg: 24,
  md: 20,
  body: 17,
  sm: 14,
  xs: 12,
};

export const spacing = {
  none: 0,
  xs: 4,
  sm: 8,
  md: 12,
  lg: 16,
  xl: 20,
};

export const themes = {
  light: { colors: lightPalette, fontSizes, spacing },
  dark:  { colors: darkPalette,  fontSizes, spacing },
  pastel:{ colors: pastelPalette,fontSizes, spacing }
};
```

Mecanismul de **switch între teme la runtime** se bazează pe React Context. Putem defini un `ThemeContext` care furnizează tema curentă și o funcție `toggleTheme` [4]. De exemplu, `ThemeProvider` arată astfel [4]:

```
// ThemeContext.tsx
import React, { createContext, useState } from 'react';
import { themes } from './theme';  // tema definită mai sus

export const ThemeContext = createContext({
  theme: themes.light,
  toggleTheme: () => {}
});
```

```
export const ThemeProvider: React.FC = ({ children }) => {
  const [currentTheme, setCurrentTheme] = useState(themes.light);
  const toggleTheme = () => {
    setCurrentTheme(prev => (
      prev === themes.light ? themes.dark : themes.light
    ));
  };
  return (
    <ThemeContext.Provider value={{ theme: currentTheme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

În orice componentă copii putem folosi `useContext(ThemeContext)` pentru a obține culorile și funcția de toggle [9]. Astfel, stilurile componentei sunt dinamice în funcție de tema selectată. Tokenii definiți (culori, fontSizes etc.) pot fi folosiți în stiluri sau cu `styled-components` / `tailwind-rn` pentru consistență [6].

Imaginea de mai jos arată un exemplu de layout cu navigator cu tab-uri (Eveniment echivalent cu modul de navigare):

Exemplu de navigare Bottom Tabs (navigare prin filă). Fiecare tab are un ecran dedicat (`Explore` și `Profile`) conform recomandărilor din React Navigation [12] [2].

Navigare și componente principale

- **Navigare prin tab-uri:** Folosim React Navigation (`@react-navigation/bottom-tabs`) pentru bară de navigare jos. De ex. definim un bottom tab cu două ecrane: `ExploreScreen` și `ProfileScreen` [2]. Exemplu de utilizare:

  ```
  import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
  const Tab = createBottomTabNavigator();
  function AppTabs() {
    return (
      <Tab.Navigator>
        <Tab.Screen name="Explore" component={ExploreScreen} />
        <Tab.Screen name="Profile" component={ProfileScreen} />
      </Tab.Navigator>
    );
  }
  ```

  Aceasta creează o bară de jos cu tab-uri “Explore” și “Profile” [2]. Opțional, putem folosi iconițe (ex. din `@expo/vector-icons`) pentru fiecare tab.

- **Ecrane principale și componente:**

- `ExploreScreen`: Arată lista locațiilor (cu `FlatList` sau `SectionList`) și un buton/toggle pentru schimbare hartă/listă. Fiecare element de listă este un `LocationItem`.
- `MapScreen` (sau modul hartă în `Explore`): Folosește `react-native-maps` pentru afișarea hărții. Se suprapun markere de locații și tile-uri OSM. De ex. `<MapView><UrlTile urlTemplate="https://a.tile.openstreetmap.org/{z}/{x}/{y}.png" /></MapView>`. Puteți plasa marker pe coordonatele locației.
- `DetailsScreen`: Ecranul de detaliu al unei locații selectate. Afișează nume, imagine, scurtă descriere, plus „descriere vibe" generată de AI. Conține și un buton „Deschide în WhatsApp" care folosește `Linking.openURL("https://wa.me/...")`.
- `ProfileScreen`: Setări utilizator, inclusiv un switch de temă (Dark/Light/Pastel).
- Componente suplimentare: `LocationItem` (afișează datele din JSON), `LoadingIndicator` (ex. `<ActivityIndicator />` React Native pentru așteptare), `ThemeSwitcher` etc.

Rolul fiecărei componente: fiecare afișează informații specifice și poate fi documentată cu tipuri TypeScript. De ex., modelul TS al unei locații:

```
interface Location {
  id: string;
  name: string;
  latitude: number;
  longitude: number;
  shortDescription: string;
  // alte câmpuri (ex. category, imageUrl, contactWhatsapp, etc.)
}
```

Listarea locațiilor se face eficient cu `FlatList`, care primește `data` și un `renderItem`. FlatList „nu randărește" toate elementele simultan, ci doar cele vizibile [3], evitând probleme de performanță.

Configurații Expo, EAS, TypeScript, ESLint

- **Expo Managed Workflow:** Creăm proiectul cu `npx create-expo-app --template expo-template-blank-typescript` (sau varianta CLI) [13]. Expo gestionează aparițiile cross-platform și accesul la API-urile native. În `app.json` / `app.config.js` setăm numele aplicației, scheme, permisiuni.
- **EAS Build:** Folosim EAS pentru build-uri de producție. Configurăm `eas.json` cu profiluri pentru release. Pentru a genera **APK Android** clar, setăm în profilul de build proprietatea `android.buildType: "apk"` [14] și rulăm `eas build -p android --profile <profil>`. După finalizare putem descărca și instala APK-ul pe device/emulator [14] [15]. (De reținut: formatele implicite sunt AAB pentru Play Store; pentru APK explicit se setează profilul cum este arătat).
- **TypeScript:** Proiectul este în TS. `tsconfig.json` extinde `expo/tsconfig.base` și activăm `strict: true` pentru tipare ferite [11]. Putem defini alias-uri (ex. `"@/*": ["src/*"]`) și scripturi de verificare `npm run tsc`.

- **ESLint/Prettier:** Rulăm `npx expo lint` pentru setup inițial și instalare dependențe ESLint [5]. Folosim configurația Expo care vine gata pentru TS. De asemenea integrăm Prettier pentru stil. Putem folosi config `eslint-config-expo/flat` sau un `.eslintrc.js` / `eslint.config.js` personalizat. Exemplu script: `"lint": "expo lint"`. Documentația Expo recomandă acest mod de configurare simplu [5].
- **Testare (opțional):** Putem configura Jest + React Native Testing Library pentru teste de componente. Configurile pot proveni din `react-native-testing-library` sau `@testing-library/react-native`. Scriem teste unitare pentru funcționalitate (de ex. generatorul AI sau componentele vizuale).

Teme vizuale: design tokens și switch runtime

Așa cum s-a menționat, definim *design tokens* centrali (culori, fonturi, spațieri) într-un fișier de temă [6] [7]. De exemplu, putem avea un obiect `themes` cu 3 variante. În `ThemeProvider`, punem obiectul `theme` în context și pe baza lui stilăm componentele. Schimbarea temei poate fi declanșată de un **toggle** sau de settingul sistemului (React Native `Appearance`). Exemplu de hook pentru switch:

```
// Folosind contextul definit anterior
const { theme, toggleTheme } = useContext(ThemeContext);
...
<View style={{ backgroundColor: theme.colors.background, flex: 1 }}>
  <Text style={{ color: theme.colors.text }}>Exemplu text</Text>
  <Button title="Schimbă tema" onPress={toggleTheme} />
</View>
```

Astfel, la apăsarea butonului, se apelează `toggleTheme()` și toate componentele re-randează cu noile culori de temă [4] [9]. Tabelul de culori (light/dark/pastel) furnizează consistent designul general.

Componente și ecrane (rolul fiecăruia)

Componente de bază: - `LocationItem`: Afișează elementul din listă (nume, scurtă descriere, iconițe temă, & buton WhatsApp). Legăm onPress la `DetailsScreen`. - `MapViewComponent`: Învelitor pentru `<MapView>` care afișează marker-ele locațiilor. Folosește `<UrlTile urlTemplate="...OSM..." />` pentru tiles OSM și `<Marker>` pentru locații.
- `LoadingIndicator`: Componentă simplă care afișează `<ActivityIndicator>` centrat (folosit pe ecranul de detalii în timp ce așteptăm răspuns AI).
- `ThemeSwitcher`: Componente UI (switch/cadre) pentru schimbarea temei curente; transmite apel către `toggleTheme`.
- `BottomTabsNavigator`: Containerul navigatorului bottom tab (Explore/Profile) [12] [2].
- **Screens:** - `ExploreScreen`: Reunește lista/harta locațiilor. Poate conține comutator între mod Listă/Mapă (ex. buton în header). Include și buton “+” pentru alte acțiuni viitoare (ex. filtrare).
- `DetailsScreen`: Afișează date complete despre locație. Face o cerere AI folosind scurta descriere din JSON; afișează un spinner până la generarea textului. Exemplu schelet:

```
const DetailsScreen = ({ route }) => {
  const { location } = route.params;
  const [vibe, setVibe] = useState<string>('');
  const [loading, setLoading] = useState<boolean>(false);

  useEffect(() => {
    setLoading(true);
    generateVibe(location.shortDescription).then(text => {
      setVibe(text);
      setLoading(false);
    });
  }, []);
  return (
    <View>
      <Text>{location.name}</Text>
      {/* Imagine locație, etc. */}
      {loading ? <LoadingIndicator /> : <Text>{vibe}</Text>}
      <Button title="WhatsApp" onPress={...} />
    </View>
  );
};
```

- `ProfileScreen`: Setări utilizator (temă, cont, despre aplicație).
- Alte ecrane necesare (ex. un ecran de încărcare inițial dacă se cere, ecran de eroare, etc.).

Fiecare componentă și ecran va avea propriile **proprietăți (props)** și tipuri TS asociate. Ex: `LocationItem` primește `location: Location` și `onPress: () => void`.

Exemple de cod și modele TypeScript

- **Model date locație:**

```
export interface Location {
  id: string;
  name: string;
  latitude: number;
  longitude: number;
  shortDescription: string;
  category: 'restaurant' | 'cafe';
  imageUrl?: string;
  whatsappLink?: string;
}
```

- **Exemplu hook generare AI (vite):**

```
import axios from 'axios';
export const generateVibe = async (prompt: string): Promise<string> => {
  const instance = axios.create({
    baseURL: 'https://api.openai.com/v1/completions',
    headers: { 'Authorization': `Bearer ${process.env.OPENAI_KEY}` }
  });
  const resp = await instance.post('', {
    model: 'text-davinci-003',
    prompt: `Descrie vibrația acestui loc: ${prompt}`,
    max_tokens: 100
  });
  return resp.data.choices[0].text.trim();
};
```

Acest exemplu folosește axios pentru OpenAI (așa cum arată tutorialul pentru integrare ChatGPT în RN [16]).

- **Exemplu de componentă bazată pe tokens și context (stiluri din temă):**

```
import { useContext } from 'react';
import { ThemeContext } from './ThemeContext';
const Title = ({ text }: { text: string }) => {
  const { theme } = useContext(ThemeContext);
  return (
    <Text style={{
      color: theme.colors.primary,
      fontSize: theme.fontSizes.xl
    }}>
      {text}
    </Text>
  );
};
```

- **Config ESLint:** După `npx expo lint` se generează `eslint.config.js`. De exemplu:

```
// eslint.config.js
const { defineConfig } = require('eslint/config');
module.exports = defineConfig({
  extends: ['eslint:recommended', 'plugin:react/recommended', 'eslint-
config-expo'],
  parser: '@typescript-eslint/parser',
  plugins: ['@typescript-eslint'],
  rules: { /* reguli personalizate */ }
});
```

- **Config EAS (`eas.json`):** Exemplu profil de build pentru APK:

```
{
  "build": {
    "production": {
      "android": {
        "buildType": "apk"
      },
      "distribution": "store"
    }
  }
}
```

Și se rulează apoi: `eas build --platform android --profile production` (similar instrucțiunilor oficiale [14]).

Contract API (Endpoint `/vibe`)

Putem defini un endpoint intern sau extern pentru generarea descrierii „vibe". De exemplu, un serviciu backend la `/vibe` care primește: - **Request:**

```
{
  "locationId": "string",
  "shortDescription": "string"
}
```

- **Response:**

```
{
  "fullDescription": "string"
}
```

Aplicația client trimite un POST cu ID-ul locației și descrierea scurtă (din JSON) către `/vibe`. Serviciul apelează OpenAI/Gemini, întărește promptul și returnează descrierea generată. (Ex: `{ shortDescription: "restaurant cu specific local", ...}`). Răspunsul JSON conține textul complet. Se va evidenția token-ul de autorizare dacă e necesar.

Prompts AI recomandate

Pentru generarea automată a componentelor sau testelor cu AI (Copilot, Cursor etc.), folosim bune practici de prompt engineering [17] [18]:

- **Pentru componente:** În prompt se descrie clar componenta. Ex.: „**Scrie o componentă React Native în TypeScript numită `LocationItem` care afișează numele locației (`name`), o scurtă descriere și un buton care deschide WhatsApp folosind `Linking.openURL`**. Usează design

tokens din tema aplicației și stiluri responsive." În prompt putem oferi exemplu de props sau format al datelor.

- **Pentru ecrane:** Ex.: „**Generează un ecran `DetailsScreen` care primește o locație și afișează detaliile acesteia. Include un titlu, imagine, descriere scurtă și descriere generată de AI (te lasă comentariul să completezi partea AI)**."
- **Pentru testare:** Ex.: „**Scrie un test unit pentru componenta `LocationItem` folosind Jest și React Native Testing Library. Testul trebuie să verifice că afișează corect numele locației și că butonul WhatsApp primește link-ul potrivit din props.**"
- **Exemple de bune practici (GH Copilot):** Pornim cu descriere generală și apoi detalii [17], plus exemple de intrare-ieșire dacă este necesar [18]. Astfel AI generează un cod mai precis. De exemplu, în promptul pentru test putem da exemplu de obiect locație și așteptarea conținutului text.

Respectăm recomandările de a specifica toate cerințele clare în prompt și a da exemple. Această abordare poate accelera generarea de cod de calitate de către AI [17] [18].

Plan de dezvoltare pe pași

1. **Inițializare proiect:** Creăm proiect Expo cu template TS. Configurăm tsconfig (extends expo base) [11] și rulează `expo start`.
2. **Structură & navigare:** Implementăm navigarea Bottom Tabs (Explore, Profile) [2] [12]. Creăm ecranele goale `ExploreScreen` și `ProfileScreen`.
3. **Modul Listă Locații:** În `ExploreScreen`, importăm JSON-ul cu locații (din assets sau URL) și afișăm cu `FlatList` itemii de tip `LocationItem` (folosind date TS).
4. **Detalii Locație:** Definim `DetailsScreen`, conectăm navigația pentru a afișa detalii când utilizatorul apasă pe un element din listă.
5. **Mapă OSM:** Adăugăm un `MapScreen` sau integrăm hartă în `Explore`. Folosim `<MapView>` și `<UrlTile urlTemplate="...osm..."/>`. Testăm afișarea hărții și a markerelor (folosim Google Maps implicit, sau configurăm OSM) [19].
6. **AI Integration:** Implementăm serviciul de apel AI folosind axios (ca în exemplul dev.to [16]). În `DetailsScreen`, apelăm API-ul AI cu promptul bazat pe `shortDescription` și afișăm rezultatul. Arătăm `LoadingIndicator` până primim răspunsul [1] [10].
7. **Teme vizuale:** Definim fișierul de teme (culori, fonturi) și `ThemeProvider`. Adăugăm buton pentru toggling temă în `ProfileScreen`. Confirmăm că componentele citesc culorile din context. Testăm Light/Dark/Pastel.
8. **Completări UI:** Stilăm componenta `LocationItem`, ecranul listă, harta, ecran detalii. Adăugăm butonul WhatsApp pe detalii.
9. **Refinări și teste:** Verificăm funcționalități. Scriem teste unitare pentru componente critice. Ajustăm ESLint/Prettier.
10. **EAS Build:** Configurăm `eas.json` pentru Android/iOS. Generăm build-uri de test (ex: `eas build --profile preview`). Urmăm instrucțiunile Expo pentru generare APK [14].

TODO-uri și extensii viitoare

- **Autentificare/Cont utilizator:** Adăugarea ecran de login/signup (ex. cu Firebase/Auth0) ca sugerat de hackathon. Se poate opta și pentru „autentificare fără efort" (ex. login link prin WhatsApp/E-mail). Un exemplu de integrare AI în formularul de login (sugestii smart de email) este descris în [30].
- **Filtre și căutare:** În ecranul de explorare, adăugarea de filtre (ex. tip locație) sau căutare text.

- **Chatbot de asistență:** Un ecran de chat folosind GPT (de ex. Chatbot suport client) – poate fi un feature adițional.
- **Sisteme de rating și recenzii:** Gestionarea feedback-ului utilizatorilor.
- **Tratamentul erorilor și mesaje UI:** Tratăm cazurile de eroare (ex. eșec la rețea, AI nereușit) și afișăm mesaje prietenoase.
- **Animații și tranziții:** Pentru tranziții între ecrane (ex. fade-in pe hartă, efecte la schimbarea temei) putem folosi `react-native-reanimated` sau `LayoutAnimation`.
- **Optimizări de performanță:** Lazy load imagini, memoizarea componentelor heavy.
- **Îmbunătățiri UI/UX:** Iconițe, splashscreen personalizat, localizare (multilanguage), butoane de share.

Sfaturi EAS și generare APK

- Folosiți **EAS CLI** instalat global (`npm i -g eas-cli`).
- Asigurați-vă că `eas.json` conține profile clare (ex.: `development`, `preview`, `production`). Pentru APK, folosiți opțiunea `android.buildType: "apk"` [14].
- Pentru Android, configurați semnăturile: pentru build de producție, furnizați cheile de semnare (SHA-1) în Google Cloud și Play Console, conform instrucțiunilor Expo [20].
- Verificați periodic simulările locale cu emulator (`expo go` sau `npx react-native run-android` în dev mode).
- Pregătiți spațiul necesar pentru așteptarea unui build (~15-20 minute).
- După build, folosiți `eas build:run` pentru a instala automat pe emulator/dispozitiv [21].
- Când totul este gata, rulați `eas build -p android -p ios --profile production`. Nu uitați să lăsați timp pentru eventuale corecții de build înainte de deadline.

Checklist de livrare

- [ ] **Build stabil:** Aplicația funcționează pe Android (APK) și/sau iOS.
- [ ] **Toate funcționalitățile:** Lista locațiilor, harta OSM, detalii AI, navigare tab, theme switch, buton WhatsApp.
- [ ] **Teste trecute:** Unit și/sau integrări esențiale scrise și rulate cu succes.
- [ ] **Lint/Jest:** Fără erori ESLint și fără erori de build TypeScript.
- [ ] **Release config:** `eas.json` corect configurat (profiluri și taste), chei Firebase/API, etc.
- [ ] **App Store:** Capturi de ecran și descriere pregătite (pentru Android Play Store/Apple App Store).
- [ ] **Crash/Analitică:** Dacă este cazul, integrați bug-reporting (ex. Sentry) și Google Analytics/Segment pentru statistici (recomandare generală) [22].
- [ ] **Documentație:** Ghid de deploy/rulare și mai sus – acest document poate fi folosit ca referință.
- [ ] **Adoptare feedback:** S-au corectat eventualele bug-uri sau cereri suplimentare primite în review.
- [ ] **Buffer timp:** Rețineți că iOS necesita aprobare în App Store (pot dura zile, recomandăm să lăsați 1-2 săptămâni rezervă) [23].

Folosiți această documentație ca ghid complet pentru dezvoltarea aplicației. Informațiile și exemplele date (cu referințele incluse) servesc atât ca roadmap de dezvoltare, cât și ca input pentru generare automată asistată de AI (Copilot, Cursor, etc.), permițând crearea rapidă și coerentă a întregului proiect.

Surse: Documentația Expo (TypeScript, ESLint, Maps, EAS) [11] [5] [14], React Native docs (FlatList) [3], React Navigation (Bottom Tabs) [2] [12], integrare OpenAI în RN [16], teme și design tokens [6] [4], ghid prompt Copilot [17] [18], sugestii AI login, checklist release [22] [23].

---

[1] reactjs - show loading when fetching data react - Stack Overflow
https://stackoverflow.com/questions/69526519/show-loading-when-fetching-data-react

[2] Bottom Tabs Navigator | React Navigation
https://reactnavigation.org/docs/bottom-tab-navigator/

[3] Using List Views · React Native
https://reactnative.dev/docs/using-a-listview

[4] [9] Theme switching in React Native. React Native provides a powerful and... | by Faraz Irfan | Medium
https://medium.com/@farazirfan47/theme-switching-in-react-native-b70a110c41dc

[5] Using ESLint and Prettier - Expo Documentation
https://docs.expo.dev/guides/using-eslint/

[6] [7] How I set up Design System for my React Native Projects for 10x Faster Development - DEV Community
https://dev.to/msaadullah/how-i-set-up-design-system-for-my-react-native-projects-for-10x-faster-development-1k8g

[8] [10] [16] How to Integrate ChatGPT API in React Native - DEV Community
https://dev.to/mrcflorian/how-to-integrate-chatgpt-api-in-react-native-3k1j

[11] [13] Using TypeScript - Expo Documentation
https://docs.expo.dev/guides/typescript/

[12] JavaScript tabs - Expo Documentation
https://docs.expo.dev/router/advanced/tabs/

[14] [15] [21] Build APKs for Android Emulators and devices - Expo Documentation
https://docs.expo.dev/build-reference/apk/

[17] [18] Prompt engineering for GitHub Copilot Chat - GitHub Docs
https://docs.github.com/en/copilot/concepts/prompting/prompt-engineering

[19] [20] react-native-maps - Expo Documentation
https://docs.expo.dev/versions/latest/sdk/map-view/

[22] [23] 8 Things You Should Do Before Going Live with React Native | by Kadi Kraman | Medium
https://medium.com/@hellokadi/8-things-you-should-do-before-the-first-release-of-your-react-native-app-119e385ce0fa