This is a copy of a conversation between ChatGPT & Anonymous.
Report conversation

screen 1

name
email
age
contact(country code)
organisation(optional)

cta
////////////

screen 1 ke cta pe click krenge
hmri open ai ki api hit hogi
and user ka data collect kregi
and it is compulosry to fill all the fields
aaage btata hu kya kya karna hoga and kaise karna hoga
smje
////////////

abb for example
name - ankit , age - 22 , contact +1 9898989898
abb open ai jakr

phle search krega pura iske baare me
and dont worry hmre pas user ka consent hoga to get the profile data
and open ais e bhi baat hogyi to collect the data

and then with logic
like indian of age 21 mostly likes to drink , eat veg and all

wo query kaise pass karni hai and kya pass karni hai wo sbb hoga smje

so
food preference collect krenge , isme 5 filelds hogi
aap decide karo and btaoge , data model bhi waise hi

bnega

same drink preference
same hotel preference
same coffe preference
same brand preference

and basic prompting with open ai se collect krenge ye sbb
data model kya hoga
data model kaise kaam krega
open ai ke response ko kaise structrurte krenge

and then open ai ke response aane ke baad user ke profile ko kaise populate krenge
ye sbb we have to do at any cost smje

/////////////////

so do web search
we have complete consent for this and then we will do this

Pro thinking >

Answer now

Chalo isko pura system-level soch ke design karte hain. I'll break it into:

1. Screen-1 → backend → OpenAI flow
2. Data model (user + preferences)
3. Prompt + OpenAI response format (JSON)
4. "Ankit, 22…" example end-to-end
5. Profile populate ka flow + some safety/bias tips

---

1. Screen 1 → Backend → OpenAI: high-level flow

Screen 1 fields (all required except org):

- name
- email
- age

- contact (country code + phone)
- organisation (optional)

On CTA click:

1. **Frontend validation**
   - Check sab required fields bhare hue hain.
   - Age must be number, email valid, phone valid, etc.
2. **Send to backend**
   - POST `/api/onboarding` with JSON:

```json
{
  "name": "Ankit",
  "email": "ankit@example.com",
  "age": 22,
  "phone_country_code": "+91",
  "phone_number": "9898989898",
  "organisation": null
}
```

3. **Backend:**
   - Create a `User` record in DB.
   - Derive `country` from phone code (e.g. `+91 → India`, `+1 → US/Canada`).
   - Call OpenAI to **infer default preferences** (food, drink, hotel, coffee, brand) using structured JSON output. OpenAI Platform +1
   - Save returned preferences into DB under `UserProfile`.
4. **Next screens**
   - User ko profile screen dikhao jahan ye preferences pre-filled hain.
   - User manually edit/confirm kar sakta hai.

---

2. Data model design

Main assume kar raha hoon aap JSON / NoSQL style use karoge; same thing RDBMS (Postgres/MySQL) me tables banake bhi kiya ja sakta hai.

2.1 User core model

```ts
type User = {
  id: string;
  name: string;
  email: string;
  age: number;
  phoneCountryCode: string; // "+91"
  phoneNumber: string;      // "9898989898"
  organisation?: string | null;

  country?: string | null;  // derived from phone code
  createdAt: string;
  updatedAt: string;

  profile?: UserProfile;
};
```

2.2 UserProfile + preference sections

```ts
type UserProfile = {
  food?: FoodPreference;
  drink?: DrinkPreference;
  hotel?: HotelPreference;
  coffee?: CoffeePreference;
  brand?: BrandPreference;

  // meta
  lastEnrichedAt?: string;
};
```

Har preference ke liye **5 core business fields** rakhenge (jo aapne bola). Meta fields (confidence, source, etc.) optional add kar sakte ho.

2.3 FoodPreference (5 fields)

Goal: default food behaviour.

```ts
type FoodPreference = {
  dietType: "veg" | "non_veg" | "vegan" | "mixed" | "unknown";
  spiceLevel: "low" | "medium" | "high" | "unknown";
  favoriteCuisines: string[];      // ["North Indian", "Chinese"]
  budgetLevel: "low" | "mid" | "high" | "unknown"; // per meal
  eatingOutFrequency: "rarely" | "weekly" | "few_times_week" | "daily" | "unknown
};
```

2.4 DrinkPreference (5 fields)

Yahan pe alcohol + non-alcoholic dono generic way me:

```ts
type DrinkPreference = {
  alcoholPreference: "never" | "occasionally" | "frequently" | "unknown";
  favoriteAlcoholTypes: string[];      // ["beer", "whisky"] | []
  favoriteNonAlcoholicTypes: string[]; // ["coffee", "tea", "soft_drink"]
  sugarLevel: "low" | "medium" | "high" | "unknown";
  caffeineTolerance: "none" | "low" | "medium" | "high" | "unknown";
};
```

> Important note: model ko clearly bolna hoga “if not enough info → alcoholPreference: `unknown`”, warna stereotype estate me chala jayega.

2.5 HotelPreference (5 fields)

```ts
type HotelPreference = {
  budgetPerNight: {
    currency: string; // "INR", "USD"
    min: number | null;
    max: number | null;
  };
  hotelClass: "hostel" | "budget" | "3_star" | "4_star" | "5_star" | "unknown";
  locationPreference: "city_center" | "suburbs" | "near_airport" | "scenic" | "un
  roomType: "single" | "double" | "dorm" | "suite" | "unknown";
  amenitiesPriority: string[]; // ["wifi", "breakfast", "gym", "pool"]
};
```

2.6 CoffeePreference (5 fields)

ts Copy code

```ts
type CoffeePreference = {
  coffeeConsumerType: "none" | "occasional" | "daily" | "heavy" | "unknown";
  coffeeStyle: string[]; // ["latte", "cold brew", "filter coffee"]
  milkPreference: "dairy" | "oat" | "soy" | "almond" | "none" | "unknown";
  sweetnessLevel: "no_sugar" | "low" | "medium" | "high" | "unknown";
  cafeAmbiencePreference: "quiet_work" | "casual" | "social_loud" | "no_preferenc
};
```

2.7 BrandPreference (5 fields)

ts Copy code

```ts
type BrandPreference = {
  fashionStyle: "streetwear" | "minimal" | "formal" | "sporty" | "mixed" | "unkno
  techEcosystem: "apple" | "android" | "windows" | "mixed" | "unknown";
  shoppingChannels: string[]; // ["online_marketplace", "direct_brand", "offline_
  priceSensitivity: "very_price_sensitive" | "value_for_money" | "mid_range" | "p
  brandValues: string[]; // ["sustainability", "status", "comfort", "local_suppor
};
```

---

3. OpenAI se preferences kaise nikalne (prompt + JSON)

3.1 Use structured JSON outputs

OpenAI ke latest models (GPT-4o, etc.) **structured outputs / JSON schema** support karte hain, jisse model predictable JSON return karta hai. OpenAI Platform +1
Aap `response_format` me schema de sakte ho.

Example: Node.js pseudo-code using `fetch` + `gpt-4o-mini`: OpenAI Platform

js

Copy code

```js
const response = await fetch("https://api.openai.com/v1/chat/completions", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Authorization": `Bearer ${process.env.OPENAI_API_KEY}`
  },
  body: JSON.stringify({
    model: "gpt-4o-mini",
    response_format: {
      type: "json_schema",
      json_schema: {
        name: "UserPreferences",
        strict: true,
        schema: {
          type: "object",
          properties: {
            food: {
              type: "object",
              properties: {
                dietType: { type: "string" },
                spiceLevel: { type: "string" },
                favoriteCuisines: {
                  type: "array",
                  items: { type: "string" }
                },
                budgetLevel: { type: "string" },
                eatingOutFrequency: { type: "string" }
              },
              required: [
                "dietType",
                "spiceLevel",
                "favoriteCuisines",
                "budgetLevel",
                "eatingOutFrequency"
              ]
            },
            // same pattern: drink, hotel, coffee, brand...
          },
          required: ["food", "drink", "hotel", "coffee", "brand"],
          additionalProperties: false
        }
      }
    },
    messages: [
      {
        role: "system",
        content: `
You are an assistant that estimates default lifestyle preferences for onboarding.
Rules:
- Use ONLY the user profile information given.
- You may use general trends for AGE + COUNTRY, but avoid stereotypes.
- If you are not confident, set the field to "unknown" or an empty list.
- Do NOT infer sensitive attributes (religion, politics, health, sexuality).
        `.trim()
      },
      {
        role: "user",
        content: JSON.stringify({
          name: "Ankit",
          age: 22,
          country: "India",
          phoneCountryCode: "+91"
        })
      }
    ]
  })
});

const data = await response.json();
const prefs = JSON.parse(data.choices[0].message.content);
// prefs.food / prefs.drink / ...
```

> Real code me `json_schema` object ko properly build karna hoga, maine yahan sirf food part dikhaya. Baaki 4 categories bhi isi pattern se.

3.2 “Web search first” ka concept

Honestly, **har user ke liye alag-alag web crawling karna** naam + phone se karna:

- technically unreliable,
- privacy & policy point of view risky ho sakta hai (even with consent, aapko local laws follow karne hote hain). OpenAI +1

Aap jo example bol rahe ho “**Indian 21 age → mostly veg / likes to drink**”, wo actually **population-level trend** hai, *personal* information nahi. Ye cheez LLM ke training data se already learnt hoti hai; alag web search ki zaroorat nahi padti.

Agar aap fir bhi real-time trend chahte ho (e.g. “2025 me India me youth ka coffee preference kya hai?”), to ek alag step rakh sakte ho:

- Step A: ek baar per segment call karo `gpt-4o-search-preview` jaise search-specialized model ko, jo web browse karke latest data de. OpenAI Platform
- Step B: uska summary store karo `demographic_insights` table me.
- Step C: User ke age + country → us segment ki insights + user data → OpenAI ko feed karke final preferences nikaalo.

Lekin user-level “checking social media, etc.” ka detailed design main yahan nahi de raha, kyunki woh bahut jaldi privacy/bad-use side pe chala jaata hai. Better approach: **user se hi question puch ke confirm karna** (see below).

---

4. Example flow with “Ankit, 22, +1 9898...”

Maan lo:

- `name = "Ankit"`
- `age = 22`
- `phone_country_code = "+91"` (India)

Step 1: Backend pre-processing

```ts
const userInput = {
  name: "Ankit",
  email: "ankit@example.com",
  age: 22,
  phoneCountryCode: "+91",
  phoneNumber: "9898989898"
};

const derived = {
  country: "India",
  ageBracket: "18-24"
};
```

Step 2: OpenAI request

- System prompt: rules, safety, unknown handling.
- User content: `userInput + derived`.

Model internally will do something like:

- 22, India → maybe:
  - dietType: `"mixed"` (veg + non-veg) **OR** `"unknown"` (if not sure)
  - spiceLevel: `"medium"` / `"high"` (youth often like spicy but again not hard rule)
  - etc.

You store whatever JSON came back.

Step 3: Save to DB

```ts
await db.user.update({
  where: { id: userId },
  data: {
    profile: {
      food: prefs.food,
      drink: prefs.drink,
      hotel: prefs.hotel,
      coffee: prefs.coffee,
      brand: prefs.brand,
      lastEnrichedAt: new Date().toISOString()
    }
  }
});
```

Step 4: Show to user

Next screen pe:

- “We’ve pre-filled your preferences. Please check & update.”
- Har section (Food/Drink/Hotel/Coffee/Brand) editable.

Isse **AI sirf starting point** banega, final truth nahi. Ye important hai biases avoid karne ke liye.

---

5. Prompting strategy (basic but effective)

Aapko lagbhag teen type ke prompts lagenge:

5.1 System prompt (global rules)

- Explain:
  - app ka goal: onboarding personalization
  - allowed sources: user data + general trends
  - disallowed: guessing sensitive stuff, over-confident assumptions

Example (short Hindi/English mix allowed):

```text
You are an onboarding assistant.

Your job:
- Estimate default preferences for FOOD, DRINK, HOTEL, COFFEE, BRAND.
- Use only the user profile given and general behavioral trends for their age gro
- Avoid stereotypes; if you're not reasonably confident, output "unknown" or [].

Constraints:
- Do NOT infer sensitive attributes: religion, politics, health, sexual orientati
- Never say you "looked up" or "tracked" the user; you only see the provided data
- Strictly follow the JSON schema provided.
```

5.2 User message (context)

json Copy code

```json
{
  "name": "Ankit",
  "age": 22,
  "country": "India",
  "phoneCountryCode": "+91",
  "organisation": null
}
```

5.3 JSON schema (response_format)

- Already shown above for `food`; same structure for `drink`, `hotel`, `coffee`, `brand`.
- Strict mode + `additionalProperties: false` so output clean रहे.

---

6. User profile populate logic (step-by-step)

1. **CTA click on Screen 1**
    - Frontend → backend `/api/onboarding`.
2. **Backend: upsert user**

ts Copy code

```ts
const user = await db.user.create({
  data: {
    name,
    email,
    age,
    phoneCountryCode,
    phoneNumber,
    organisation,
    country: derivedCountry
  }
});
```

3. **Call OpenAI → `prefs` JSON**
4. **Validate + map to DB model**
   - Joi/Zod ya custom validator se check:
     - All required keys present: `food`, `drink`, `hotel`, `coffee`, `brand`.
     - Har object ke 5 fields string/number/array as expected.
   - If validation fail → fallback: set everything `unknown`, ask user directly later.
5. **Save profile**

ts

Copy code

```ts
await db.user.update({
  where: { id: user.id },
  data: {
    profile: {
      ...prefs,
      lastEnrichedAt: new Date().toISOString()
    }
  }
});
```

6. **Return to frontend**

json

Copy code

```json
{
  "userId": "123",
  "profile": {
    "food": { ... },
    "drink": { ... },
    "hotel": { ... },
    "coffee": { ... },
    "brand": { ... }
  }
}
```

7. **Frontend: populate screen 2+**
   - Show UI cards: “Food preferences”, “Drink preferences”, etc.

- User edits & saves → backend updates profile again, but is baar `source = "user_confirmed"` (agar aap meta add karna chaho).

---

7. Important safety / bias notes (real-world product ke liye)

- **Avoid harmful profiling**: OpenAI policies explicitly restrict using the API for harmful social scoring / risk scoring, or inferring sensitive attributes. OpenAI
- **Consent is good, but not everything**: Local data-protection laws (GDPR, PDP etc.) ka dhyaan rakho; consent ke saath bhi "over-collection" avoid karo. OpenAI +1
- **User-editable final say**: Har AI-inferred field ko user se confirm karwana is a good UX and safer ethically.

Agar tum chaho to next step me main:

- exact JSON schema (saare 5 sections fully),
- ya Node/Express endpoints ka skeleton,
- ya UI wireframe (Screen 1, Screen 2 with cards)

bhi likh sakta hoon. Just bol do kis part ko detail me banana hai ab.

Sources

i want one single prompt that i will give to openai and
then it answer all of this smje
like i dont need anything null

i want open ai to answer all of this fields when i m giving
this + query that you will give {{
  "name": "Ankit",
  "email": "ankit@example.com",
  "age": 22,
  "phone_country_code": "+91",
  "phone_number": "9898989898",
  "organisation": null
}
}

```
type UserProfile = {
  food?: FoodPreference;
  drink?: DrinkPreference;
  hotel?: HotelPreference;
  coffee?: CoffeePreference;
  brand?: BrandPreference;

  // meta
  lastEnrichedAt?: string;
};


type FoodPreference = {
  dietType: "veg" | "non_veg" | "vegan" | "mixed" | "unknown";
  spiceLevel: "low" | "medium" | "high" | "unknown";
  favoriteCuisines: string[];      // ["North Indian", "Chinese"]
  budgetLevel: "low" | "mid" | "high" | "unknown"; // per meal
  eatingOutFrequency: "rarely" | "weekly" | "few_times_week" | "daily" | "unknown";
};

type DrinkPreference = {
  alcoholPreference: "never" | "occasionally" | "frequently" | "unknown";
  favoriteAlcoholTypes: string[];      // ["beer", "whisky"] | []
  favoriteNonAlcoholicTypes: string[]; // ["coffee", "tea", "soft_drink"]
  sugarLevel: "low" | "medium" | "high" | "unknown";
  caffeineTolerance: "none" | "low" | "medium" | "high" | "unknown";
};


type HotelPreference = {
  budgetPerNight: {
    currency: string; // "INR", "USD"
    min: number | null;
    max: number | null;
  };
  hotelClass: "hostel" | "budget" | "3_star" | "4_star" |
"5_star" | "unknown";
  locationPreference: "city_center" | "suburbs" |
"near_airport" | "scenic" | "unknown";
  roomType: "single" | "double" | "dorm" | "suite" |
"unknown";
  amenitiesPriority: string[]; // ["wifi", "breakfast", "gym",
"pool"]
};

type CoffeePreference = {
  coffeeConsumerType: "none" | "occasional" | "daily" |
"heavy" | "unknown";
  coffeeStyle: string[]; // ["latte", "cold brew", "filter coffee"]
  milkPreference: "dairy" | "oat" | "soy" | "almond" | "none" |
"unknown";
  sweetnessLevel: "no_sugar" | "low" | "medium" | "high" |
"unknown";
  cafeAmbiencePreference: "quiet_work" | "casual" |
"social_loud" | "no_preference";
};

type BrandPreference = {
  fashionStyle: "streetwear" | "minimal" | "formal" | "sporty"
| "mixed" | "unknown";
  techEcosystem: "apple" | "android" | "windows" | "mixed" |
"unknown";
  shoppingChannels: string[]; // ["online_marketplace",
"direct_brand", "offline_mall"]
  priceSensitivity: "very_price_sensitive" | "value_for_money"
| "mid_range" | "premium" | "unknown";
  brandValues: string[]; // ["sustainability", "status",
"comfort", "local_support"]
};
```

and once that is done then how will you get the reponse from open ai and then populate in profile section of hushh user smje and supabase db

do websearch and a lot of stuff that you can do for us also if there is any other free api which can help let me know about that too

Thought ›

Thought for 1m 1s ›

Got it. Here's a **single-call, copy-paste** setup that (a) forces OpenAI to return **every field** with **no nulls**, (b) gives you one compact "prompt" you can reuse, and (c) shows exactly how to **parse it** and **upsert** into **Supabase**.

---

1) One-call OpenAI template (Chat Completions + JSON Schema)

> Guarantees a valid JSON that matches your `UserProfile` shape using Structured Outputs (`response_format: json_schema`). If you prefer the newer **Responses API**, see the note right after this snippet. OpenAI Platform +2

```ts
// npm i openai supabase @supabase/supabase-js google-libphonenumber
import OpenAI from "openai";
import { createClient } from "@supabase/supabase-js";
// Optional: use libphonenumber to derive ISO country from phone and set currency
import { PhoneNumberUtil } from "google-libphonenumber";

const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });
const supabase = createClient(process.env.SUPABASE_URL!, process.env.SUPABASE_KEY

// ---- Helper: derive ISO country + currency hint from phone (fallbacks safe) --
async function deriveGeoHints(phone_country_code: string, phone_number: string) {
  try {
    const phoneUtil = PhoneNumberUtil.getInstance();
    const parsed = phoneUtil.parseAndKeepRawInput(`${phone_country_code}${phone_n
    const iso2 = phoneUtil.getRegionCodeForNumber(parsed) || "US";       // e.g.
    // Use REST Countries to get currency, free & no auth
    const res = await fetch(`https://restcountries.com/v3.1/alpha/${iso2}`);
    const json = await res.json();
    const currencyCode =
      (json && json[0] && json[0].currencies && Object.keys(json[0].currencies)[0
    return { iso2, currencyCode };
  } catch {
    return { iso2: "US", currencyCode: "USD" };
  }
}

// ---- SINGLE CALL to OpenAI that fills every field (no nulls) ----
export async function enrichUserProfile(inputUser: {
  name: string;
  email: string;
  age: number;
  phone_country_code: string;
  phone_number: string;
  organisation: string | null;
}) {
  const geo = await deriveGeoHints(inputUser.phone_country_code, inputUser.phone_

  const schema = {
    name: "UserProfile",
    schema: {
      type: "object",
      additionalProperties: false,
      required: ["food", "drink", "hotel", "coffee", "brand", "lastEnrichedAt"],
      properties: {
        food: {
          type: "object",
          additionalProperties: false,
          required: ["dietType", "spiceLevel", "favoriteCuisines", "budgetLevel",
          properties: {
            dietType: { type: "string", enum: ["veg", "non_veg", "vegan", "mixed"
            spiceLevel: { type: "string", enum: ["low", "medium", "high", "unknow
            favoriteCuisines: { type: "array", items: { type: "string" }, minItem
            budgetLevel: { type: "string", enum: ["low", "mid", "high", "unknown"
            eatingOutFrequency: {
              type: "string",
              enum: ["rarely", "weekly", "few_times_week", "daily", "unknown"]
            }
          }
        },
        drink: {
          type: "object",
          additionalProperties: false,
          required: [
            "alcoholPreference",
            "favoriteAlcoholTypes",
            "favoriteNonAlcoholicTypes",
            "sugarLevel",
            "caffeineTolerance"
          ],
          properties: {
            alcoholPreference: { type: "string", enum: ["never", "occasionally",
            favoriteAlcoholTypes: { type: "array", items: { type: "string" }, min
            favoriteNonAlcoholicTypes: { type: "array", items: { type: "string" }
            sugarLevel: { type: "string", enum: ["low", "medium", "high", "unknow
            caffeineTolerance: { type: "string", enum: ["none", "low", "medium",
          }
        },
        hotel: {
          type: "object",
          additionalProperties: false,
          required: ["budgetPerNight", "hotelClass", "locationPreference", "roomT
          properties: {
            budgetPerNight: {
              type: "object",
              additionalProperties: false,
              required: ["currency", "min", "max"],
              properties: {
                currency: { type: "string", pattern: "^[A-Z]{3}$" }, // e.g. INR,
                min: { type: "number", minimum: 0 },
                max: { type: "number", minimum: 0 }
              }
            },
            hotelClass: { type: "string", enum: ["hostel", "budget", "3_star", "4
            locationPreference: {
              type: "string",
              enum: ["city_center", "suburbs", "near_airport", "scenic", "unknown
```

```
    },
    roomType: { type: "string", enum: ["single", "double", "dorm", "suite
    amenitiesPriority: { type: "array", items: { type: "string" }, minIte
  }
},
coffee: {
  type: "object",
  additionalProperties: false,
  required: [
    "coffeeConsumerType",
    "coffeeStyle",
    "milkPreference",
    "sweetnessLevel",
    "cafeAmbiencePreference"
  ],
  properties: {
    coffeeConsumerType: { type: "string", enum: ["none", "occasional", "d
    coffeeStyle: { type: "array", items: { type: "string" }, minItems: 1
    milkPreference: { type: "string", enum: ["dairy", "oat", "soy", "almo
    sweetnessLevel: { type: "string", enum: ["no_sugar", "low", "medium",
    cafeAmbiencePreference: {
      type: "string",
      enum: ["quiet_work", "casual", "social_loud", "no_preference"]
    }
  }
},
brand: {
  type: "object",
  additionalProperties: false,
  required: ["fashionStyle", "techEcosystem", "shoppingChannels", "priceS
  properties: {
    fashionStyle: {
      type: "string",
      enum: ["streetwear", "minimal", "formal", "sporty", "mixed", "unkno
    },
    techEcosystem: { type: "string", enum: ["apple", "android", "windows"
    shoppingChannels: { type: "array", items: { type: "string" }, minItem
    priceSensitivity: {
      type: "string",
      enum: ["very_price_sensitive", "value_for_money", "mid_range", "pre
      },
      brandValues: { type: "array", items: { type: "string" }, minItems: 1
     }
    },
    lastEnrichedAt: { type: "string", format: "date-time" }
   }
  },
  strict: true
 } as const;

 const system = `
You are an onboarding preference estimator. Produce realistic defaults from age +
HARD RULES:
- Output must strictly match the provided JSON schema. Do not include any extra k
- Never output null. If uncertain for enums, use "unknown".
- Arrays MUST have at least 1 item. If uncertain, include ["unknown"].
- budgetPerNight must include a plausible numeric [min, max] in the user's curren
- Use these hints: country=${geo.iso2}, preferred_currency=${geo.currencyCode}, a
- Keep tastes reasonable for 18-24 without stereotyping; avoid sensitive inferenc
- Set lastEnrichedAt to current time in ISO 8601 (UTC).
`.trim();

 const user = {
  user_profile_seed: {
   name: inputUser.name,
   email: inputUser.email,
   age: inputUser.age,
   phone_country_code: inputUser.phone_country_code,
   phone_number: inputUser.phone_number,
   organisation: inputUser.organisation,
   detected_country_iso2: geo.iso2,
   preferred_currency: geo.currencyCode
  }
 };

 const completion = await openai.chat.completions.create({
  model: "gpt-4o-mini",
  response_format: { type: "json_schema", json_schema: schema },
  messages: [
   { role: "system", content: system },
      {
        role: "user",
        content: JSON.stringify(user)
      }
    ],
    temperature: 0.4
  });

  const content = completion.choices[0].message?.content || "{}";
  const parsed = JSON.parse(content); // Guaranteed to match schema (no nulls)

  return parsed; // -> { food, drink, hotel, coffee, brand, lastEnrichedAt }
}
```

Using the new Responses API? Replace `chat.completions.create` with `client.responses.create` and set `text.format = { type: "json_schema", schema }` per the migration guide. Functionally equivalent; the above is the most widely compatible today. OpenAI Platform

---

2) Upsert into Supabase (store on the user record)

> Upsert merges new profile JSON into your row. Use a `jsonb` column like `profile`.
> Supabase +1

ts

Copy code

```ts
type Profile = Awaited<ReturnType<typeof enrichUserProfile>>;

export async function saveProfileToSupabase(userId: string, profile: Profile) {
  const { data, error } = await supabase
    .from("members")
    .upsert(
      {
        id: userId,                     // must be part of PK/unique for upsert
        profile: profile,               // JSONB column
        profile_last_updated_at: new Date().toISOString()
      },
      { onConflict: "id" }               // adjust to your PK/unique column(s)
    )
    .select();

  if (error) throw error;
  return data;
}
```

If you keep **core fields** (name, email, age, phones) on `members` and preferences under `members.profile` (JSONB), the above is perfect. If you split into two tables (`members` + `user_profiles`), run two upserts accordingly. (For multi-column conflicts, pass a comma-separated list to `onConflict`, e.g., `"user_id, version"`.) Supabase +1

---

3) End-to-end usage (your example)

ts

Copy code

```ts
const input = {
  name: "Ankit",
  email: "ankit@example.com",
  age: 22,
  phone_country_code: "+91",
  phone_number: "9898989898",
  organisation: null
};

const profile = await enrichUserProfile(input);   // one OpenAI call → full JSON,
await saveProfileToSupabase("user_123", profile); // persist
```

---

4) Why this guarantees “no nulls”

- We use **Structured Outputs** with a **strict JSON Schema**. No `null` types are permitted, arrays require `minItems: 1`, all objects have `required` lists, so the model **must** fill everything. For uncertainty, it returns `"unknown"` or `["unknown"]`. OpenAI Platform +1

- If you switch to **Responses API**, the same structure is set via `text.format`.
  OpenAI Platform

---

5) Free helpers you may want

- **Phone → Country (ISO) & validation**: `google-libphonenumber` / `libphonenumber-js` (no external API, runs locally). GitHub +1
- **Country → Currency**: **REST Countries API** (free, no key): `/v3.1/alpha/{ISO2}` → `currencies`. restcountries.com
- **Currency conversion** (if later you want budgets normalized): **exchangerate.host** (free, no API key). exchangerate.host +1

Notes you can keep in your playbook

- Keep `temperature` low (0.2–0.5) for stable, deterministic defaults.
- If you ever want stricter control on numeric hotel budgets, pass a **policy object** in the user message (e.g., default min/max per currency) and say "if uncertain, use policy