Absolutely. Here’s a clear, end-to-end **Replit task write-up** you can drop into your project workspace README (or a Replit “Task” doc). It treats this as an **augmentation** to your existing lead-finder—keeping your current UI intact while adding a GPT-5 planning/synthesis layer around **Google Places** + deterministic website checks.

# Task: Augment Lead Finder with GPT-5 Reasoning + Google Places Verification (Replit)

## 0) Why this exists (one paragraph)

We’re upgrading the current “surface-level” lead search into a **high-fidelity, evidence-first** pipeline. GPT-5 will handle **planning, triage, and synthesis**, while code does **deterministic verification** (Google Places, DOM checks, contact verification). We’ll use the **Responses API** for reasoning models and control cost/latency via model sizing and reasoning-effort knobs. ([OpenAI](https://openai.com/index/new-tools-and-features-in-the-responses-api/?utm_source=chatgpt.com))

## 1) Scope (augmentation, not replacement)

* Keep the existing endpoints/UX.
* Add an **opt-in** pathway: enrichment\_flags.gpt5=true.
* New worker pipeline: **Plan → Seed (Places) → Verify (website/booking) → Enrich (contacts) → Cross-validate → Score → Synthesize (GPT-5)**.
* Output remains backward-compatible; we add optional fields (evidence, scores, reasons).

## 2) Environment (Replit)

**Language:** Node.js (recommended)  
 **Secrets:** Add these in Replit → Tools → Secrets

* OPENAI\_API\_KEY
* GOOGLE\_MAPS\_API\_KEY

**Packages (npm):**

****openai axios express p-limit cheerio zod

(Optional for headless rendering: playwright — if you prefer to start renderless, skip headless and rely on static HTML + vendor-link detection; you can add Playwright later.)

**Run script (package.json):**

****{

"type": "module",

"scripts": { "start": "node src/server.js" }

}

**Suggested file layout**

****/src

server.js // Express server

controller.search.js // Orchestrates the pipeline

planner.gpt5.js // Calls GPT-5 for plan & synthesis

places.client.js // Google Places Text Search + Place Details

audit.site.js // Website path checks + booking detection

enrich.contacts.js // Optional: contact discovery + verification

scoring.js // Lead scoring

schemas.js // Zod schemas for inputs/outputs



## 3) What happens when the user clicks “Search”

**Example:** “HVAC businesses in Charleston, South Carolina without an appointment booking capability on their website.”

### 3.1 Intent → Plan (GPT-5; Responses API)

* GPT-5 parses the query into {vertical, geo, constraints, max\_results} and emits a **plan JSON**:  
  + Places queries (primary + alternates),
  + website paths to check: /, /book, /schedule, /appointments, /contact,
  + booking vendors/keywords to scan (e.g., calendly, acuityscheduling, squareup.com/appointments, housecallpro, servicetitan/scheduleengine, setmore, thryv, workiz),
  + enrichment order (contacts),
  + cross-validation rules.
* Use **Responses API** with a reasoning model; set **reasoning.effort: "low"** here for speed. ([OpenAI](https://openai.com/index/new-tools-and-features-in-the-responses-api/?utm_source=chatgpt.com))

### 3.2 Seed candidates (Google Places)

* Call **Text Search (New)** with the plan’s text query (e.g., “HVAC in Charleston SC”), biasing to the city; **must supply a FieldMask** header to choose returned fields. Typical mask for seeding:  
   places.id,places.name,places.displayName,places.formattedAddress,places.types,places.websiteUri,places.rating,places.userRatingCount,places.googleMapsUri  
   (New v1 uses places:searchText endpoint and FieldMask; there is **no default field list**—omitting it returns an error.) ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/text-search?utm_source=chatgpt.com))
* For each returned place, call **Place Details (New)** only if you need deeper fields later; otherwise, Text Search with a wider field mask is often enough. **Place objects include websiteUri, rating, userRatingCount, etc.** ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/place-details?utm_source=chatgpt.com))
* Dedupe by normalized name + address + phone.

### 3.3 Website discovery (deterministic)

* If websiteUri from Places exists, use it; else do a lightweight SERP (optional) to find a likely domain and store it as website\_url\_candidate with evidence.
* Normalize and store canonical URL.

### 3.4 Booking detection (deterministic)

* Fetch homepage and the planned subpaths: /, /book, /schedule, /appointments, /contact.
* Parse static HTML first (cheap); optionally render headless later.
* Search for:  
  + Buttons/links with *book/schedule/appointment* tokens,
  + known booking vendor iframes/URLs (examples above).
* Record **exact URL + selector + text snippet** for every hit (or negative check) in evidence\_log.

### 3.5 Constraint evaluation (rule engine)

* no\_booking = true only if:  
  + No booking tokens across tested paths **and**
  + No vendor URLs detected **and**
  + (Optional) SERP shows no booking page.
* has\_website = false only if:  
  + Places websiteUri is null **and**
  + SERP fails to find a resolvable domain (or the candidate domain is parked/404).
* All **absence claims require ≥2 independent negatives** (e.g., DOM + SERP).

### 3.6 Contact discovery (optional; deterministic first)

* Sources: website /contact, Google profile, Facebook page, state/county registry.
* If enabled, call enrichers (e.g., verification APIs) and attach verification\_score / carrier lookup metadata.

### 3.7 Score & rank (transparent)

* Weighted signals (configurable):  
   no\_booking (+30), higher userRatingCount (+ up to 25), rating 3.8–4.9 (+5–10), owner found (+5), verified email (+10), verified phone (+5), “has website **but** no booking” (+15), chains (−).
* Return lead\_score 0–100 and the raw reasons.

### 3.8 Synthesize (GPT-5; Responses API)

* Send **structured results** (leads + evidence\_log) to GPT-5 for:  
  + confidence\_reasons (3–5 bullets citing evidence entries),
  + one-line recommendation (what to pitch),
  + ranked summary of the top leads.
* Use **reasoning.effort: "medium"** here; keep the model as **synthesizer, not source of truth**. ([OpenAI](https://openai.com/index/new-tools-and-features-in-the-responses-api/?utm_source=chatgpt.com))

## 4) API call examples (ready to paste)

### 4.1 Google Places — Text Search (New)

****curl -X POST 'https://places.googleapis.com/v1/places:searchText' \

-H "Content-Type: application/json" \

-H "X-Goog-Api-Key: $GOOGLE\_MAPS\_API\_KEY" \

-H "X-Goog-FieldMask: places.id,places.displayName,places.formattedAddress,places.types,places.websiteUri,places.rating,places.userRatingCount,places.googleMapsUri" \

-d '{

"textQuery": "HVAC in Charleston, South Carolina",

"maxResultCount": 20

}'

*   
  **Notes:** Text Search (New) uses POST + FieldMask; add fields like places.websiteUri to get website URLs directly. ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/text-search?utm_source=chatgpt.com))

### 4.2 OpenAI — Responses API (Node sketch)

****import OpenAI from "openai";

const client = new OpenAI({ apiKey: process.env.OPENAI\_API\_KEY });

// 1) Plan quickly

export async function planSearch(userQuery) {

const res = await client.responses.create({

model: "gpt-5-mini",

reasoning: { effort: "low" },

input: [{

role: "user",

content: `Plan a verification workflow for: "${userQuery}".

Output strict JSON with:

- places\_queries: [primary, alternates...]

- website\_paths\_to\_check: ["/","/book","/schedule","/appointments","/contact"]

- booking\_vendor\_patterns: ["calendly","acuityscheduling","squareup.com/appointments","housecallpro","servicetitan","scheduleengine","setmore","thryv","workiz"]

- cross\_validation\_rules: short bullets

- enrichment\_order: sources to try for contacts`

}]

});

return JSON.parse(res.output\_text);

}

// 2) Synthesize after deterministic checks

export async function synthesize(leads, evidenceLog) {

const res = await client.responses.create({

model: "gpt-5",

reasoning: { effort: "medium" },

input: [{

role: "user",

content: JSON.stringify({ leads, evidence\_log: evidenceLog })

}]

});

return res.output\_text; // parse if you’ve asked for JSON

}

*   
  **Why Responses API:** it’s the recommended surface for reasoning models; supports reasoning controls and summaries for debugging. ([OpenAI](https://openai.com/index/new-tools-and-features-in-the-responses-api/?utm_source=chatgpt.com))

## 5) Data model (additive fields)

Add these to each lead (keep your existing fields):

{

"has\_website": true,

"detected\_features": {

"online\_booking": { "found": false, "evidence": [ { "url": "...", "selector": "#btn-book", "snippet": "..." } ] }

},

"emails": [ { "email": "...", "verified": true, "verification\_score": 0.92, "source": "..." } ],

"phones": [ { "number": "...", "verified": true, "source": "..." } ],

"lead\_score": 87,

"confidence\_reasons": ["..."],

"evidence\_log": [ { "ts": "...", "check": "booking\_scan", "url": "...", "selector": "...", "result": "not\_found" } ]

}



## 6) Deterministic rules you must enforce (reliability)

1. **Absence needs two negatives** (e.g., DOM scan + SERP).
2. A “no website” claim requires: websiteUri absent in Places **and** no resolvable domain by SERP (or parked/404).
3. A “no booking” claim requires: homepage + one booking path checked (both negative) **and** no vendor URL found.
4. Log every check to evidence\_log (URL, selector/snippet, timestamp).

## 7) Google Places notes that matter

* **FieldMask is required** for Text Search (New) / Place Details (New) — specify the list of fields or you’ll get an error. ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/choose-fields?utm_source=chatgpt.com))
* **Text Search (New)** returns places[] Place objects; you can include places.websiteUri, places.displayName, etc., in your FieldMask. ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/text-search?utm_source=chatgpt.com))
* **Place Details (New)** can enrich a known place by place\_id (resource name places/PLACE\_ID). ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/place-details?utm_source=chatgpt.com))
* The **Place** resource includes rating, websiteUri, userRatingCount, googleMapsUri, and more—choose what you need via FieldMask. ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/reference/rest/v1/places))
* If you rely on **Place Types**, consult Google’s types list; otherwise, prefer text queries (e.g., “HVAC in {city}”). ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/place-types?utm_source=chatgpt.com))

## 8) Acceptance criteria (what “done” means)

* **Accuracy:** ≤10% false positives for “no website”; ≤15% false positives for “no booking” on manual audits.
* **Evidence:** Every major claim has ≥1 evidence item; absence claims have ≥2 negatives.
* **Explainability:** Each lead includes confidence\_reasons and an evidence\_log.
* **Performance:** Stream seed results quickly; full enrichment finishes within your current budget; Places calls respect quotas.
* **Compatibility:** Existing API response shape preserved; new fields are optional.

## 9) Telemetry & guardrails

* Log: plan JSON, checks run, latency by step, rate-limit errors, and % of leads with sufficient evidence.
* Respect robots.txt; throttle fetches.
* Handle PII carefully; store only necessary contact data.
* Keep an audit trail (timestamps, URLs, optional screenshots) for 30 days.

## 10) Minimal controller sketch (Express)

****// src/server.js

import express from "express";

import { planSearch, synthesize } from "./planner.gpt5.js";

import { searchText, toPlacesFieldMask } from "./places.client.js";

import { auditWebsitePaths } from "./audit.site.js";

import { scoreLead } from "./scoring.js";

const app = express();

app.use(express.json());

app.post("/api/search", async (req, res) => {

const { query, enrichment\_flags = {} } = req.body;

const plan = await planSearch(query);

const places = await searchText({

textQuery: plan.places\_queries[0],

fieldMask: toPlacesFieldMask([

"places.id","places.displayName","places.formattedAddress",

"places.types","places.websiteUri","places.rating",

"places.userRatingCount","places.googleMapsUri"

])

});

const leads = [];

const evidenceLog = [];

for (const p of places) {

const website = p.websiteUri || null;

const audit = await auditWebsitePaths(website, plan.website\_paths\_to\_check || []);

evidenceLog.push(...audit.evidence);

const hasBooking = audit.found;

const lead = {

business\_name: p.displayName?.text ?? p.displayName ?? "",

address: p.formattedAddress ?? "",

website\_url: website,

has\_website: Boolean(website),

detected\_features: { online\_booking: { found: hasBooking, evidence: audit.evidence.slice(0, 3) } },

rating: p.rating,

user\_rating\_count: p.userRatingCount

};

lead.lead\_score = scoreLead(lead);

leads.push(lead);

}

const synthesis = enrichment\_flags.gpt5 ? await synthesize(leads, evidenceLog) : null;

res.json({ plan, leads, synthesis, evidence\_log: evidenceLog });

});

app.listen(3000, () => console.log("Server running on :3000"));



## 11) Prompts to hard-code (so it works every time)

**Planning (system):**

You plan verification; the code collects evidence. Emit a compact JSON plan with: Places queries (1 primary + up to 4 alternates), website paths to check, booking vendor patterns, enrichment order, and cross-validation rules. Keep it machine-readable and ≤25 steps.

**Planning (user):**

Plan a verification workflow for: “{{user\_query}}”. Output JSON with keys: places\_queries, website\_paths\_to\_check, booking\_vendor\_patterns, enrichment\_order, cross\_validation\_rules.

**Synthesis (system):**

Given deterministic fields and an evidence\_log, produce confidence\_reasons (bullets citing evidence items by URL), a one-line recommendation, and (if requested) ranked lead labels. Do not override boolean flags—explain uncertainty if evidence is weak.

## 12) Testing (smoke tests you can run now)

* “HVAC in Charleston SC without online booking” → top results with has\_website set correctly; any no\_booking must have ≥2 negatives in evidence\_log.
* “Dentists near Mount Pleasant SC without websites” → ensure “no website” claims show Places websiteUri null + SERP negative.
* “Roofers in 29412 with >100 reviews and no online scheduling” → verify ranking by userRatingCount.
* “Plumbers in Savannah GA with website but no ‘book now’” → catch vendor URLs if present.

## 13) Cost & latency knobs (what to tune)

* Use **gpt-5-mini** for planning and **gpt-5** for final synthesis (where nuance matters).
* Adjust **reasoning.effort** per step (low for plan, medium for synthesis).
* Cap output with max\_output\_tokens if needed; monitor usage/latency via Responses API usage fields. ([OpenAI](https://openai.com/index/o1-and-new-tools-for-developers/?utm_source=chatgpt.com))

## 14) References (for implementers)

* **Responses API & Reasoning models** (reasoning effort, summaries, and tooling). ([OpenAI](https://openai.com/index/new-tools-and-features-in-the-responses-api/?utm_source=chatgpt.com))
* **Places API — Text Search (New) & FieldMask** (how to request websiteUri, etc.). ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/text-search?utm_source=chatgpt.com))
* **Places API — Place Details (New)** and **Place resource** (fields like rating, userRatingCount, websiteUri). ([Google for Developers](https://developers.google.com/maps/documentation/places/web-service/place-details?utm_source=chatgpt.com))

### Bottom line for Replit

Ship this as a **feature-flagged augmentation**. GPT-5 **plans + explains**, Google Places + your code **verify + evidence**. The result is a **ranked, auditable hot-lead list** instead of a shallow directory dump.