LLM Deliverables Integration — Claude / OpenAI

This document adds a complete, copy-pasteable implementation for calling an LLM (OpenAI or Anthropic/Claude) from your Firebase Cloud Function to generate the `deliverables` JSON for each audit. It includes:

- environment variable list
- `functions/src/llmClient.ts` (provider-agnostic wrapper)
- prompt template (system + user) tuned to StepLock + Microsite Empire
- example response schema (strict JSON format the app will save)
- `index.ts` snippet showing where to call the LLM and how to save the deliverables
- testing notes and safety reminders

---

1) Environment variables (set in Firebase Functions)

- `OPENAI_API_KEY` (if using OpenAI)
- `LLM_PROVIDER` = `openai` or `anthropic` (default `openai`)
- `ANTHROPIC_API_KEY` (if using Anthropic/Claude)
- `LLM_MODEL` (optional; defaults below)

Set in Firebase with:

```
firebase functions:config:set openai.key="sk_..." llm.provider="openai"
anthropic.key="claude_key_here"
```

And locally for testing: use `.env` or function emulator environment.

---

2) `functions/src/llmClient.ts` (provider wrapper)

Copy this file into `functions/src/llmClient.ts`. It uses the OpenAI Chat Completions endpoint by default. If you prefer Anthropic/Claude, switch provider to `anthropic` and implement the commented Anthropic call with your SDK.

```
// functions/src/llmClient.ts
import fetch from 'node-fetch';

type LLMProvider = 'openai' | 'anthropic';

const PROVIDER: LLMProvider = (process.env.LLM_PROVIDER as LLMProvider) ||
'openai';
```

```
const OPENAI_KEY = process.env.OPENAI_API_KEY || '';
const ANTHROPIC_KEY = process.env.ANTHROPIC_API_KEY || '';
const MODEL = process.env.LLM_MODEL || (PROVIDER === 'openai' ? 'gpt-4o-mini' :
'claude-v1');

export async function generateDeliverables(payload: {
  url: string;
  title?: string;
  bodyText: string;
  stepLock: any;
  links: string[];
}){
  const systemPrompt = `You are an expert local SEO strategist and copywriter.
Given a website's scraped content, StepLock keyword buckets (Emergency/Service/
Problem/Local), and links, produce a strict JSON object called \"deliverables\"
containing keywords by intent, a prioritized page_map (service × location) with
titles, meta descriptions, content outlines, suggested JSON-LD schema snippets,
technical audit notes, internal linking suggestions, and immediate next actions
for launch. Output must be valid JSON and follow the schema exactly.`;

  const userPrompt = `URL: ${payload.url}\n\nTitle: ${payload.title || ''}
\n\nSTEPLOCK: ${JSON.stringify(payload.stepLock)}\n\nLINKS (sample up to 50): $
{JSON.stringify(payload.links.slice(0,50))}\n\nBODY:\n$
{payload.bodyText.slice(0, 12000)}\n\nRespond only with the JSON object. Do not
include any explanatory text.`;

  if (PROVIDER === 'openai'){
    // OpenAI Chat Completions
    const resp = await fetch('https://api.openai.com/v1/chat/completions', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json', 'Authorization': `Bearer $
{OPENAI_KEY}` },
      body: JSON.stringify({
        model: MODEL,
        messages: [
          { role: 'system', content: systemPrompt },
          { role: 'user', content: userPrompt }
        ],
        temperature: 0.0,
        max_tokens: 2000
      })
    });
    const j = await resp.json();
    // Defensive parsing — many LLMs wrap JSON in backticks or text. Attempt to
extract JSON substring.
    const raw = j?.choices?.[0]?.message?.content || j?.choices?.[0]?.text ||
'';
    const json = extractJson(raw);
```

```
    return json;
  } else {
    // Anthropic / Claude (placeholder) — replace with official SDK call if
available
    // Example pseudocode:
    // const resp = await fetch('https://api.anthropic.com/v1/complete',
{ ... })
    // const raw = (await resp.json()).completion;
    // const json = extractJson(raw);
    // return json;
    throw new Error('Anthropic provider not implemented in this wrapper. Set
LLM_PROVIDER=openai or implement Anthropic call.');
  }
}

function extractJson(text: string){
  // find first { and last } and parse
  const start = text.indexOf('{');
  const end = text.lastIndexOf('}');
  if (start === -1 || end === -1) throw new Error('No JSON found in LLM
output');
  const substring = text.slice(start, end + 1);
  try{
    return JSON.parse(substring);
  }catch(err){
    // last resort: try to fix common issues
    const cleaned = substring.replace(/\n/g,' ').replace(/\`/g,'');
    return JSON.parse(cleaned);
  }
}
```

Notes: keep `temperature: 0.0` to favor deterministic JSON outputs. `max_tokens` may need to be increased for long outputs.

---

3) Prompt Template (system + user)

System prompt (exact)

> You are an expert local SEO strategist and copywriter. Given a website's scraped content, StepLock keyword buckets (Emergency/Service/Problem/Local), and links, produce a strict JSON object called "deliverables" containing:
>
> 1. `keywords`: array of objects `{ term, intent, score }`

2. `page_map`: prioritized array of pages `{ slug, title, meta_description, intent, recommended_wordcount, sections[] }` where each `sections` entry is `{ heading, bullets[] }`
3. `schema`: an array of JSON-LD snippets (as strings) keyed by page slug
4. `technical_audit`: array of issues `{ id, severity, description, fix_snippet }`
5. `internal_links`: array of `{ from_slug, to_slug, anchor_text }`
6. `next_actions`: array of short steps for launch and outreach
7. `estimated_lead_potential`: numeric estimate and rationale

Output must be valid JSON only. No extra commentary.

User prompt (example)

URL: `https://example.com`\n Title: `Example Title`\n STEPLOCK: `{"emergency":[],"service":["repair"],"problem":[],"local":["charlotte"]}`\n LINKS: `[...]`\n BODY: (first 12,000 chars of page body)

Respond only with the JSON object following the exact keys above.

---

4) Example `deliverables` JSON schema

```
{
  "keywords": [
    { "term": "mobile diesel repair charlotte", "intent": "emergency", "score": 0.98 },
    { "term": "diesel mechanic near me", "intent": "local", "score": 0.92 }
  ],
  "page_map": [
    {
      "slug": "mobile-diesel-repair-charlotte",
      "title": "Mobile Diesel Repair in Charlotte — We Come to You",
      "meta_description": "Fast mobile diesel repair in Charlotte. 24/7 service to get you back on the road.",
      "intent": "emergency",
      "recommended_wordcount": 700,
      "sections": [
        { "heading": "Why choose our mobile repair", "bullets": ["24/7 response","Experienced mechanics"] }
      ]
    }
  ],
  "schema": [
    { "slug": "mobile-diesel-repair-charlotte", "jsonld": "{\"@context\":\"https://schema.org\",...}" }
```

```
  ],
  "technical_audit": [
    { "id": "speed-images", "severity": "medium", "description": "Large images
not converted to webp", "fix_snippet": "Convert images to webp and add srcset" }
  ],
  "internal_links": [
    { "from_slug": "mobile-diesel-repair-charlotte", "to_slug": "about",
"anchor_text": "About our team" }
  ],
  "next_actions": ["Create location page for Charlotte (mobile-diesel-repair-
charlotte)", "Claim Google Business Profile"],
  "estimated_lead_potential": { "monthly_calls": 12, "rationale": "High
emergency intent, low competition" }
}
```

---

5) `index.ts` integration snippet (where to call the LLM)

Replace the TODO block in `functions/src/index.ts` with the snippet below. It calls `generateDeliverables`, validates, and updates Firestore.

```
// inside functions/src/index.ts after saving `ref`
import { generateDeliverables } from './llmClient.js';

// after upsertCompetitorToAirtable(...)
try{
  const deliverables = await generateDeliverables({ url, title, bodyText,
stepLock, links });
  if (deliverables && typeof deliverables === 'object'){
    await db.collection('audits').doc(ref.id).update({ deliverables });
  } else {
    console.warn('LLM returned no deliverables or invalid format');
  }
}catch(err){
  console.error('LLM error', err);
}
```

Notes:

- Wrap LLM call with try/catch — LLMs can fail or return non-JSON.
- Use `temperature:0` for reproducibility; increase only if you want variety.
- Consider streaming for very large outputs (advanced).

---

6) Testing & validation

1. Deploy to Firebase functions with test env vars (use a small sample URL)
2. Call `auditUrl` and check Firestore `audits/{id}` for `deliverables` object
3. If parsing fails, log `raw` LLM output to a `logs/` collection for debugging (be careful with PII)
4. Add a simple schema validator (AJV) in `llmClient` to ensure required keys exist before saving

`npm i ajv` then run a quick schema check before `db.update`.

---

7) Security & cost notes

- LLM calls cost money — add rate-limiting or require Pro plan to enable
- Don't log API keys or raw user content in public logs
- Consider caching deliverables per URL to avoid re-calls
- For sensitive sites (banking, legal), warn the user that content may include sensitive data and comply with Terms of Service

---

8) Next steps (automation & UX)

- Add a UI button "Generate Full Deliverables" and show progress spinner while LLM runs
- Add worker queue for long jobs (Cloud Tasks / PubSub) to avoid HTTP timeouts
- Add a simple preview view that renders the `page_map` outline into a one-page brief for download (PDF)

End of LLM Deliverables Integration doc.