BoringBiz.ai — AI-Driven Boring Business Lead Generation SaaS

Tagline: Find boring, high-intent local niches → auto-generate SEO deliverables → launch microsites → sell leads.

Purpose of this doc: a single, Gamma-importable, developer+product playbook combining the *Microsite Empire* operational manual and the *StepLock Protocol* into a concrete Firebase + Next.js implementation. Copy this markdown into a file `boringbiz-gamma-merged.md` or import directly into Gamma.

Table of contents

---

1) Overview & positioning

BoringBiz.ai helps agencies, freelancers, and small teams find under-served local niches ("boring businesses") and rapidly generate SEO deliverables and microsite briefs to capture high-intent leads. The product combines automated scraping, StepLock keyword mapping, and LLM-driven deliverable generation — delivered via a friendly dashboard.

Primary users: solo operators, local agencies, franchise marketers, business owners.

Business model: freemium → paid plans for unlimited audits, deliverables generation, Airtable/Notion sync, and agency features.

---

2) Microsite Empire phases → App modules

- **Phase 0 — Market selection (Lead Finder)**
- Module: Lead Finder UI — discover local markets and verticals.
- Data: city/state seed; competitors list; high-intent keywords.
- **Phase 1 — Keyword mapping (StepLock)**
- Module: Keyword Mapper service — classifies into Emergency / Service / Problem / Local.
- **Phase 2 — Microsite planning**
- Module: Microsite Planner (generates pages matrix: services × locations, content briefs).
- **Phase 3 — Build & deploy (MVP: brief generation)**
- Module: Deliverables exporter — JSON/PDF briefs, content outlines, schema.
- **Phase 4 — Traffic & capture**
- Module: Dashboard + tracking (lead potential estimates, saved audits, competitor hotlists).
- **Phase 5 — Scale & repeat**
- Module: Batch-operations (generate 50 microsites), integrations (Airtable/Notion/GitHub for pushes).

---

3) StepLock Protocol (concise)

StepLock is a deterministic keyword taxonomy that maps raw site content into four priority buckets:

- **Emergency** — keywords indicating urgent need ("mobile diesel repair", "24/7 tow truck")
- **Service** — transactional services ("AC repair", "window replacement")
- **Problem** — informational/problem queries ("AC not cooling")
- **Local** — geo modifiers and landmark-based queries ("near me", "Charlotte I-77 truck stop")

Scoring & Prioritization: assign higher weight to Emergency > Service > Local > Problem for lead-capture features. Use StepLock to prioritize microsite pages.

---

4) Product features & user flows

Main flow (MVP)

1. User pastes competitor URL into Dashboard → presses Analyze.
2. Backend scrapes page (meta, H1, body, links, counts) → saves audit to Firestore.
3. StepLock mapper runs over body to produce keyword buckets.
4. Optional: Notion/Airtable logs created.
5. UI displays audit + quick actions: Generate brief, Export JSON.

Admin flows

- Settings: add Notion DB ID, Airtable base, LLM API key (Claude/OpenAI).
- Batch: upload CSV of competitor URLs → queued audits (future).

---

5) Tech stack & deployment

- Next.js 14 (App Router) + Tailwind
- Firebase Hosting, Cloud Functions (Node 20 + TypeScript)
- Firestore (collections: `audits`, `projects`, `users` later)
- Optional: Notion API, Airtable API, Claude/OpenAI LLMs

Deployment: Firebase Studio or CLI (`firebase deploy --only functions,hosting`).

---

6) Full scaffold (folder tree)

```
boringbiz-firebase-starter/
  ├── firebase.json
  ├── .firebaserc
  ├── firestore.rules
  ├── storage.rules
  ├── README.md
  ├── functions/
  │   ├── package.json
  │   ├── tsconfig.json
  │   └── src/
  │       ├── index.ts
  │       ├── stepLockMapper.ts
  │       ├── notionClient.ts
  │       └── airtableClient.ts
  └── web/
      ├── package.json
      ├── next.config.mjs
```

```
├── postcss.config.js
├── tailwind.config.js
├── .env.local.example
└── app/
    ├── globals.css
    ├── layout.tsx
    ├── page.tsx
    └── api/
        └── audit/route.ts
```

---

7) Cloud Functions (full files)

`functions/src/stepLockMapper.ts`

```ts
export function stepLockMapText(text: string){
  const t = text.toLowerCase();
  const emergency = [];
  const service = [];
  const problem = [];
  const local = [];

  const emergencyTerms = ['24/7','24 hour','emergency','mobile'];
  const serviceTerms = ['repair','install','service','replace','maintenance'];
  const problemTerms = ['not working','won\'t','broken','leak','won\'t start'];
  const localTerms = ['near me','\bin\s','\bcharlotte\b','\bnew york\b'];

  emergencyTerms.forEach(w => { if(t.includes(w)) emergency.push(w); });
  serviceTerms.forEach(w => { if(t.includes(w)) service.push(w); });
  problemTerms.forEach(w => { if(t.includes(w)) problem.push(w); });
  localTerms.forEach(w => { if(t.includes(w)) local.push(w); });

  return { emergency, service, problem, local };
}
```

`functions/src/notionClient.ts`

```ts
export async function logAuditToNotion({ notionDbId, notionToken, title, url,
projectId, wordCount }:{ notionDbId?:string, notionToken?:string,
title?:string, url:string, projectId?:string, wordCount?:number }){
  if (!notionDbId || !notionToken) return;
  const { Client } = await import('@notionhq/client');
  const notion = new Client({ auth: notionToken });
  await notion.pages.create({
```

```
    parent: { database_id: notionDbId },
    properties: {
      Name: { title: [{ text: { content: title || url } }] },
      URL: { url },
      Project: { rich_text: [{ text: { content: projectId || 'default' } }] },
      WordCount: { number: wordCount || 0 }
    }
  });
}
```

`functions/src/airtableClient.ts`

```
import Airtable from 'airtable';
export async function upsertCompetitorToAirtable({ baseId, token, url, title }:
{ baseId?:string, token?:string, url:string, title?:string }){
  if (!baseId || !token) return;
  const base = new Airtable({ apiKey: token }).base(baseId);
  await base('Competitors').create([{ fields: { URL: url, Title: title ||
url } }]);
}
```

`functions/src/index.ts` **(audit endpoint)**

```
import * as functions from 'firebase-functions';
import * as admin from 'firebase-admin';
import axios from 'axios';
import * as cheerio from 'cheerio';
import metascraper from 'metascraper';
import metascraperTitle from 'metascraper-title';
import metascraperDescription from 'metascraper-description';
import metascraperUrl from 'metascraper-url';
import { logAuditToNotion } from './notionClient.js';
import { upsertCompetitorToAirtable } from './airtableClient.js';
import { stepLockMapText } from './stepLockMapper.js';

const scraper = metascraper([metascraperTitle(), metascraperDescription(),
metascraperUrl()]);

admin.initializeApp();
const db = admin.firestore();

function validUrl(u: string): boolean { try { new URL(u); return true; } catch
{ return false; } }

export const auditUrl = functions
```

```
  .region('us-central1')
  .runWith({ memory: '512MB', timeoutSeconds: 120 })
  .https.onRequest(async (req, res) => {
    res.set('Access-Control-Allow-Origin','*');
    res.set('Access-Control-Allow-Headers','Content-Type');
    if (req.method === 'OPTIONS') return res.status(204).send('');

    try {
      const { url, projectId = 'default', notionDbId, notionToken,
airtableBaseId, airtableToken } = req.body || {};
      if (!url || !validUrl(url)) return res.status(400).json({ error: "Provide
a valid 'url'." });

      const resp = await axios.get(url, { timeout: 20000, headers: { 'User-
Agent':'Mozilla/5.0' } });
      const html = resp.data as string;
      const $ = cheerio.load(html);
      const meta = await scraper({ html, url });

      const h1 = $('h1').first().text().trim();
      const title = meta.title || $('title').text().trim();
      const description = meta.description || $
('meta[name="description"]').attr('content') || '';
      const bodyText = $('body').text().replace(/\s+/g,' ').trim();
      const wordCount = bodyText.split(' ').filter(Boolean).length;
      const imagesCount = $('img').length;
      const scriptsCount = $('script').length;
      const hasWebp = $("img[src$='.webp']").length > 0;
      const links = Array.from(new Set($('a[href]').map((_,a)=>String($
(a).attr('href'))).get())).slice(0,100);

      const stepLock = stepLockMapText(bodyText);

      const doc = {
        url, projectId,
        fetchedAt: admin.firestore.FieldValue.serverTimestamp(),
        title, description, h1,
        wordCount,
        speedHints: { imagesCount, scriptsCount, hasWebp },
        links,
        stepLock
      };

      const ref = await db.collection('audits').add(doc);

      await logAuditToNotion({ notionDbId, notionToken, title, url, projectId,
wordCount });
      await upsertCompetitorToAirtable({ baseId: airtableBaseId, token:
```

```
airtableToken, url, title });

      // TODO: Call Claude/OpenAI with the scraped content to produce
`deliverables` JSON
      // Example: deliverables = await callLLM({ url, bodyText, stepLock,
links })
      // await db.collection('audits').doc(ref.id).update({ deliverables });

      return res.json({ id: ref.id, ...doc });
    } catch (e:any) {
      console.error(e);
      return res.status(500).json({ error: e.message || 'Failed to audit
URL' });
    }
  });
```

---

8) Web app code (full files)

`web/app/page.tsx`

```
'use client';
import { useState } from 'react';

export default function Page(){
  const [url,setUrl]=useState('');
  const [loading,setLoading]=useState(false);
  const [data,setData]=useState<any>(null);
  const [err,setErr]=useState<string|undefined>();

  async function run(){
    try{
      setLoading(true); setErr(undefined); setData(null);
      const r = await fetch('/api/audit',{method:'POST',headers:{'Content-
Type':'application/json'},body:JSON.stringify({url,projectId:'default'})});
      const j = await r.json();
      if(!r.ok) throw new Error(j.error||'Failed');
      setData(j);
    }catch(e:any){ setErr(e.message) }finally{ setLoading(false) }
  }

  return (
    <main className="max-w-4xl mx-auto p-6 space-y-6">
      <header className="flex items-center justify-between">
        <div>
```

```
          <h1 className="text-2xl font-semibold">BoringBiz.ai</h1>
          <p className="text-sm text-gray-500">Paste a competitor URL → get
instant audit signals.</p>
        </div>
        <button className="btn btn-accent" onClick={()=>alert('Upgrade flow
TBD')}>Upgrade</button>
      </header>

      <section className="card p-4 space-y-3">
        <div className="flex gap-2">
          <input className="flex-1 border rounded-lg px-3 py-2"
placeholder="https://competitor.com" value={url}
onChange={e=>setUrl(e.target.value)}/>
          <button className="btn btn-primary" onClick={run} disabled={loading}
>{loading?'Analyzing…':'Analyze URL'}</button>
        </div>
        {err && <p className="text-red-600 text-sm">{err}</p>}
      </section>

      {data && (
        <section className="card p-4 space-y-2">
          <div className="font-semibold">{data.title || data.url}</div>
          <div className="text-sm text-gray-600">{data.description}</div>
          <div className="grid grid-cols-3 gap-3 mt-2 text-sm">
            <div className="p-3 bg-green-50 rounded">Words: <b>{data.wordCount}
</b></div>
            <div className="p-3 bg-yellow-50 rounded">Imgs:
<b>{data.speedHints?.imagesCount}</b></div>
            <div className="p-3 bg-orange-50 rounded">Scripts:
<b>{data.speedHints?.scriptsCount}</b></div>
          </div>
          <details className="mt-3">
            <summary className="cursor-pointer">Links (sample)</summary>
            <pre className="text-xs bg-gray-50 p-3 rounded overflow-auto max-
h-64">{JSON.stringify(data.links,null,2)}</pre>
          </details>
        </section>
      )}
    </main>
  );
}
```

web/app/api/audit/route.ts

```
import { NextRequest, NextResponse } from 'next/server';
```

```
export async function POST(req: NextRequest){
  const payload = await req.json();
  const url = `https://us-central1-YOUR_PROJECT_ID.cloudfunctions.net/auditUrl`;
  const r = await fetch(url, { method:'POST', headers:{'Content-
Type':'application/json'}, body: JSON.stringify(payload)});
  const data = await r.json();
  return new NextResponse(JSON.stringify(data), { status: r.status, headers: {
'Content-Type': 'application/json' } });
}
```

---

9) Integration hooks (how & where)

- **Notion**: pass `notionDbId` + `notionToken` in POST body to `auditUrl` — `notionClient.ts` writes a page.
- **Airtable**: pass `airtableBaseId` + `airtableToken` — `airtableClient.ts` creates Competitor record.
- **Claude/OpenAI**: TODO — add LLM keys to functions environment and call after scraping to produce `deliverables` JSON. Save as `audits/{id}.deliverables`.

Inline TODO in `functions/src/index.ts` included for where to call the LLM.

---

10) UI/UX maps & Reference HTML

- Navigation: Dashboard, Lead Finder, Competitors, Trends, Settings
- Primary CTA: Analyze URL -> Generate Brief
- Reference UI: (Embed your dashboard HTML in a slide or attach as code block)

(Use the Dashboard HTML from your design as a reference — include in Gamma as a code block or as an iframe preview.)

---

11) Midjourney + Ideogram prompts (copy/paste ready)

Midjourney — dashboard doodle (16:9)

```
friendly line-art doodle of a sprouting plant growing from a map pin, minimal
outlines, slight texture, playful, flat colors, brand palette #32C682 #FF785A
#F5C542 on white, vector feel --ar 16:9 --v 6
```

Midjourney — icon set (1:1)

```
set of 8 simple vector icons, rounded shapes, consistent stroke, topics: wrench,
map pin, truck, phone, star reviews, speedometer, sitemap, link, brand palette
#32C682 #FF785A #F5C542 on white --ar 1:1 --v 6
```

Ideogram — Logo Prompt

```
Wordmark logo: "BoringBiz.ai" with a small sprouting leaf over the dot on the
"i", rounded friendly sans, subtle stroke, primary #32C682, secondary accent
#FF785A, flat vector style, minimal and modern
```

---

12) How to import into Gamma

1. Save this file as `boringbiz-gamma-merged.md` locally.
2. Open https://gamma.app → New doc → Import → Upload Markdown.
3. Gamma will split headings into slides/pages. Check code blocks and expand as needed.

---

13) Next steps & checklist (short)

-

Ready to go

This merged markdown is designed to be **imported into Gamma** for a living, editable playbook. Open the canvas to review, edit, and export as slides or handoffs.

End of document