

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

01

Overview & Positioning

Introduction to BoringBiz.ai and its market position

02

Microsite Empire Phases

Phases mapped to app modules

03

StepLock Protocol

Keyword mapping + prioritization

04

Product Features

User flows and functionality

05

Tech Stack

Development and deployment details

01

Full Scaffold

Complete folder tree structure

02

Cloud Functions Code

Complete implementation files

03

Web App Code

Frontend implementation

04

Integration Hooks

Notion, Airtable, Claude/OpenAI

05

UI/UX Maps

Reference HTML and design

01

Midjourney + Ideogram

Visual asset prompts

02

Gamma Import

How to import into Gamma

03

Next Steps

Checklist and future development

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.

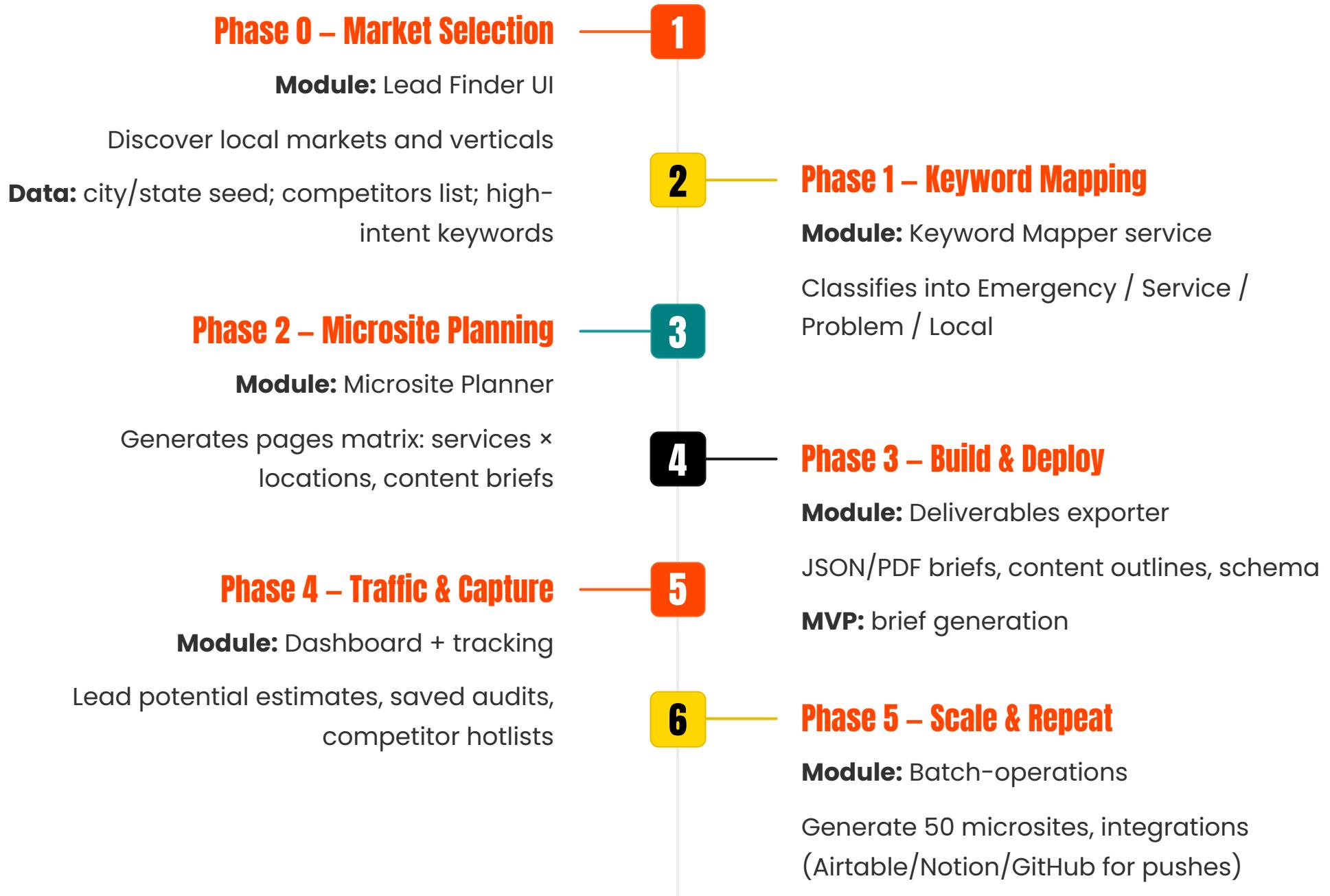
Primary Users

- Solo operators
- Local agencies
- Franchise marketers
- Business owners

Business Model

Freemium → paid plans for unlimited audits, deliverables generation, Airtable/Notion sync, and agency features.

Microsite Empire Phases → App Modules



StepLock Protocol

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

Emergency

Keywords indicating urgent need

Examples: "mobile diesel repair", "24/7 tow truck"

Service

Transactional services

Examples: "AC repair", "window replacement"

Problem

Informational/problem queries

Examples: "AC not cooling"

Local

Geo modifiers and landmark-based queries

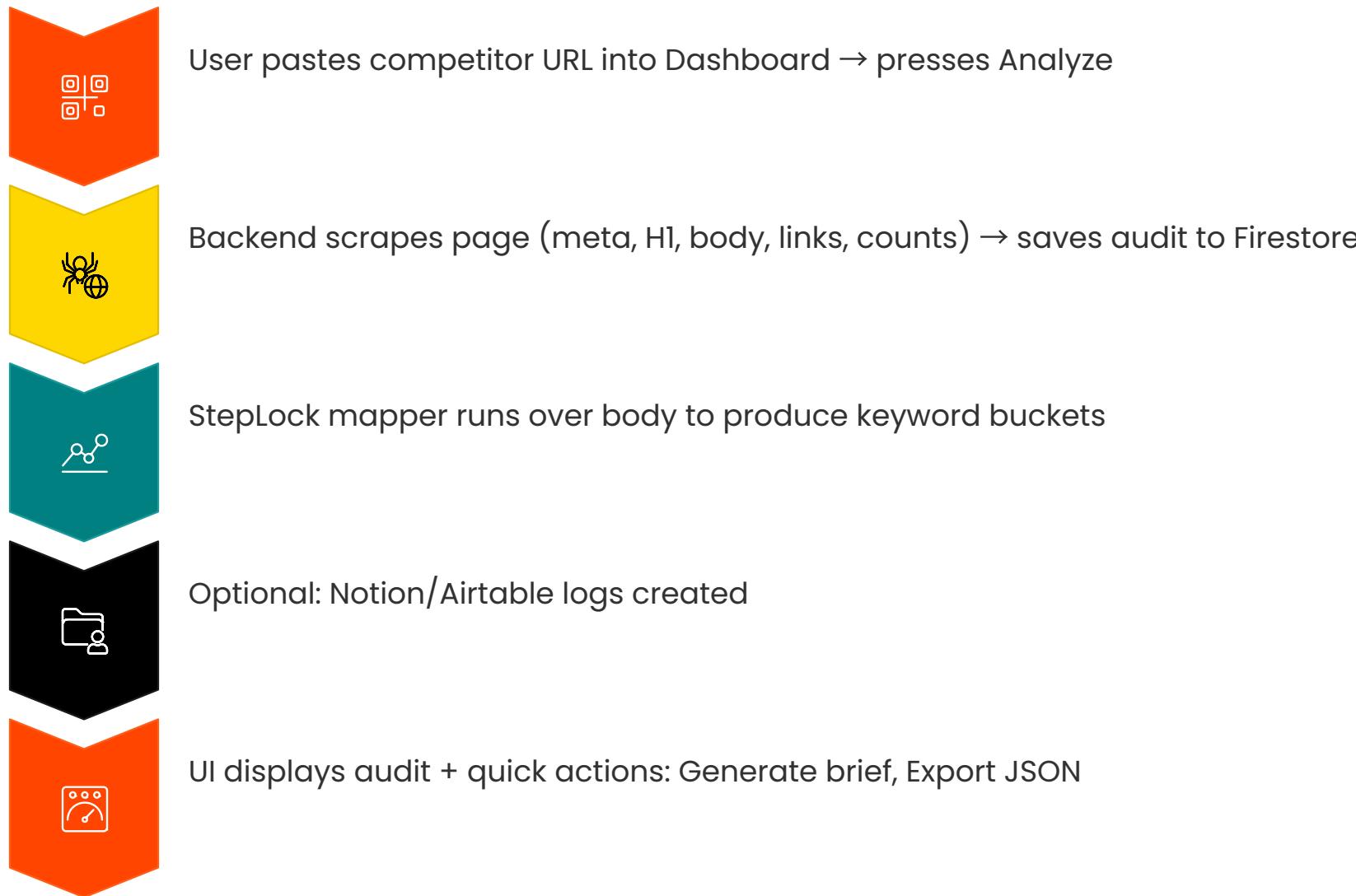
Examples: "near me", "Charlotte I-77 truck stop"

Scoring & Prioritization

Assign higher weight to Emergency > Service > Local > Problem for lead-capture features. Use StepLock to prioritize microsite pages.

Product Features & User Flows

Main Flow (MVP)



Admin Flows

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

Tech Stack & Deployment



Next.js 14

App Router + Tailwind



Firebase

Hosting, Cloud Functions (Node 20 +
TypeScript)



Firebase

Collections: audits, projects, users (later)

Optional Integrations

- Notion API
- Airtable API
- Claude/OpenAI LLMs

Deployment

Firebase Studio or CLI:

```
firebase deploy --only functions,hosting
```

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
```

The folder structure follows standard Next.js and Firebase conventions, with clear separation between frontend (web) and backend (functions) code.

Cloud Functions: stepLockMapper.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 };  
}
```

This function implements the core StepLock Protocol, categorizing text into the four priority buckets: Emergency, Service, Problem, and Local.

Cloud Functions: notionClient.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 }
    }
  });
}
```

This function handles integration with Notion, creating a new page in a specified database with audit information.

Cloud Functions: airtableClient.ts

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

This function handles integration with Airtable, creating a new record in the Competitors table with the URL and title information.

Cloud Functions: index.ts (Part 1)

```
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
```

This is the first part of the main Cloud Function that handles URL auditing, importing necessary dependencies and setting up the Firebase environment.

Cloud Functions: index.ts (Part 2)

```
.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 || !isValidUrl(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({ baselId: airtableBaseId, token:
```

Cloud Functions: index.ts (Part 3)

```
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' });  
}  
});
```

This is the final part of the auditUrl Cloud Function, which processes the scraped data, saves it to Firestore, and integrates with Notion and Airtable. It includes a TODO comment for future LLM integration.

Web App Code: 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(null);
  const [err,setErr]=useState();
  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 (

```

This is the first part of the main page component for the web application, handling state management and the URL analysis function.

```
{data.title || data.url}
```

```
{data.description}
```

Words: **{data.wordCount}**

Img: **{data.speedHints?.imagesCount}**

Scripts: **{data.speedHints?.scriptsCount}**

▼ Links (sample)

```
{JSON.stringify(data.links,null,2)}
```

Web App Code: 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' } });
}
```

Integration Hooks

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

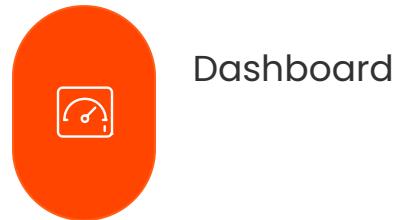
Call after scraping to produce
deliverables JSON

Save as
audits/{id}.deliverables

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

UI/UX Maps & Reference HTML

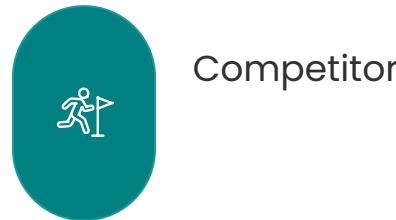
Navigation Structure



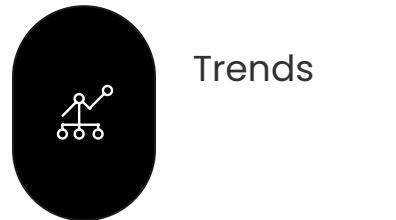
Dashboard



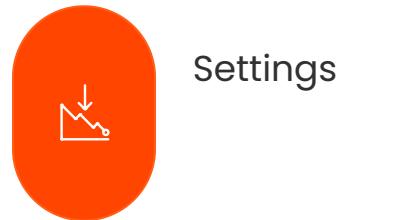
Lead Finder



Competitors



Trends



Settings

Primary CTA Flow



Analyze URL

Generate Brief

Reference UI: (Dashboard HTML from your design should be included as a code block or as an iframe preview)

Midjourney + Ideogram Prompts

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