Master Plan: SlabHaul (Pre-Production)

Section	Status	Current State
Goal	🟢 Locked	Build the "Deep Dive" for Crappie: Visual, data-rich, predictive.
User Stories	🟡 Draft	Core flows defined. Missing: Offline mode, Hardware export flows, Tournament verification.
Tech Stack	🟠 Decisions	**Flutter** (Mobile) + **Supabase** (Backend/PostGIS) + **Mapbox** (Vis).
Architecture	🔴 Critical	**Clean Architecture** required to separate "Fishing Physics" from UI.
Risks	🔴 High	**Data Quality**: DNR GPX files are dirty/inconsistent. **Surface Temp**: USGS coverage is spotty; Satellite latency.
Open Questions	🟠 Medium	How to handle "staleness" of satellite data? Can we legally scrape state DNR PDF maps if GPX is missing?
Next Steps	🟢 Action	1. Prototype the "Spider Rigging Calculator" (Logic only). 2. Manually clean *one* state's GPX data to test schema.

Executive Summary

We are building a **precision tool**, not a social network. Unlike general fishing apps (Fishbrain), SlabHaul is a utility for "meat hunters" who care about specific depth, structure, and water physics.

The Strategy: We will use **Clean Architecture** to isolate our "Fishing Intelligence" (thermocline logic, spider rigging physics) from the delivery mechanism (app). This allows us to swap data sources (e.g., changing from USGS to a paid weather API) without breaking the app. We will use **Pragmatic Programmer** principles: "Tracer Bullets" first—get *one* lake, *one* brush pile, and *one* weather report working end-to-end before scaling to 100 lakes.

The Hard Truth: Your biggest risk isn't code; it's **data**. State DNR GPX files are notoriously messy (duplicates, wrong datums, vague names). If we don't build a robust "Data Ingestion Pipeline" (ETL) early, the map will look like garbage, and users will churn immediately.

Phase 1: Specification Lock & "Tracer Bullet" Design

Objective: Validate the "Physics" and "Data" before drawing a single UI pixel.

Task	Description	Dependency	Risk
1.1 Data Schema Design	Define Supabase tables: lakes, attractors (PostGIS point), weather_snapshots, techniques.	None	**High**: Must support different DNR formats.
1.2 Physics Logic (No UI)	Write the *pure logic* for the **Spider Rigging Calculator** (Physics Entity). Input: weight, speed, line type. Output: Depth. Test with unit tests.	None	**Med**: Physics must be accurate or pros will mock it.
1.3 The "Data Tracer"	Manually download *one* GPX file (e.g., TN Tech). Write a script to clean it and insert it into Supabase. Display it on a basic Mapbox web map.	1.1	**High**: This proves if the data is usable.
1.4 Privacy Audit	Define exactly what user data we store (catches/waypoints). **Decision:** Store waypoints locally on device first? (Privacy vs. Sync trade-off).	None	Low

- **Tools:** Python (for GPX parsing), Supabase SQL Editor.
- **Ras Mic Planning:** Use the "Ask User Question Tool" technique to rigorously interview yourself about the *exact* inputs for the Spider Rig Calculator (e.g., "Do we account for line diameter? Current drag?").

Phase 2: UX/UI Design (The "Meat Hunter" Interface)

Objective: Design for bright sun, wet hands, and old phones.

Task	Description	Dependency
2.1 High-Contrast Mode	Design map markers for visibility in direct sunlight (High Contrast Yellow/Black).	None
2.2 "One-Thumb" Flows	Ensure the "Spider Rig Calculator" can be operated with one hand while driving a boat.	1.2
2.3 Map Layering UX	Design how Wind, Temp, and Attractors stack without cluttering the screen.	None

Task	Description	Dependency
2.4 Wireframe Prototype	Full click-through in Figma. **Test:** Give it to a 50-year-old angler. If they squint, redesign.	2.1

- **AI Workflow:** Use **Gemini/Claude** to generate "User Personas" (e.g., "Tournament Tim" vs. "Weekend Wally") and critique the UI flow against their needs.

Phase 3: Tech Stack & Architecture (The Foundation)

Objective: Set up a scalable, pivot-ready codebase.

Decision Log: Devil's Advocate

- **Mobile Framework: Flutter** (Winner) vs. React Native.
  - *Why Flutter?* The **Impeller** rendering engine is superior for map overlays and 60fps animations on mid-range Android devices (common in our demographic). We need "butter smooth" map panning.
  - *Why not RN?* Bridge overhead *can* stutter with thousands of map markers unless optimized heavily.
- **Backend: Supabase** (Winner) vs. Firebase.
  - *Why Supabase?* **PostGIS**. You cannot build a serious "Find brush piles near me" feature without true geospatial SQL queries. Firebase GeoFire is clunky.
  - *The "Moat":* SQL allows complex queries like "Show me brush piles < 20ft deep within 5 miles of a boat ramp."

Task	Description	Est. Cost
3.1 Repo Setup	Monorepo: slabhaul_app (Flutter), slabhaul_cloud (Supabase functions), slabhaul_etl (Python data scripts).	$0
3.2 Mapbox Integration	Initialize Mapbox GL in Flutter. Set up style URLs.	Free Tier
3.3 Auth & Profiles	Supabase Auth (Email + Google/Apple).	Free Tier

---

Phase 4: MVP Development (The "Core 5")

Objective: Build the features that justify the $49.99 price tag.

Task	Feature	Complexity	Notes
4.1 Attractor Map	**Public Attractor Map**	Med	Connect Supabase PostGIS to Mapbox. Implement clustering (1000s of points).
4.2 Weather Widget	**Weather for Anglers**	Low	Integrate OpenWeatherMap or NWS API. Focus on **Barometric Pressure** graph.
4.3 Hydro Tracker	**Lake Level & Temp**	High	Integrate USGS Water Services API. **Risk:** Surface temp is often missing. Fallback to Sentinel-3 satellite API (latency ~3hrs) or nearest airport temp.
4.4 Calculator UI	**Spider Rig Calc**	Low	Connect the Phase 1 Logic Entity to a Flutter UI slider input.
4.5 Wind Overlay	**Basic Wind Map**	Med	Mapbox "Wind" layer or simple arrow vectors based on grid forecast.

- **Dev Principle:** "Orthogonality." If the USGS API changes, it should *only* break the `USGS_Adapter` file, not the UI.

Phase 5: The Intelligence Layer (Differentiation)

Objective: Turn data into "Cheating." (Phase 2 Features)

Task	Feature	Complexity	Strategy
5.1 Thermocline AI	**Thermocline Predictor**	**Extreme**	Build a heuristic model: `Depth = f(SurfaceTemp, Clarity, Season)`. Use "Expert Systems" logic (if-this-then-that) rather than Machine Learning initially.
5.2 Mudline Map	**Water Clarity**	High	Use Sentinel-2/Landsat imagery processing (NDVI/Turbidity). **Note:** This is expensive to compute. Start with manual updates for top 10 lakes.
5.3 Dam Schedules	**Generation Schedule**	Med	Scrape TVA/USACE sites. They rarely have clean APIs. Expect brittle scrapers.

Phase 6: Polish, Testing & Hardware Integration

Objective: Make it "Pro" ready.

Task	Description
6.1 Hardware Export	Generate .gpx and .usr (Lowrance) files from the user's saved waypoints or selected public piles.
6.2 Offline Mode	Mapbox Offline Manager. Download specific lake regions. Essential for rural lakes.
6.3 Payment Rails	RevenueCat integration (manages Apple/Google sub logic).
6.4 Beta Testing	"Creekside Chat" program: Give 50 local guides free lifetime access in exchange for brutal feedback.

Gantt Chart (Estimates)

Month	Focus	Key Deliverable
M1	**Specs & Data**	Validated GPX schema, Spider Rig logic, Figma Proto.
M2	**Foundation**	Flutter Mapbox working, Supabase connected.
M3	**MVP Core**	Attractor Map populated (5 states), Weather widget.
M4	**MVP Tools**	Spider Rig Calc, Lake Levels (USGS).
M5	**Alpha Test**	Internal testing on the water. Fix UI glare issues.
M6	**Intelligence**	Thermocline Logic v1.
M7	**Polish**	Offline maps, RevenueCat, Hardware Export.
M8	**Launch**	Release v1.0 to App Stores.

Financial Estimates (Year 1)

Item	Est. Cost	Notes
Mapbox	$0 - $500/mo	Free up to 50k MAUs (generous). Scales with usage.
Supabase	$25/mo	Pro tier needed for production backups/PostGIS.
Weather API	$40 - $150/mo	NWS is free; OpenWeatherMap/Aeris is better/paid.
RevenueCat	% of Sales	Free until $10k/mo revenue.
Dev Accounts	$99 + $25	Apple + Google one-time/annual fees.

Item	Est. Cost	Notes
Marketing	$1,000+	Initial ads, influencer (guide) seeding.
Total Overhead	**~$200/mo**	Very lean.

Immediate Action Plan (Next 4 Weeks)

1. **Week 1 (Data Scavenger Hunt):**
    - Download GPX files for **Lake Grenada (MS)** and **Kentucky Lake (KY/TN)**.
    - Write a Python script to parse/normalize them.
    - *Result:* A clean CSV/JSON ready for database import.
2. **Week 2 (The Physics Logic):**
    - Create the `SpiderRiggingCalculator` class in Dart (pure Dart, no Flutter).
    - Unit test: `calcDepth(speed: 0.8, weight: 1oz, line: 10ft) == ?`
    - *Result:* Verified math.
3. **Week 3 (Map Spike):**
    - Spin up a "Hello World" Flutter app with Mapbox.
    - Load the Week 1 data onto the map.
    - *Result:* Visual proof of concept.
4. **Week 4 (Weather Integration):**
    - Connect to USGS API for one gauge.
    - Display "Current Level" vs. "Summer Pool."
    - *Result:* Real-time data pipeline active.

Sandbox Planner Readiness Score: 15/100

We are NOT ready for code. We are ready for **Data Validation**.

Blocking the next 10 points:

1. **DNR Data Reality Check:** You haven't confirmed if the state DNR files actually exist for *all* target states or if they are PDFs we have to scrape. **Check this immediately.**
2. **Surface Temp Source:** We claimed "Critical Addition: Surface Temperature" but haven't identified a reliable, real-time API source for it. USGS is spotty.
3. **Thermocline Math:** We have a concept, but no formula. We need to interview a guide or find a biological paper to base the algorithm on.

Would you like me to start the "Data Scavenger Hunt" by generating the Python script to normalize a sample GPX file, or should we deep-dive into the "Thermocline Logic" first?