1️⃣ MASTER PRD PROMPT (Idea → Architecture → Structure → Execution)

Use this as the FIRST prompt after you finalize the problem statement.

🔷 Prompt: PRD + Architecture + Repo Structure

You are a senior product architect + tech lead.

Context:

- Hackathon project
- Team of 3
- Frontend: Next.js (App Router, TypeScript, Tailwind)
- Backend: Supabase (Auth, Database, Storage, Edge Functions if needed)
- Auth: Supabase Email + Google OAuth
- 3D / Pricing visuals: Spline
- Payments: Stripe (or best suitable gateway)
- Deployment: Vercel (frontend), Supabase (backend)

TASK:

Create a COMPLETE PRD that the entire team will follow strictly.

PRD MUST INCLUDE:

1. PRODUCT IDEA
   - Problem statement (real-world, hackathon-friendly)
   - Target users
   - Pain points
   - Why existing solutions are weak
   - Core idea (1–2 sentences)
   - Driving force / innovation (what makes this stand out)

2. SOLUTION BREAKDOWN
   - How the product solves the problem step-by-step
   - User journey (from landing → auth → core feature → payment if any)

- Key features (MVP vs nice-to-have)

3. TECH STACK (WITH PURPOSE)

For each technology:

- Why it is chosen
- What responsibility it handles

Example:

- Next.js → UI, routing, SEO
- Supabase Auth → Authentication & session handling
- Supabase DB → User data, profiles, app data
- Spline → Interactive pricing / feature visualization
- Stripe → Secure payments

4. GLOBAL PROJECT FOLDER STRUCTURE (MANDATORY)

- One mono-repo
- Clear separation
- Stable structure to avoid merge conflicts

REQUIRED ROOT STRUCTURE:

```
/
 ├── frontend/
 |   ├── app/
 |   |   ├── (auth)/
 |   |   ├── dashboard/
 |   |   ├── profile/
 |   |   ├── pricing/
 |   |   └── api/
 |   ├── components/
 |   ├── lib/
 |   ├── hooks/
 |   ├── styles/
 |   └── types/
 |
 ├── backend/
 |    ├── supabase/
 |   |    ├── migrations/
 |   |    ├── seed.sql
 |   |   └── functions/
 |    ├── schema/
 |   └── policies/
 |
 ├── docs/
 |    ├── prd.md
 |    ├── api-contracts.md
 |   └── frontend-backend-sync.md
 |
 └── README.md
```

Explain each folder's responsibility.

5. DATABASE DESIGN (SUPABASE)

- Tables
- Fields
- Relations
- Auth ↔ Profile linking
- RLS policy overview

6. API CONTRACTS (IMPORTANT)

- Define frontend ↔ backend data contracts
- Example:
  - getUserProfile(userId) → returns { name, avatar, role }
- This is NON-NEGOTIABLE to keep sync

7. DEVELOPMENT RULES

- No direct DB access from frontend except via Supabase client
- All pages must have backend readiness before UI finalization
- No last-moment integration

8. DEPLOYMENT & ENV STRATEGY

- Env variables
- Local vs prod

OUTPUT FORMAT:

- Clean markdown
- Headings
- Diagrams in ASCII if needed
- No vague statements
- Hackathon-optimized

2️⃣ FRONTEND ↔ BACKEND SYNC ROADMAP (Excel-style)

This is **critical** for parallel work without chaos.

🔷 Prompt: Parallel Development Roadmap (Excel / Sheet)

You are a technical project manager.

TASK:

Create a DEVELOPMENT ROADMAP in TABULAR FORMAT suitable for Excel / Google Sheets.

GOAL:

- Frontend and Backend teams work in parallel
- Each feature is designed, implemented, and connected immediately
- No "connect everything at the end" workflow

TABLE COLUMNS (MANDATORY):

1. Feature Name
2. Page / Module
3. Frontend Tasks
4. Backend Tasks (Supabase)
5. Database Tables Involved
6. Auth Required (Yes/No)
7. API / Supabase Function
8. Integration Checkpoint
9. Status

RULES:

- Start with Auth (Login / Signup)
- Then Profile
- Then Core Feature
- Then Pricing & Payment
- Then Dashboard

EXAMPLE ROW LOGIC:

- Login Page:
  - Frontend builds UI
  - Backend sets up Supabase Auth + OAuth
  - Integration happens BEFORE moving to next feature

OUTPUT:

- Markdown table
- Clear, short tasks
- No abstract wording
- Hackathon-speed optimized

3️⃣ **FRONTEND UI + BUTTON VERIFICATION AGENT PROMPT**

This is your **quality-control agent**.

🔷 **Prompt: Frontend Integrity & Backend Connectivity Checker**

You are a strict frontend QA + system integration agent.

INPUT:

- Next.js frontend codebase
- UI designs generated from Spline / HTML templates

TASK (FOR EACH PAGE):

1. PAGE ANALYSIS
   - Identify all buttons, links, CTAs, icons
   - Count them
   - List them explicitly

2. RELEVANCE CHECK

   For EACH button:
   - Is it necessary?
   - Does it match the product goal?
   - If not → flag for removal

3. NAVIGATION CHECK
   - Does the button route to an existing page?
   - If not:
     - Specify missing page
     - Suggest route name

4. BACKEND CONNECTIVITY CHECK

   For buttons requiring data:
   - Is Supabase connected?
   - Is auth enforced?
   - Is the database ready?

- If missing → list exact backend requirement

5. UI CONSISTENCY RULES
   - Same theme
   - Same spacing
   - Same typography
   - No random styles

6. BLOCKING RULE
   ❌ A page is NOT considered complete unless:
   - All buttons are functional
   - All backend-dependent buttons are wired
   - No dead links

OUTPUT:

- Per-page report
- Action items
- Errors marked as CRITICAL / WARNING

4️⃣ FRONTEND DESIGN RESTRICTION / REQUIREMENTS DOC PROMPT

This prevents random UI mess.

🔷 Prompt: Frontend UI Rules Document

You are a design system enforcer.

TASK:

Create a FRONTEND UI REQUIREMENTS DOCUMENT that must be followed strictly.

INCLUDE:

1. Color system
2. Typography rules
3. Button styles

4. Layout grid

5. Animation rules

6. Spline integration rules

7. Accessibility rules

8. Responsive rules

RULE:

If a UI element violates this document → it must be rejected.

OUTPUT:

- Short

- Strict

- Non-negotiable

5️⃣ FINAL: PRE-HACKATHON PREP (DO THIS TONIGHT)

✅ Non-Negotiable Setup

- One GitHub repo created
- Branches:
  - main
  - frontend
  - backend
- Supabase project created
- Env vars template ready
- README with basic instructions

✅ Decide BEFORE Hackathon

- Auth method (email + Google)
- Payment gateway
- Deployment target
- Naming conventions

✅ Team Rules (Say this out loud)

- No solo features

- No skipping API contracts
- No UI without backend readiness
- Every feature = design → backend → connect → move on