PRODUCT REQUIREMENTS DOCUMENT (PRD)

AI-Powered No-Code / Low-Code SaaS Builder Platform

---

1. PRODUCT OVERVIEW

1.1 Product Vision

- Build an **end-to-end AI SaaS Builder** that replaces:
    - UI design tools (Figma / Framer)
    - Backend development
    - Database consoles (Supabase / Neon)
    - Integration glue code
    - CI/CD and deployment pipelines
- Enable users to visually design, connect, deploy, and scale **real production software**.

1.2 Core Philosophy

- Visual authoring, **deterministic execution**
- JSON as single source of truth
- Code is generated, not guessed
- AI assists, never bypasses rules
- Frontend and backend deploy independently

---

2. HIGH-LEVEL ARCHITECTURE

2.1 Project Model

- One **Project** = One Product
- One project contains:
    - Multiple frontend workspaces
    - One backend system
    - One or more database groups
    - Multiple deployment pipelines

2.2 Core Modules

1. Authentication Module
2. Project & Workspace Module
3. Design Module (Frontend Builder)
4. Integration Module (UI ↔ Backend)
5. Backend Module (Visual Microservices)
6. Database Module
7. Deployment Module (CI/CD)
8. AI Agent System
9. Observability & Admin

---

3. AUTHENTICATION MODULE

3.1 Features

- Email + password auth
- OAuth (Google, GitHub)
- OTP login
- 2FA (TOTP)
- Password reset
- Magic links
- Role-based access control (RBAC)

3.2 Expected Behavior

- Secure token-based authentication (JWT + refresh)
- Auth guards available as backend nodes
- Auth state available to frontend integration module

3.3 Dependencies

- Required by Project, Backend, Integration, Deployment modules

---

4. PROJECT & WORKSPACE MODULE

4.1 Project

- Create, edit, archive projects
- Each project is fully isolated

4.2 Workspace (Frontend)

- Each workspace represents **one frontend app**
- Supported workspace types:
    - Mobile App
    - Web App
    - Website
    - Admin Dashboard

4.3 Workspace Properties

- Framework (Next.js, React Native, etc.)
- Routing mode
- Theme
- Deployment target
- AI agent context

4.4 Dependencies

- Design Module operates per workspace
- Integration Module is workspace-scoped
- Deployment is per workspace

---

5. DESIGN MODULE (CORE FRONTEND BUILDER)

5.1 Purpose

- Visual UI builder similar to Figma + Framer
- Produces **structured UI data**, not code

5.2 Core Capabilities

- Multi-screen design per workspace
- Reusable components
- Auto layout (row, column, grid)

- Responsive constraints
- Design / Animate / Prototype modes
- Real-time preview

5.3 Design Data Model

- Workspace JSON
- Screen JSON
- Component tree
- Style metadata

5.4 Style Properties (Must Support)

- Padding
- Margin
- Gap
- Width / height (fill, hug, fixed)
- Border
- Border radius
- Background
- Typography

5.5 Screen Features

- Route definition
- Auth requirement flag
- Grouping (Auth, Checkout, Admin, etc.)
- Version history
- Rollback

5.6 Transition & Prototype Rules

- Screen navigation
- Conditional navigation (success / error)
- Loading, error, success states
- Gesture support (mobile)

5.7 Design → Runtime

- Design JSON rendered live using a renderer
- No file generation during editing

5.8 Design → Code (Deploy Time)

- Design JSON compiled into real Next.js project
- Clean component-based output
- No inline CSS unless controlled

5.9 Dependencies

- Integration Module consumes screen definitions
- Deployment Module compiles design into build artifacts

---

6. INTEGRATION MODULE (FRONTEND ↔ BACKEND)

6.1 Purpose

- Connect UI events to backend endpoints
- Own all frontend-backend contracts

6.2 Core Features

- Bind UI events (click, submit, load)
- Map UI inputs → backend inputs
- Map backend outputs → UI state
- Loading, success, error handling
- Mock vs real backend toggle
- Integration testing

6.3 Integration Rules

- Frontend never calls backend directly
- All calls go through generated integration hooks
- Schema validation required

6.4 Integration Data Model

- Screen ID
- UI event
- Backend main node reference
- Input mapping
- Output mapping
- Side effects (navigation, state update)

6.5 Dependencies

- Depends on Backend Main Nodes
- Used by Design Module screens
- Used by Deployment for runtime wiring

---

7. BACKEND MODULE (VISUAL → EXECUTABLE)

7.1 Purpose

- Replace custom backend development
- Visual authoring of real microservices

7.2 Backend Structure

- One backend per project
- Multiple logical services
- Each service contains endpoints

7.3 Endpoint Model

- Each endpoint has:
    - HTTP method
    - Path
    - Input schema
    - Output schema
    - One Main Node

7.4 Main Node

- Entry point for endpoint
- Contract between frontend and backend
- References a workflow definition

7.5 Workflow Builder

- Visual node-based editor
- Directed execution graph
- Deterministic order

7.6 Node Types (Required)

- Input parser
- Validation
- Rate limiting
- Auth guard
- Database CRUD
- Logic (if, switch)
- Utility
- Error handler
- AI agent node
- Custom code node

7.7 Node Execution Rules

- Visual nodes do not execute
- Node executors execute (TypeScript)
- Context-based data flow
- Strict input/output contracts

7.8 Runtime Engine

- Executes workflows step by step
- Handles retries, errors, logging
- Supports versioned workflows

7.9 Debugging

- Step-by-step execution logs

- Input/output inspection
- Replay support

7.10 Dependencies

- Integration Module consumes endpoint contracts
- Database Module used via DB nodes
- Deployment Module packages backend services

---

8. DATABASE MODULE

8.1 Purpose

- Visual database management
- Replace Supabase / Neon consoles

8.2 Database Model

- Database Groups per project
- Environments:
    - Test
    - Dev
    - Staging
    - Production

8.3 Supported Providers

- Supabase (Postgres)
- Neon (Serverless Postgres)
- Self-hosted Postgres

8.4 Features

- Visual schema editor
- Table, column, relationship management
- SQL editor
- Views, functions, triggers
- Row-level security (RLS)

- Migrations & versioning
- Backups & restore
- Performance monitoring

8.5 Safety Rules

- Production is read-only by default
- Manual approval for prod migrations
- Schema diff detection

8.6 Dependencies

- Backend Module uses DB abstraction layer
- Integration Module maps UI data models

---

9. DEPLOYMENT MODULE (NODE-BASED CI/CD)

9.1 Purpose

- Replace GitHub Actions, Jenkins, DevOps scripts
- Abstract AWS from users

9.2 Deployment Targets

- Frontend workspaces (separate)
- Backend services
- Background workers

9.3 Pipeline Builder

- Visual node-based CI/CD
- Preset pipelines
- Custom pipelines

9.4 Pipeline Nodes

- Source
- Install
- Build
- Test
- Security scan
- Deploy
- Verify
- Rollback

9.5 Environments

- Dev
- Staging
- Production

9.6 Features

- Blue-green deployments
- Canary releases
- One-click rollback
- Artifact registry
- Terminal access
- Logs & metrics
- Cost visibility

9.7 AI Deployment Agent

- Explains failures
- Suggests pipeline improvements
- Cannot deploy automatically

9.8 Dependencies

- Consumes frontend build output
- Consumes backend services
- Uses database environments

---

10. AI AGENT SYSTEM

10.1 Agent Hierarchy

- Project Agent
- Workspace Agent
- Screen Agent
- Backend Agent
- Integration Agent
- Deployment Agent

10.2 AI Capabilities

- Generate screens
- Generate backend workflows
- Suggest integrations
- Optimize pipelines
- Explain errors

10.3 AI Safety Rules

- Cannot bypass schemas
- Cannot deploy without approval
- Cannot modify prod directly
- Outputs must validate

---

11. OBSERVABILITY & ADMIN

11.1 Logging

- Backend logs
- Workflow logs
- Deployment logs

11.2 Metrics

- API latency
- Error rates
- Resource usage

11.3 Audit

- Who changed what
- When
- Rollbacks

---

12. NON-FUNCTIONAL REQUIREMENTS

12.1 Performance

- No runtime JSON interpretation in production
- All code compiled ahead of time

12.2 Security

- Zero trust between services
- Secrets vault
- RBAC everywhere

12.3 Scalability

- Horizontal scaling
- Workspace isolation
- Microservice-ready

---

13. FINAL SUCCESS CRITERIA

- Users can build full products without custom code
- Generated apps are readable, debuggable, and scalable
- Frontend and backend deploy independently
- AI accelerates but never breaks systems