Part 1: Overall Goal & Architectural Prerequisite

1.1. Project Goal

This document outlines the MVP for the "Campaigns" Engine. The goal is to build a single, unified tool within our existing Admin Panel that allows a non-technical team to create, configure, and deploy all dynamic content (Smart Widgets, Popups, and Content Collections) across the site without touching the codebase.

1.2. Core Architectural Prerequisite: The "Stubbed Tenant" Pattern

To ensure this MVP is future-proof for multi-tenancy *without* the overhead of building it now, all work **must** adhere to the following "Conscious Foundation."

Epic	User Story & Acceptance Criteria
Foundation: Schema	**User Story 1.1: Create `tenants` Table** **As a** Developer, **I want to** add a `tenants` table to the Drizzle schema **so that** all future data has a master record to link to. **AC:** 1.1.1. Create a `tenants` table in `shared/schema.ts` (`tenant_id` , `client_name` , `subdomain`). 1.1.2. **Insert one row** into this table: `{ tenant_id: 'tnt_revenueparty_default', client_name: 'Revenue Party (Default)', subdomain: 'www' }` . **User Story 1.2: Make All Data Tenant-Aware** **As a** Developer, **I want to** add a `tenant_id` column to all *new* tables and *existing* client-facing tables **so that** all data is associated with a tenant. **AC:** 1.2.1. Add a `tenant_id` column to all new tables (`campaigns` , `campaign_zones` , etc.).

1.2.2. Add a `tenant_id` column to all relevant *existing* tables (`blogPosts`, `leadCaptures`, `assessmentResponses`, `widgetConfig`, `testimonials`, etc.).

1.2.3. This column **must** be set to `default('tnt_revenueparty_default')` to tag all existing and new data to our default tenant.

Foundation: Logic

User Story 1.3: Implement "Stub" Middleware

As a Developer, **I want to** create a simple Express middleware that hard-codes the `tenant_id` for every request **so that** all our API queries function correctly.

AC:

1.3.1. Add a new middleware to `server/index.ts` *before* the API routes.

1.3.2. This middleware **must** perform only one function: `req.tenantId = 'tnt_revenueparty_default'; next();`

User Story 1.4: Enforce "Conscious" Querying

As a Developer, **I want to** ensure all *new* database queries for the "Campaigns" Engine are tenant-aware.

AC:

1.4.1. All new Drizzle queries **must** be written with a `WHERE` clause that filters by `req.tenantId` (which is being provided by the "Stub" Middleware).

1.4.2. **Correct:** `db.select().from(campaigns).where(eq(campaigns.tenantId, req.tenantId))`.

Part 2: The "Campaigns" Engine (MVP)

Goal: To build the "Campaigns" wizard, the JSON-powered "Widget Factory," and the frontend rendering engine.

Epic 2.1: The "Campaigns" Admin UI & Schema

- **User Story 2.1.1: Create "Campaigns" Admin UI**
  - **As an:** Admin
  - **I want to:** Have a new "Campaigns" section in the Admin Panel (e.g., `/admin/campaigns`) to create, edit, and manage all my widgets.
  - **So that:** I have a single, unified workflow.
  - **AC:**
    - (AC 2.1.1.1) Create new admin routes for "Campaigns" (List, New, Edit) using the existing admin UI patterns.
    - (AC 2.1.1.2) Create a new `campaigns` table in the Drizzle schema, ensuring it includes a `tenant_id` column per `AC 1.2.2`.
- **User Story 2.1.2: Define Campaign Content ("Section 1: What is this?")**
  - **As an:** Admin
  - **I want to:** Define the name and content type of my new campaign.
  - **AC:**
    - (AC 2.1.2.1) The "New Campaign" form must have a **"Campaign Name"** text field.
    - (AC 2.1.2.2) A **"Content Type"** radio button group must be present:
      - `( ) Smart Widget` (for JSON-built Calculators, Forms, Assessments)
      - `( ) Content Collection` (for existing CMS content like Testimonials, Blogs)
    - (AC 2.1.2.3) A *new* `campaigns` table schema must be created to store all fields defined in this Epic.
- **User Story 2.1.3: Define "Smart Widget" Content (Conditional UI)**
  - **As an:** Admin
  - **I want to:** Use a "Smart Prompt Builder" form to generate a perfect LLM prompt when I select "Smart Widget."
  - **So that:** I can easily get the safe, valid JSON config I need.
  - **AC:**
    - (AC 2.1.3.1) If "Content Type: Smart Widget" is selected (from 2.1.2), a new form section must appear.
    - (AC 2.1.3.2) This section must include a "Widget Type" radio: `( ) Form`, `( ) Calculator`, `( ) Assessment`.
    - (AC 2.1.3.3) *Conditionally* display repeater fields based on the "Widget Type" (e.g., "Inputs" & "Formulas" for Calculator; "Questions" & "Result Pages" for Assessment).
    - (AC 2.1.3.4) A read-only **"Magic Prompt"** text area must update in real-time as I fill out these fields.

    - (AC 2.1.3.5) A "Copy Prompt" button must be provided.
    - (AC 2.1.3.6) A final **"Paste JSON Config"** text area must be present.
    - (AC 2.1.3.7) The save endpoint **must** validate this JSON with Zod. If it fails, return a 400 error: "Sorry, that doesn't work with our schema. Here's the prompt template again to help you fix it."

- **User Story 2.1.4: Define "Content Collection" Content (Conditional UI)**
  - **As an:** Admin
  - **I want to:** Select from my existing, hard-coded CMS content when I select "Content Collection."
  - **So that:** I can easily place my Testimonial Carousel or Video Gallery.
  - **AC:**
    - (AC 2.1.4.1) If "Content Type: Content Collection" is selected (from 2.1.2), a *new* dropdown must appear.
    - (AC 2.1.4.2) The dropdown must be labeled **"Select Collection"** and include hard-coded options: `Testimonial Carousel`, `Video Gallery`, `Blog Feed`. (These will be built in Epic 2.2).
- **User Story 2.1.5: Define Campaign Display ("Section 2: How?")**
  - **As an:** Admin
  - **I want to:** Control the visual appearance and behavior of my campaign.
  - **AC:**
    - (AC 2.1.5.1) A **"Display As"** radio button must be present: `( ) In-Page Section`, `( ) Overlay (Popup)`.
    - (AC 2.1.5.2) A **"Select Theme"** dropdown must be present: `Light (Default)`, `Dark`, `Brand Accent`, `Transparent`.
    - (AC 2.1.5.3) *If "In-Page Section" is selected,* a "Select Size" radio must appear: `( ) Full Width`, `( ) Small (Constrained)`.
    - (AC 2.1.5.4) *If "Overlay (Popup)" is selected,* the following dropdowns must appear:
      - `Placement`: "Pop-up (Center)", "Pop-up (Bottom-Right)", "Pop-off (Right Side)"
      - `Trigger`: "On Page Load (after X seconds)", "On Scroll (after X %)"
      - `Dismissal`: "Static (User clicks 'X')", "Fade Out (after X seconds)"
- **User Story 2.1.6: Define Campaign Targeting ("Section 3: Where?")**
  - **As an:** Admin
  - **I want to:** Control exactly which pages my campaign runs on.
  - **AC:**

- (AC 2.1.6.1) A **"Target Pages"** checklist must be present, listing all static pages (Homepage, /problem) and page types (/blog).
- (AC 2.1.6.2) *If "In-Page Section" is selected,* a **"Select Zone"** dropdown must appear: `[Dropdown: Hero, Section 1, Section 2, ... Section 10]`.

Epic 2.2: The "Widget Factory" (The Frontend Renderers)

- **User Story 2.2.1: Build "Smart Widget" Renderers**
  - **As a:** Developer
  - **I want to:** Build the frontend React components that can read and render the "Smart Widget" JSON config.
  - **AC:**
    - (AC 2.2.1.1) Create new dynamic components: `<DynamicCalculator />` and `<DynamicForm />` (and `<DynamicAssessment />`).
    - (AC 2.2.1.2) These components **must** accept a `jsonConfig` prop. They will parse this JSON to render themselves (e.g., mapping `jsonConfig.inputs` to create form fields).
    - (AC 2.2.1.3) These components **must** also accept a `theme` prop and a `size` prop to control styling, using the existing CVA/Tailwind stack.
- **User Story 2.2.2: Build "Collection" Renderers**
  - **As a:** Developer
  - **I want to:** Build hard-coded React components for our "Content Collections."
  - **So that:** The "Campaigns" engine can render our standard CMS content.
  - **AC:**
    - (AC 2.2.2.1) Create `<TestimonialCarousel />`. This component must fetch all testimonials from the `testimonials` table (for the current `req.tenantId`) and render them.
    - (AC 2.2.2.2) Create `<VideoGallery />`. This component must fetch all videos from the `videoPosts` table (for the `req.tenantId`) and render them in a grid.
    - (AC 2.2.2.3) Create `<BlogFeed />`. This component must fetch the 3 most recent posts from `blogPosts` (for the `req.tenantId`).
    - (AC 2.2.2.4) These components **must** also accept the `theme` and `size` props to control styling.

Epic 2.3: Frontend Placement (Consuming the Campaigns)

- **User Story 2.3.1: Implement "In-Page Section" (Widget Zones)**
  - **As a:** Developer
  - **I want to:** Dynamically render all "In-Page Section" campaigns in their designated zones.

    - **AC:**
        - (AC 2.3.1.1) Modify the main page templates (e.g., `Home.tsx`).
        - (AC 2.3.1.2) The page component must fetch all *active* campaigns for the current page and `req.tenantId`.
        - (AC 2.3.1.3) The page will loop from 1 to 10, rendering a `<WidgetZone />` for each "Section."
        - (AC 2.3.1.4) The `<WidgetZone />` component must find the correct campaign(s) assigned to its zone (e.g., "Section 1").
        - (AC 2.3.1.5) It will then render the correct component (e.g., `<DynamicCalculator />` or `<TestimonialCarousel />`) with the correct `jsonConfig`, `theme`, and `size` props.
- **User Story 2.3.2: Implement "Overlay" (Popup) Engine**
    - **As a:** Developer
    - **I want to:** Build a global, site-wide engine to handle all "Overlay (Popup)" campaigns.
    - **AC:**
        - (AC 2.3.2.1) Build a new, single `<PopupEngine />` component and ensure it runs on every page (e.g., in `App.tsx`).
        - (AC 2.3.2.2) `<PopupEngine />` must fetch all *active* "Overlay" campaigns for the current URL and `req.tenantId`.
        - (AC 2.3.2.3) It must use `useEffect`, `setTimeout`, and scroll listeners (or `IntersectionObserver`) to manage all "Trigger" and "Dismissal" logic (timers, scroll percent, etc.).
- (AC 2.3.2.4) It **must** use **Framer Motion** (already in stack) to handle all "pop-up" and "fade-out" animations.
- (AC 2.3.2.5) It must render the correct "Smart Widget" (e.g., `<DynamicForm />`) inside the popup with the correct `jsonConfig` and `theme`.

Part 3: Future Vision (Long-Term)

This MVP enables the following future epics. **No work is required on these items for the MVP.**

- **Activate Full Multi-Tenancy:** Replace the "Stub" Middleware with a dynamic, subdomain-aware middleware.
- **Build Client-Facing Portal:** Create a new `client_users` table and a client-facing, permission-restricted version of the Admin Panel.
- **Direct LLM API Integration:** Add a "Generate with AI" button that calls the Gemini API directly from the server, auto-populating the JSON config.
- **Advanced Analytics:** Build the "Lead Journey" dashboard (based on the `events` table) to provide "Facebook-grade" surveillance and prove ROI.
- **Social Media Portal:** Create a new "Content Collection" type for embedding social media feeds