Treasure Coast AI – Simple + Powerful Client-Ready Platform Blueprint

This document defines, in detail, how to turn the existing Treasure Coast AI multi-tenant chatbot platform
into a simple-to-manage, powerful, premium product that you can confidently sell for top dollar.

Goals:
• You can control EVERYTHING from one place (the Control Center).
• You do NOT need to dig into code or JSON files for normal client changes.
• The platform looks and feels like a polished SaaS (landing page, dashboards, demos).
• Subscriptions and client status are easy to manage (Active / Paused / Demo).
• Complex flows (like Faith House crisis/intake) are hidden from regular clients.

1. One Central “Control Center” Page (Super-Admin)

The Control Center is a single page in the super-admin dashboard where you can perform all key actions
without navigating through multiple screens or editing files manually.

From this one page, you must be able to:
• Create a new client from any demo template.
• Edit ANY client’s bot settings (business info, services, FAQs, tone).
• Pause/activate client accounts and bots.
• View & filter logs per client/bot.
• Change lead collection options.
• Toggle crisis mode (Faith House only).
• See high-level analytics (messages, leads, booking requests).

1.1 Control Center Layout (UI Blueprint)

Recommended layout:

Top-Level Route:
• /super-admin/control-center

Page Regions:
1) Header Bar:
• Left: “Treasure Coast AI – Control Center”
• Right: currently logged-in super-admin username and a simple status indicator (e.g. “All Systems Operational”).

2) Left Sidebar (Vertical):
• Section: “Clients”
- Dropdown or searchable list of clients.
- When you select a client, the main panel updates.

• Section: “Templates”
- Short list of demo templates (Restaurant, Barber, Auto, Home, Gym, Faith House).
- Button: “Use Template to Create Client”.
• Section: “System”
- Links: “Settings”, “Logs Overview”, “Stripe Settings” (later).

3) Main Panel (Client-Focused):
When a client is selected, show a tabbed interface:

Tabs:
• Overview
• Bot Settings
• Status & Subscription
• Analytics
• Logs

Overview Tab:
• Top row:
- Business name, clientId, type (restaurant, barber, sober_living, etc.).
- Status badge: ACTIVE / PAUSED / DEMO (clickable toggle for super-admin).
• Middle:
- Quick stats: messages (7d), leads captured, booking requests.
• Bottom:
- List of bots for this client with mini-status.

Bot Settings Tab:
• Select bot (if client has more than one).
• Forms for business info, services, FAQs, integrations, tone (detailed in Section 2).

Status & Subscription Tab:
• Toggle status: ACTIVE / PAUSED / DEMO.
• Show Stripe subscription id (if present).
• Show last payment date (if integrated).
• Button: “Open Stripe Portal” (optional later).

Analytics Tab:
• Cards showing core metrics (messages, leads, booking requests).
• Basic charts (bar/line) for messages over time.

Logs Tab:
• Filter by date range.
• Filter by bot.
• Show list of recent conversation metadata (time, bot, brief preview).
• Optional: button to download raw log file.

4) Right-Side Panel (Context / Actions):
• When viewing a template:
- Show a button: “Create New Client From This Template”.
• When viewing a client:
- Show quick action buttons:
- “Impersonate Client Dashboard” (later)
- “Open Client Demo”

- “Edit Client Details”

1.2 Control Center – Backend Requirements

API Endpoints to support the Control Center:

1) GET /api/super-admin/clients
• Returns a list of all clients with:
- id, name, type, status, createdAt
- basic stats (optional cached or computed: messageCount, leadCount).

2) GET /api/super-admin/clients/:clientId
• Returns full details:
- client info
- list of bots (botId, name, description, type, metadata)
- status, subscription info (if stored)
- summary analytics.

3) PUT /api/super-admin/clients/:clientId/status
• Request body: { status: "active" | "paused" | "demo" }
• Effect: update clients.json/store + affect chat behavior.

4) GET /api/super-admin/bots?clientId=...
• Returns all bots for a given client (for populating Bot Settings tab).

5) GET /api/super-admin/bots/:botId
• Returns full bot config (businessProfile, rules, faqs, etc.).

6) PUT /api/super-admin/bots/:botId
• Accepts updates to bot config from the Bot Settings form.
• Writes changes back to JSON or database.

7) GET /api/super-admin/templates
• Returns list of template bots (e.g. where metadata.isTemplate = true).

8) POST /api/super-admin/clients/from-template
• Request body:
- templateBotId
- clientId (desired)
- clientName
- type
- businessProfile overrides (name, phone, email, website, hours, services)
• Server actions:
- Create new client entry.
- Clone template bot to a new bot assigned to this client.
- Save new bot config file.
- Return new client + bot info.

9) GET /api/super-admin/analytics/:clientId
• Returns aggregated stats: messages (period), leads, bookings.

10) GET /api/super-admin/logs/:clientId
• Returns list or summary of log file names and optionally recent entries.

The Control Center page is effectively just a front-end that talks to these APIs.

2. All Business Info Editable in UI (No File Editing)

Business-specific settings must be editable from a web form, not by editing JSON files directly.

Editable fields per bot:
• Business name
• Business type (restaurant, barber, auto, home_services, gym, sober_living, etc.)
• Phone number
• Email
• Website
• Physical location (city, address or general area)
• Hours (officeHours, visitingHours, open/close times)
• Services (list of strings or structured objects)
• Short description / tagline
• FAQs (question + answer pairs)
• Integrations (booking URL, Facebook page, Instagram, etc.)
• Tone settings (friendly, professional, supportive, high-energy, etc.)

Implementation pattern:
• Use a single “Bot Settings” React page/tab inside the Control Center.
• When a bot is selected:
- Load its config via GET /api/super-admin/bots/:botId.
- Populate form fields with data from businessProfile and faqs.

Form Sections:
1) Basic Info:
- Business name (text)
- Type (select dropdown)
- Phone (text)
- Email (text)
- Website (text)
- Location (text)

2) Hours:
- Different fields for each schedule (or a structured weekly editor if desired).
- Can start simple: a single descriptive text field for hours.

3) Services:
- A repeatable list where you can:
- Add service (name + optional description)
- Remove service
- Reorder (optional)

4) FAQs:
- List of rows with:
- Question (text)
- Answer (textarea)
- Buttons: “Add FAQ”, “Remove”.

5) Integrations:
- Booking URL

- Social links
- Any other third-party links you want to show or use in messages.

6) Tone:
- Simple dropdown or toggle:
- “Friendly & casual”
- “Professional & concise”
- “Supportive & reassuring”
- The backend uses this selection to adjust the systemPrompt text or add a small instruction fragment.

Saving:
• When you hit “Save”:
- Frontend sends PUT /api/super-admin/bots/:botId with updated fields.
- Backend merges changes into existing JSON/DB record.
- Important: validate and sanitize input, ensure JSON is valid.

Result:
• No need to open /bots/*.json to update hours, FAQs, or contact info.
• Every change goes through the Bot Settings UI.

3. Templates for Everything (Demo → Client)

Templates are pre-configured bot definitions designed for common industries.

Required templates:
• Restaurant template
• Barber template
• Auto shop template
• Home services template
• Gym template
• Faith House template (for sober living / recovery homes)

Template Behavior:
• A template is just a normal bot config with extra metadata:
- metadata.isTemplate = true
- metadata.templateCategory = "restaurant" | "barber" | ...

• Each template has:
- A generic but strong systemPrompt.
- Default FAQs common to that industry.
- Generic example services.
- Placeholder businessProfile values that will be overridden when creating a real client.

Creating a Client from a Template (Flow):
1) Super-admin opens Control Center.
2) In Templates section, clicks on “Restaurant Template” (for example).
3) Clicks “Create New Client From This Template”.
4) A modal or full-page form appears requesting:
- New clientId (slug-like, e.g. “blue_harbor_grill”).
- Client name (e.g. “Blue Harbor Grill”).
- Business type (pre-filled: restaurant).
- Required businessProfile fields:
• Phone
• Email
• Website
• Location/city
• Hours
- Optional customization:
• Services (a starter list can be pre-filled from template).
5) On submit:
- Backend:
• Validates clientId uniqueness.
• Creates new client entry.
• Clones the template bot JSON.
• Overrides businessProfile with the new client’s data.
• Assigns botId (e.g. clientId + "_main").
• Saves new bot config under /bots/.
• Links bot to the client in clients.json or database.
6) Control Center refreshes:
- New client appears in the Clients list.
- New bot appears under that client.

Why templates matter:
• Onboarding goes from “manual editing” to “fill a single form”.
• You never rebuild logic for similar businesses.
• You can sell fast: show demo → click “Create client from this” → client is live.

Backend Endpoints:
• GET /api/super-admin/templates
• POST /api/super-admin/clients/from-template (as described earlier)

Template Maintenance:
• If you improve a template (better FAQs, better tone), new clients created from that template benefit automatically.
• Existing client bots remain independent and can be customized further.

4. Client Status Control (Active / Paused / Demo)

Client status is what connects business logic, billing, and bot behavior.

Status values:
• active – client's bots are live and respond normally.
• paused – client's bots are temporarily disabled (non-payment, manual pause).
• demo – bots operate in demo mode (limited usage or labeled as demo).

Where status is stored:
• In clients.json or in the clients table in the database:
- status: "active" | "paused" | "demo"

UI Requirements:
• In Control Center:
- Visible status badge next to each client.
- Simple toggle or dropdown to change status.
• In client dashboards:
- Read-only status indicator, so they know if they're active or paused.
• In demo views:
- If status is demo, show a small "Demo Mode" label somewhere.

Backend Enforcement – Chat Endpoint:
• In POST /api/chat/:clientId/:botId:
1) Load the client by clientId.
2) If client.status === "paused":
- Immediately return a safe response:
"This AI assistant is temporarily unavailable. Please contact the business owner or Treasure Coast AI for support."
3) If client.status === "demo":
- Optionally, limit total messages per session or show a small notice in responses.
4) If client.status === "active":
- Proceed normally.

Backend Enforcement – Dashboard Access:
• When a client logs into their dashboard:
- If status === "paused":
• Show a page that says:
"Your account is currently paused. Please update your subscription or contact support."
• Do not show normal dashboards.
- If status === "demo":
• You may show dashboards but add a "Demo Mode" banner.
- If status === "active":
• Full access.

Connection to Billing:
• Stripe webhook events will update status:
- Payment succeeded → set status = "active".
- Subscription canceled or payment failed → set status = "paused".
• Control Center can also override status manually if needed.

5. Simple but Impressive Analytics

Analytics do not need to be complex. They just need to be clear and useful.

Core metrics to show per client:
• Messages handled this week (or last 7 days).
• Leads captured (based on conversations tagged or forms submitted).
• Booking requests (if the bot is configured to push bookings).

Data Sources:
• Conversation logs (file-based or DB) already exist.
• A leads table / flag or logs where a certain event type is “lead_captured”.
• A booking requests table / flag or logs where event is “booking_request”.

Backend:
• Implement an endpoint:
GET /api/super-admin/analytics/:clientId
GET /api/admin/analytics/:clientId (for the client’s own view)

This endpoint:
- Calculates counts for messages in the last 7/30 days.
- Counts leads and booking events by clientId (and optionally botId).
- Returns a simple JSON payload:
{
"messagesLast7d": 1925,
"leadsLast7d": 42,
"bookingsLast7d": 17,
"messageTrend": [...], // small array for charts
"leadTrend": [...]
}

Frontend:
• In Control Center (super-admin) and in client dashboard:
- Show three stat cards:
• “Messages (7d)”
• “Leads (7d)”
• “Booking Requests (7d)”
- Add a simple trend chart (line or bar) for messages over time.

Perception:
• Even though these analytics are simple, they make the platform look powerful and data-driven.
• When presenting to clients, you can use these stats to show value and justify price.

6. Clean, Unified Branding (Desktop + Mobile)

Design consistency across the entire platform strongly impacts perceived value.

Brand Style:
• Coastal, modern, professional.
• Ocean blue gradients, white surfaces, soft shadows.

Core Visual Tokens:
• Primary colors:
- Deep ocean blue
- Bright aqua/teal accent
- Clean white
- Dark navy for backgrounds (where needed)
• Font:
- Inter, Poppins, or system UI with similar look.
• Cards:
- Rounded corners (16–24px).
- Soft shadows (never harsh pure-black).
• Buttons:
- Primary: gradient blue → teal.
- Secondary: outlined / subtle background.
• Spacing:
- Generous vertical padding.
- Clear separation, no clutter.

Apply the same style to:
• Marketing landing page (public).
• Super-admin dashboard.
• Client admin dashboard.
• Demo/chat pages.
• Login screen.

Technical Implementation:
• Define a centralized theme (in Tailwind config or CSS variables).
• Use the same components (buttons, cards, badges) across all pages.
• Avoid one-off styles that break visual consistency.

Mobile:
• Ensure layouts stack gracefully:
- Single-column on small screens.
- Cards wrap or stack.
• Navigation should collapse into a simple mobile menu or stacked sections.

7. Stripe Subscription (Simple, Effective Version)

You only need a very basic Stripe setup initially, focused on recurring monthly subscriptions.

Assumptions:
• You have a Stripe account.
• You create one or a small set of Products/Prices in Stripe (e.g. “Standard Bot Plan – $297/month”).

Backend Components:
1) Environment Variables:
• STRIPE_SECRET_KEY
• STRIPE_WEBHOOK_SECRET
• FRONTEND_BASE_URL (for redirect after checkout)

2) Endpoint: POST /api/billing/create-checkout-session
• Request body:
- clientId
• Server:
- Looks up client in DB/clients.json.
- Creates a Stripe Checkout Session with:
• mode: subscription
• success_url: FRONTEND_BASE_URL + "/billing/success?clientId=..."
• cancel_url: FRONTEND_BASE_URL + "/billing/canceled?clientId=..."
• line_items: one price (your monthly plan).
- Returns session.url to the frontend.

3) Endpoint: POST /api/billing/webhook
• Verifies webhook signature using STRIPE_WEBHOOK_SECRET.
• Handles relevant events:
- checkout.session.completed:
• Get customer/subscription id from event.
• Store subscriptionId on the client record.
• Set client.status = "active".
- invoice.payment_failed or customer.subscription.deleted:
• Identify client by subscription or customer id.
• Set client.status = "paused".

Frontend Components:
• In Control Center → Status & Subscription tab:
- If no subscriptionId:
• Show button: “Start Subscription”.
• On click: call POST /api/billing/create-checkout-session,
then redirect the browser to session.url.
- If subscriptionId exists:
• Show “Active Subscription” with last payment info (if available).
• Optionally: button to “Manage Billing” (using Stripe customer portal later).

Connection to Status System:
• Billing webhook and manual super-admin actions both write to client.status.
• Chat and dashboard access logic rely on this status.

This is enough to:
• Automatically activate clients after they pay.
• Automatically pause clients when payments fail or are canceled.
• Avoid chasing payments manually.

8. Hide Complex Features From Basic Clients

Faith House (sober living) has special requirements:
• Crisis flows and messaging.
• Intake and appointment-type behaviors.
• Extra safety considerations.

Most normal business clients (barber, restaurant, auto, gym, etc.) do NOT need:
• Crisis-related UI.
• Intake flows.
• Treatment-related wording.

Solution: Feature Gating by Client Type
• Each client has a type:
- "sober_living"
- "restaurant"
- "barber"
- "auto"
- "home_services"
- "gym"
- etc.

• In frontend and backend logic:
- If client.type === "sober_living":
• Show Faith House–style options: crisis toggle, intake flows, etc.
- Else:
• Hide those sections completely.
• Use generic, simple business flows only.

Examples:
1) Admin Dashboard:
• For Faith House:
- Show “Crisis Handling Settings”, “Intake Settings”.
• For a barber:
- Only show “Business Info”, “Bot Settings”, “Analytics”, “Leads”.

2) Chat Bot Behavior:
• All bots follow global safety rules (no medical/legal/crisis advice).
• But only sober_living type bots reference house policies and crisis resources in responses.

3) Templates:
• Faith House template clearly labeled as “Sober Living / Recovery Home”.
• Other templates are normal B2C business templates.

Outcome:
• Platform remains clean and simple for typical clients.
• Complex recovery-specific functionality exists but is isolated and only visible when needed.

9. Implementation Roadmap (Order of Operations)

To avoid overwhelm and keep progress clean, implement changes in this order:

Step 1 – Enforce client.status in chat and dashboard
• Wire status checks into POST /api/chat/:clientId/:botId.
• Block paused clients from normal dashboard access.
• Show ACTIVE / PAUSED / DEMO badges in Control Center.

Step 2 – Build Control Center skeleton
• Create /super-admin/control-center route.
• Left sidebar: clients + templates.
• Main panel: Overview tab with basic client info and status.

Step 3 – Implement Templates → Create Client flow
• Add GET /api/super-admin/templates.
• Add POST /api/super-admin/clients/from-template.
• Add UI modal/form that calls this endpoint.

Step 4 – Build Bot Settings form
• For selected client + bot: edit businessProfile and FAQs.
• Connect form to GET/PUT /api/super-admin/bots/:botId.
• Confirm changes save and reflect in bot behavior.

Step 5 – Simple analytics
• Implement analytics endpoint using logs/DB.
• Show three main stats and a simple chart for messages.

Step 6 – Stripe subscription basics
• Add create-checkout-session endpoint.
• Add webhook endpoint and update client.status.
• Add “Start Subscription” button on client status tab.

Step 7 – Branding unification
• Update admin layouts, dashboards, demos to use the same theme as your landing page.
• Test on desktop and mobile.

Step 8 – Feature gating Faith House
• Add client.type checks for Faith House-only flows.
• Hide those flows for generic clients.

Step 9 – Cleanup & documentation
• Remove unused files.
• Write short docs:
- “How to create a new client”
- “How to update a bot”
- “How billing works”

10. Result: Simple to Run, Powerful to Sell

Once everything in this document is implemented:

• You will be able to manage all clients and bots from a single Control Center page.
• You will never need to manually edit JSON or dig through folders for normal operations.
• You will have a stable, clean multi-tenant SaaS platform that looks and feels premium.
• You will be able to spin up new client bots in minutes using templates.
• You will have subscription and status control wired in.
• You will be ready to confidently charge higher monthly prices because the platform is:
- Easy to use
- Visually professional
- Data-driven
- Clearly structured

This blueprint is designed specifically so you can grow Treasure Coast AI without the system becoming a mess or burning you out on maintenance.