Chatbot

Tasks - Chatbot – Developer Requirements & Architecture Document

SaaS Chatbot Platform – Developer Task Split

Project Overview

This is a **SaaS-based chatbot platform** where businesses can subscribe, customize, and embed a chatbot widget on their website. Each client has **their own schema** for data isolation. The platform supports multi-tenancy, a shared NLP engine, and client-specific dashboards.

🔷 Task 1 – Core Platform & Multi-Tenant Functionality (Build First)

Goal: Make the system **work** with multiple clients, tenant schemas, and a working chatbot.
Payment/plan validation is skipped in this phase.

Features / Responsibilities

1️⃣ Client / Tenant Management

- Add new client manually or via API.
- Create **separate schema** per client using `django-tenants`.
- Assign a **client ID** or subdomain for schema identification.
- Store client metadata (name, email, company info) in public schema.

2️⃣ Chatbot Logic

- Rule-based chatbot widget with predefined options.
- Redirects to WhatsApp or internal/third-party form.
- Limit FAQ responses per client (if needed).
- Mobile responsive and lightweight.
- Embeddable JS snippet generated for each client.

3️⃣ Shared NLP Engine (Schema-Aware)

- Tokenization, intent matching, synonym mapping.

- Fetch per-client FAQs and rules from their schema.
- Suggest closest FAQ or chatbot option.
- Save chat summary or form submission in client schema.

4️⃣ Form Handling

- Built-in form (name, email, phone, message) or third-party link.
- Submissions saved in client schema.
- Optional email notification (can be implemented later).

5️⃣ Analytics (Phase 1)

- Log events:
  - Chats started
  - FAQs clicked
  - Chat redirects (WhatsApp / form)
- Store all analytics per client schema.

6️⃣ API Endpoints (Schema-Aware)

- Fetch FAQs, rules, chatbot settings per client.
- Post chat messages, form submissions, and analytics.
- Middleware or header-based schema switching (`X-Client-ID` or subdomain).

🔹 Task 2 – Monetization Layer & SaaS Features (Build Later)

Goal: Add payment, subscription, and plan validation after the core platform is working.

Features / Responsibilities

1️⃣ Pricing & Subscription

- Define pricing plans (Basic, Pro, Enterprise).
- Set feature limits per plan (e.g., FAQ count, messages, analytics depth).

2️⃣ Payment Gateway Integration

- Razorpay, Stripe, or any other preferred gateway.
- Handle one-time or recurring payments.

3️⃣ Subscription Management

- Validate active subscription before allowing tenant access (optional for Task 1).
- Auto-disable tenants for expired payments (optional).

4️⃣ Billing Dashboard

- Admin view for payments, invoices, receipts.
- Upgrade/downgrade flow for plans.

5️⃣ Integration with Task 1

- Wrap Task 1 APIs to check subscription status before granting access.
- Update tenant metadata in public schema with plan info, expiry, limits.

🔷 Development / Build Order Recommendation

Phase	Task	Priority	Notes
Phase 1	Task 1 – Core Platform	High	Focus on schema creation, chatbot, forms, NLP, and analytics. Skip payments.
Phase 2	Task 2 – Monetization Layer	Medium	Add pricing, payment gateway, subscription validation, billing dashboard.

			Wrap Task 1 APIs.
Phase 3	Integration	High	Connect payment validation with Task 1 for real SaaS usage.

🔹 Notes for Developers

- Use **PostgreSQL** for multi-schema support.
- Use **django-tenants** for schema creation and switching.
- Task 2 depends on Task 1 APIs but can be developed in parallel for payment features.
- Keep **Task 1 modular**: chatbot, NLP, forms, analytics, and schema-aware APIs should be independent of payment logic.
- Embed script for client website:

```
<script src="https://cdn.chatbot.com/widget.js" data-client="tenant_clientID"></script>
```

✅ **Result**

- Task 1 = working multi-tenant chatbot platform (free/trial mode).
- Task 2 = SaaS billing, pricing, and subscription management.
- Clean separation ensures fast MVP first, monetization later.
