Project Proposal: API Specification Repository System

Prepared by: Emilios Richards
Date: 26/03/2025

Overview

Following our conversation, this document outlines a flexible, scalable system to collect, manage, and interact with API specification data provided by your clients. This system is meant to support your campaign workflows by enabling quick access to API specs. This system will be able to scale as the business grows.

The goal is to start simple, provide value early, and evolve over time. All features and technical depth can be customised to fit your needs.

Core Architecture Decision

The first choice to make is how we store and organise the specs:

Option A: Lean File/Folder-Based Structure

- Organise specs in folders by client and campaign
- Versioning handled via dated folders (e.g. /Client/Campaign/2025-03-26/)
- Skype message criteria entered manually and stored as .txt or .json
- Simple Streamlit dashboard to browse, filter, and download specs
- Store specs in formats like JSON, YAML, PDF

Pros:

- Simple and quick to build
- No complex setup required
- Cloud accessible (via Streamlit + GitHub or Drive)

Cons:

- Limited filtering/searching
- Folder versioning
- Harder to scale as campaigns grow

Option B: Dynamic System with PostgreSQL

- All specs and campaign data stored in a PostgreSQL database (cloud-hosted)
- Full metadata support: client, campaign, payout model, format, version
- Dashboard fetches live data and supports powerful filtering/search
- Easily add/update specs, criteria, and campaign details
- Built-in structure for future automation, tracking, and team usage

Pros:

- Fully searchable and structured
- Team-ready, scalable
- Ideal foundation for future features (auth, notifications, etc.)

Cons:

- More setup time and ongoing complexity

Modular Feature Tiers

The following features can be layered onto either Option A or Option B depending on needs and priorities:

Feature	Basic	Advanced
Dashboard UI	Streamlit (simple)	Flask or React (custom design, UX)
Criteria Input	Manual text field	Smart NLP helper (message parsing)
Version Control	Folder timestamps	DB version table/Visual tool in UI showing changes
Notifications	None	Email updates on changes
Authentication	None	User logins & user clearance
Collaboration	Shared links	Comments, notes
Scheduling Api Calls	Manual run	Daily cron job (Render, GitHub Actions)

All of these are optional and can be built incrementally.

Recommended Approach:

Start lean, prove value, and evolve.

I recommend we begin with:

- A working scraper + versioned spec storage
- A simple dashboard UI (via Streamlit)
- Manual criteria input

From there, we can expand the system in phases:

- Add PostgreSQL database and switch the dashboard to pull from it
- Build automation for daily scraping
- Add notification triggers
- Build user roles or smarter criteria parsing as needed

This modular path keeps the system functional at every stage.

Payment & Milestones

Payment structure can be milestone-based. This allows flexibility and control over progress. Each milestone will align with a deliverable (eg., scraper, dashboard, DB integration, etc.).

Pricing will depend on the level of functionality and technical scope you choose.

Ongoing Support

I viewed the video discussion you sent via email. I'd also be happy to support this system long-term in a part-time or hourly capacity. I can:

- Help maintain and update campaigns regularly – including the variable adjustments outlined in the video
- Add new features incrementally
- Ensure the system remains stable and helps with your workflow

Let’s Build This Together

This system can become a reliable tool for campaign management. Whether we go lean or dynamic, it will be thoughtfully designed, easy to use, and ready to grow with you.

Let me know which direction we should take this, and I can prepare a milestone breakdown.

Best,
Emilios Richards