

GTM Automation System

✓ Problem Statement: Automate a Use Case with Testsigma's Public APIs or Demo Environment

Build a working automation flow in Testsigma (or mock it up if you're unable to access the tool) that simulates the following GTM problem:

Use Case: "Pre-Sales Custom Demo Builder"

Your task is to automate a table or dashboard that can dynamically create demo test cases based on a prospect's product page. You can use:

- Scraping of key flows (e.g., Signup, Login, Add to Cart, Checkout URLs)
- Generate dummy test cases automatically using AI + Testsigma's test logic
- Build a structure that allows Sales Engineers to auto-generate demos for different domains (e.g., e-commerce, SaaS, Fintech)

Goal: To help Sales Engineers and BDRs at Testsigma instantly generate **domain-specific test cases + personalized demo pages** using company input. This enables them to **impress prospects with real use-case demos**, increasing engagement and accelerating deal cycles.

Solution:

Phase 1: Clay – Prospect Enrichment & QA Generation

🎯 Objective

To design an **AI-powered, scalable prospect intelligence system** that auto-enriches raw lead data, extracts contextual company information, and generates tailored QA test cases for sales demos—laying the foundation for a repeatable GTM motion.

🔧 Tools Used

- **CSV (Manual Seed)** – Initial test batch of raw leads

- **Clay** – For enrichment, scraping, and AI-generated test cases
 - **Clearbit (via Clay)** – For logos and enrichment
 - **ClayAgent (Custom GPT Agent)** – To auto-generate company-specific QA use cases
 - **Airtable** – For structured, upsertable record storage
-



Step 1: Ingestion of Raw Prospect Data

- Imported a CSV file into Clay with 9 test prospect records.
- Raw data fields included:
 - Company Name
 - Company's Website URL
 - Company Size
 - Industry

Purpose: To simulate a real-world cold lead list for testing outbound demo personalization.



Step 2: Company Enrichment using Clay AI

Clay's built-in AI features were configured to enrich each company by:

-  **Finding LinkedIn URLs** using the company name (via Clay's AI lookup).
-  **Extracting Company Logos** using **Clearbit** integration on the LinkedIn URL.
-  **Scraping Website Metadata** from the company URL:
 - Website Title
 - Meta Keywords
 - Meta Description

Purpose: Gather structured and unstructured data to infer company context.



Step 3: AI-Generated QA Test Cases (Personalization Engine)

ClayAgent was used to build an **AI-driven QA generator**. It leveraged:

- Enriched company metadata (e.g., keywords, size, industry)
- Website description & title

Result: Each company record was automatically enhanced with **personalized QA test case scenarios**, tailored for demo use by the Testsigma sales team.

Example:

For a logistics SaaS company, QA test cases included automated flows around shipment delays, carrier integration bugs, and API performance—*precisely mapped to their industry context*.

Purpose: Empower the sales team with **deeply contextual, instantly usable demo material**.



Step 4: Airtable Integration – Record Upsertion

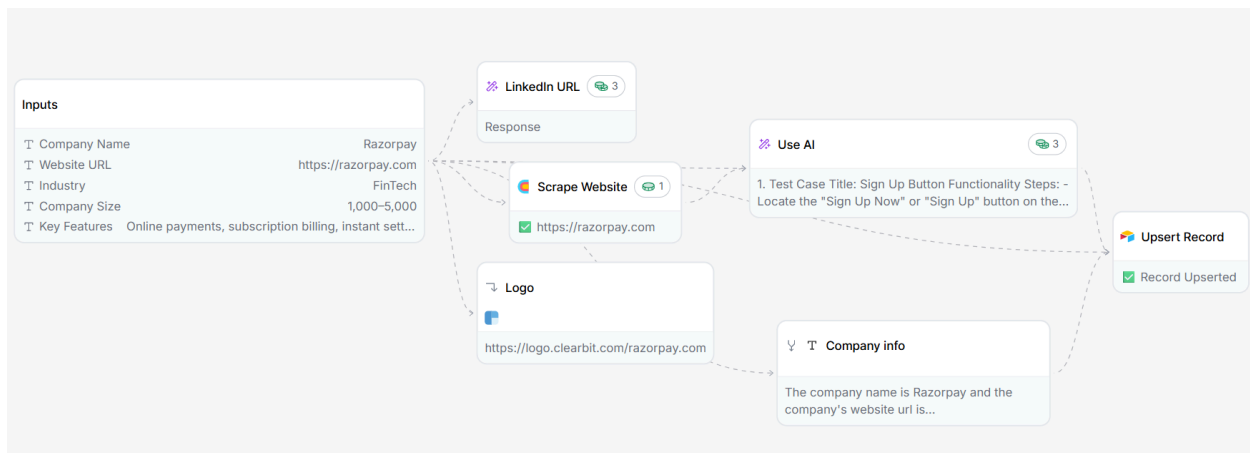
Clay was integrated with **Airtable** using:

- **Personal Access Token**
- **Base ID**
- **Table Name**
- **Merge Field** (e.g., `Company Info`, `Industry`, `Test Cases`)

Configuration:

- Used Clay's **Upsert Integration** logic to ensure records are either inserted or updated.
- Maintained real-time sync between enriched Clay records and the Airtable base.

Purpose: To act as a **single source of truth** that downstream tools (e.g., Make, Supabase, UI layer) can query without data duplication or mismatches.



Integrations made in Clay

Here's the link to the clay table: [Testsigma Prospects Clay Table](#)

Phase 2: Make – Syncing Airtable to Supabase via REST APIs

Objective

To build a **real-time, scalable data bridge** between Airtable (source of enriched QA test cases) and Supabase (backend database powering downstream systems like UI layer, analytics, and GTM agents).

Architecture Summary

Data Flow:

Airtable (enriched test cases) → Make → Supabase (Postgres)

Purpose: Enable Supabase to act as the **central operational backend**, allowing API-first access to company-specific test cases for product demos, dashboards, and further AI workflows.

Tools Used

- **Make.com** – No-code automation layer
 - **Airtable** – Record origin (source of enriched QA test cases)
 - **Supabase** – Scalable Postgres backend
 - **Supabase REST API** – For direct insertion and updates
-

Airtable → Supabase Integration via Make

Integration Design

- Airtable records (test cases) are **monitored via Make's native Airtable module**.
- Only **new or updated records** (unsynced ones) are pulled in.
- The records are **converted to structured JSON** using Make's built-in "Create JSON" module.
- The structured payload is then **pushed to Supabase** using HTTP POST requests through Supabase's RESTful API layer.

Supabase Setup

- A **Testsigma_Prospect** table was created in Supabase with the following schema:
 - **Id** (text)
 - **Company Info** (text)
 - **Industry** (text)
 - **Test Cases** (text)
- REST API access was configured using:
 - **API Endpoint:** https://<project>.supabase.co/rest/v1/Testsigma_Prospect
 - **Authorization Headers:** API Key (Bearer Token & API Key Header)

Automation Logic in Make

- **Trigger:** Watch for new records in Airtable (**Watch Records** module)
- **Transformer:** Convert records into JSON format

- **Action:** POST to Supabase via `Make a HTTP Request` module with:
 - Raw JSON payload:
-

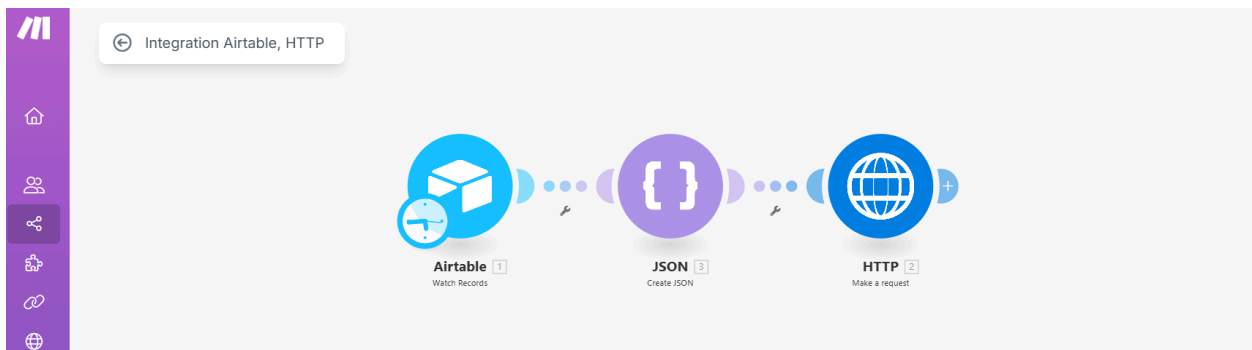
🧠 Why This Architecture?

? Why not directly push from Clay to Supabase?

- **Clay lacks native support** for REST-based upsertion into Supabase.
- Adding a Make layer enables:
 - **Modular control** over filtering, formatting, and routing
 - **Error handling**, retries, and logging
 - A **decoupled, reusable automation layer** between data tools

? Why not push directly from Airtable to Supabase?

- While technically possible, **Make allows conditional flows**, and flexibility to transform Airtable records (e.g., field validation, batching, conditional routing) before final storage.



Integration in Make

JSON Blueprint of the integration in Make:

[blueprint.json](#)

Phase 3: Supabase → Lovable UI | Frontend Dashboard Integration

Objective

To create a **lightweight, real-time frontend dashboard** that displays QA test cases enriched from company signals. This UI is purpose-built to help sales and solutions teams **instantly access demo-ready material**, personalized for each prospect — reducing manual research and accelerating time to demo.

Tools Used

- **Supabase** – Real-time backend database
 - **Lovable (lovable.so)** – No-code AI-native frontend for building contextual dashboards
 - **Prompt engineering** – Used inside Lovable to structure AI output
 - **Supabase Public API** – For fetching test cases directly from Postgres tables
-

Integration Summary

Supabase → Lovable

Lovable was connected to the Supabase table `Testsigma_Prospect` using:

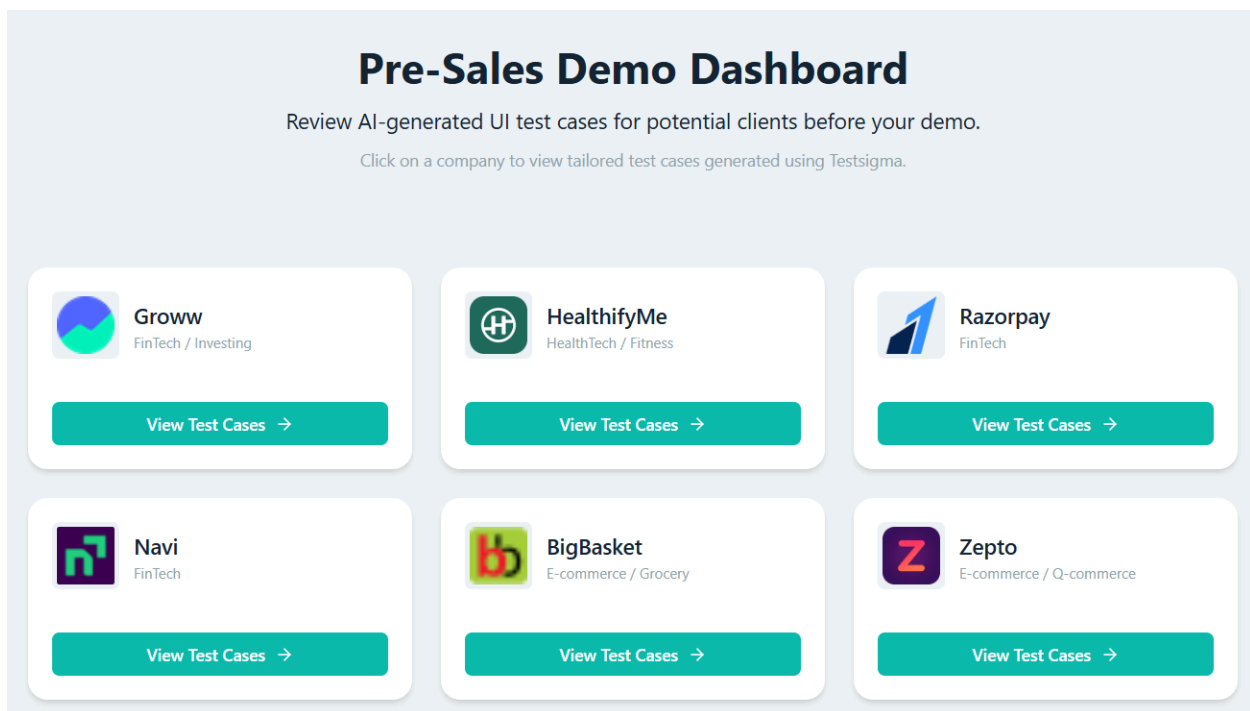
- Supabase REST API Endpoint (`https://<project>.supabase.co/rest/v1/Testsigma_Prospect`)
 - Anonymous API Key for public data fetching
 - Each test case is rendered **in a card-style layout** inside Lovable, grouped by company.
-

Prompt-Powered UI Block

Lovable allows you to plug in **prompt blocks** that consume Supabase data dynamically and generate useful insights.

Final Dashboard Features

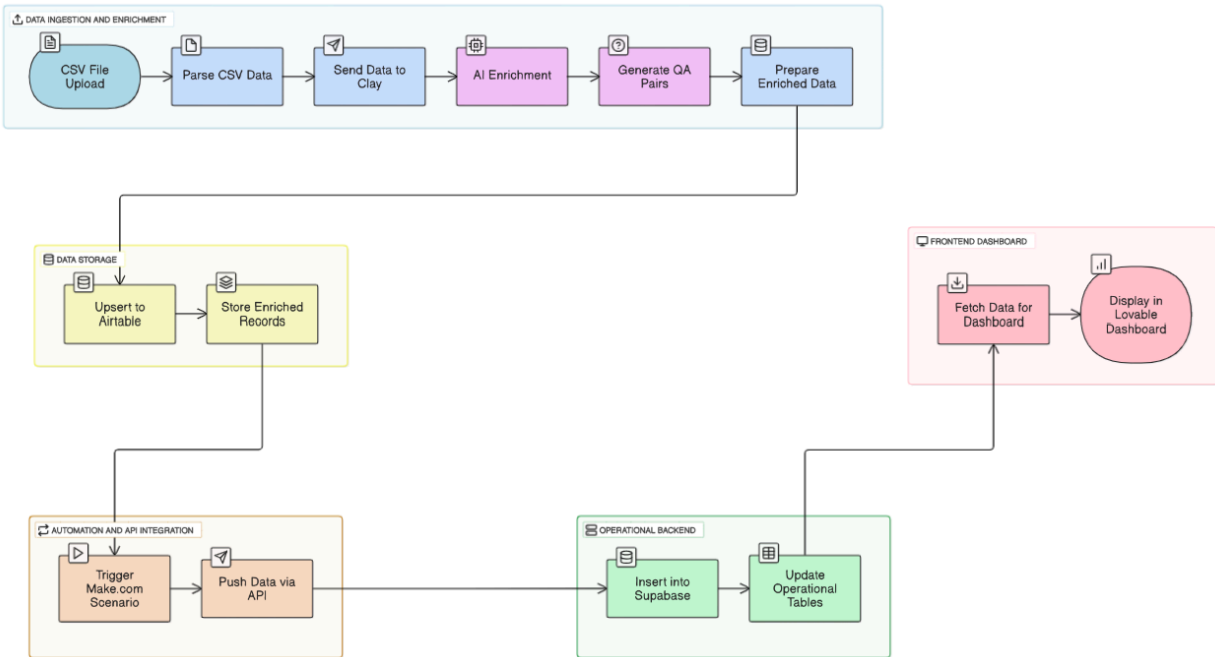
Feature	Description
 Card view by company	Test cases are grouped per prospect
 Live QA summary	Prompt block auto-generates a brief QA testing summary
 No-code backend	All records are managed in Supabase, no dev involvement
 Real-time updates	New Airtable entries reflect in UI within seconds
 Scalable frontend	Can easily support 100s of companies or embed in CRM view



Glimpse of the Pre-sales demo dashboard

Here's the link to the lovable dashboard: [Pre-sales Demo Dashboard](#)

Please find the complete flow diagram of the integration below:



Assignment submitted by - **Aditi Agrawal**