# Autonomous B2B Sales

# Agent with Multi-Step Reasoning

AI-Driven Lead Research, Outreach & Decision Intelligence Platform

## Project Overview:

The AI-Powered B2B Lead Generation & Outreach Automation System is an intelligent, end-to-end platform designed to automate the complete B2B outreach lifecycle from company discovery and lead research to personalized email generation, reply monitoring, follow-ups, and meeting scheduling. Built for sales teams, founders, recruiters, growth marketers, and business development professionals, the system replaces manual lead research and outreach activities with a scalable, AI-driven workflow that improves efficiency and consistency.

Traditional B2B outreach involves time-consuming and subjective tasks such as company analysis, decision-maker identification, email drafting, response tracking, and meeting coordination. This platform addresses these challenges by transforming unstructured web data and email interactions into structured, actionable intelligence. Powered by LangGraph, Large Language Models (LLMs), FastAPI, SMTP/IMAP automation, Google Calendar integration, and a React-based frontend, the system simulates the workflow of an experienced sales professional. By combining AI reasoning with human-in-the-loop approvals and real-time monitoring, it delivers a transparent, controlled, and repeatable outreach process that accelerates pipeline generation while maintaining quality and compliance.

## Scenario 1: Sales Outreach & Lead Generation

Sales teams and business development professionals often rely on manual processes for identifying companies, researching websites, finding decision-makers, drafting personalized emails, and tracking responses. These activities are time-consuming, repetitive, and highly dependent on individual judgment, resulting in inconsistent outreach quality and slower pipeline creation.

The AI-Powered B2B Lead Generation & Outreach Automation System simplifies this workflow by starting from a simple business query such as *"SaaS companies in fintech."* The platform automatically discovers companies, analyses websites, validates contact emails, and evaluates leads using Ideal Customer Profile (ICP) rules. It then generates AI-powered outreach emails that users can review and approve before sending, monitors replies, triggers follow-ups, and assists in scheduling meetings through Google Calendar—allowing sales teams to focus on high-value conversations while improving efficiency and consistency.

## Scenario 2: Recruitment, Agency Outreach & Business Development

Recruitment agencies, founders, and growth teams frequently conduct large-scale outreach to pitch services, partnerships, or hiring solutions. Managing this process manually becomes

difficult to scale, especially when tracking responses, follow-ups, and meeting coordination across multiple prospects.

Using the AI-powered platform, users can run structured outreach campaigns targeting specific industries or company profiles. The system automates company research, generates tailored outreach emails aligned with organizational offerings, and ensures compliance through human-in-the-loop approvals. As prospects respond, the platform tracks engagement, manages follow-ups, and enables quick meeting scheduling, transforming outreach into a structured, measurable, and repeatable process without increasing manual workload.

## Architecture Overview:

The AI-Powered B2B Lead Generation & Outreach Automation System is built on a modular, multi-layered architecture designed to automate the complete outreach workflow while maintaining scalability and human control. At its core, the system uses LangGraph to model the outreach lifecycle as a stateful graph, where each node represents a distinct stage such as company discovery, research, lead qualification, email generation, sending, monitoring, follow-ups, and meeting scheduling. This graph-based design enables conditional routing, persistent campaign state, and seamless pausing or resuming of campaigns without losing context.

The intelligence layer of the platform is powered by Large Language Models (LLMs) accessed through Groq using LLaMA-based models. These models perform deep website analysis, extract business intent signals, generate personalized outreach emails, and support follow-up logic. Rule-based Ideal Customer Profile (ICP) evaluation is combined with AI-assisted reasoning to ensure lead qualification remains explainable, consistent, and aligned with business goals. This hybrid approach balances automation with reliability and transparency.

The orchestration and intelligence layers are exposed through a FastAPI backend, which acts as the communication bridge between the AI workflow and the user interface. Email engagement is handled through SMTP and IMAP integration for sending emails and tracking replies, while Google Calendar and Google Meet APIs enable automated meeting scheduling. A React-based frontend, supported by WebSockets for real-time updates, allows users to configure campaigns, review AI-generated content, approve actions, and monitor outreach progress through an interactive and responsive interface.

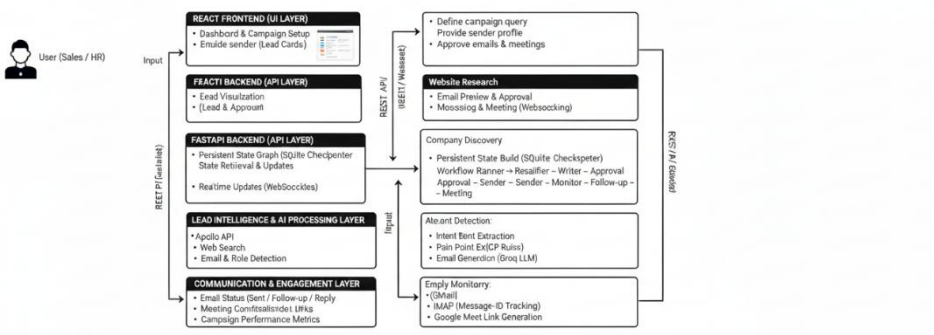



*Figure 1: System architecture diagram*

## Core Technologies:

- **LangGraph (Workflow Orchestration):** LangGraph models the B2B outreach pipeline as a stateful graph, managing node execution, conditional routing, human approvals, and persistent campaign state across long-running workflows.

- **Large Language Models (Groq / LLaMA):** LLMs perform intelligent website analysis, intent detection, email drafting, follow-up generation, and reasoning-based decisions, enabling human-like sales outreach at scale.

- **FastAPI (Backend API Layer):** FastAPI exposes the LangGraph workflow through REST APIs and WebSocket endpoints. It manages campaign lifecycle, approvals, monitoring actions, and frontend–backend communication.

- **React.js (Frontend Application):** React provides a dynamic, component-based user interface for campaign setup, lead visualization, email preview, approvals, and monitoring dashboards.

- **WebSockets (Real-Time Communication):** WebSockets enable real-time updates for campaign status, email approvals, reply detection, and monitoring events without requiring manual refresh.

- **SMTP (Email Sending Engine):** SMTP is used to send AI-generated outreach and follow-up emails securely through configured mail servers.

- **IMAP (Reply Monitoring & Tracking):** IMAP continuously monitors inboxes to detect replies using message IDs, enabling automated response handling and follow-up workflows.

- **Google Calendar API:** Used to create calendar events automatically when meetings are scheduled with prospects.

- **Google Meet Integration:** Generates secure Google Meet links for scheduled meetings, completing the outreach-to-engagement loop.

- **Apollo API (Company Data Source):** Apollo API is used to discover companies based on industry, size, and keywords, providing structured lead data for research and qualification.

- **Web Crawling & Scraping (Requests, BeautifulSoup, DDGS):** These tools extract real-time website content and business information from company websites when API data is unavailable

- **SQLite (Persistent State & Checkpointing):** SQLite stores LangGraph checkpoints, campaign states, and monitoring data, allowing campaigns to pause, resume, and recover reliably.

- **Pydantic (Data Validation & Schemas):** Pydantic enforces structured data models for API requests, responses, and internal state consistency across the system.

- **Axios (Frontend API Client):** Axios manages HTTP communication between the React frontend and FastAPI backend, handling request/response workflows efficiently.

- **Python Utility Libraries:** Libraries such as requests, dns.resolver, and email support networking, domain validation, email parsing, and backend utility functions.

## Component-Wise Architecture:

| Component | Description |
|---|---|
| **React User Interface** | Provides an interactive dashboard for campaign creation, lead visualization, email preview, approvals, and monitoring. Enables users to control outreach workflows and view real-time updates through a responsive web interface. |
| **API Service Layer** | Acts as the communication bridge between the frontend and backend. Handles REST API calls and WebSocket connections for real-time campaign updates, approvals, and monitoring events. |
| **FastAPI Backend** | Exposes the outreach workflow as REST APIs and WebSocket endpoints. Manages campaign lifecycle, user decisions, state updates, and coordination with the LangGraph engine. |
| **LangGraph Orchestration Engine** | Controls the entire outreach lifecycle as a stateful graph. Handles workflow sequencing, conditional routing, human-in-the-loop approvals, and persistent campaign state management. |
| **Company Discovery Module** | Identifies potential companies using Apollo API and web search based on user-defined queries such as industry, location, or company size. |
| **Website Research & Crawling Module** | Crawls company websites to extract business context, services, keywords, and decision-maker signals for lead enrichment and evaluation. |
| **Lead Qualification Engine** | Evaluates researched companies against Ideal Customer Profile (ICP) rules using both rule-based scoring and AI-assisted reasoning to determine lead quality. |
| **AI Email Generation Module** | Generates professional, personalized outreach and follow-up emails using Large Language Models based on company context and sender profile. |

| | |
|---|---|
| **Human Approval Module** | Introduces controlled decision points where users can approve or reject email sending and meeting scheduling, ensuring transparency and compliance. |
| **Email Sending Module (SMTP)** | Sends approved outreach and follow-up emails securely through configured SMTP servers. |
| **Reply Monitoring Module (IMAP)** | Monitors inboxes to detect replies using message identifiers and triggers follow-up or meeting workflows based on response status. |
| **Follow-Up Automation Module** | Automatically generates and sends follow-up emails when no reply is received within defined time intervals. |
| **Meeting Scheduling Module** | Integrates with Google Calendar and Google Meet APIs to schedule meetings and generate conference links when prospects respond positively. |
| **Persistent State & Checkpointing Module** | Stores campaign state, monitoring data, and workflow checkpoints using SQLite, allowing campaigns to pause, resume, and recover reliably. |
| **Notification & Real-Time Update Module** | Uses WebSockets to push live updates such as campaign status, email approvals, replies, and meeting creation to the frontend. |

## Pre-requisites:

1. **Python Environment Setup:** Install Python 3.9 or higher and create a dedicated virtual environment to ensure clean dependency management for backend services, AI orchestration, and automation workflows.

   Official Download: https://www.python.org/downloads/

2. **Backend Dependency Installation:** All backend dependencies required for AI workflow orchestration, API handling, email automation, and persistence must be installed using a virtual environment and requirements.txt.

   **Key libraries include:** LangGraph, LangChain, Groq SDK, FastAPI, Uvicorn, SQLite, BeautifulSoup & Requests, SMTP & IMAP, Google API Client.

   FastAPI Documentation: https://fastapi.tiangolo.com/

   LangGraph Documentation: https://langgraph.langchain.com/

   LangChain Documentation: https://python.langchain.com/

3. **Frontend Environment Setup:** Install Node.js (v18 or higher) to run the React-based frontend application.

   Node.js Official Download: https://nodejs.org/

   React Documentation: https://react.dev/

4. **Large Language Model (LLM) Configuration:** Groq API Key Required for AI-powered email drafting, lead analysis, and follow-up generation.

   Groq Official Website: https://groq.com/

   Groq API Documentation: https://console.groq.com/docs

5. **Lead Data Provider Configuration:** Apollo API Key Required for live company discovery and B2B lead sourcing.

   Apollo Official Website: https://www.apollo.io/

   Apollo API Documentation: https://apolloio.github.io/apollo-api-docs/

6. **Email Service Configuration (SMTP & IMAP):** Required for sending outreach emails and tracking replies automatically.
   - SMTP: Gmail SMTP server
   - IMAP: Gmail IMAP server
   - App password required for secure authentication

   Gmail SMTP Setup: https://support.google.com/mail/answer/7126229

7. **Google Calendar & Meet Integration:** Required for automatic meeting scheduling and Google Meet link generation.
   Steps:
   - Create a project in Google Cloud Console
   - Enable Google Calendar API
   - Generate OAuth credentials (credentials.json)
   - Authenticate and store tokens securely

   Google Cloud Console: https://console.cloud.google.com/

8. **Database Setup:** The system uses SQLite for persistent LangGraph state management.
   - No separate installation required
   - Database file is automatically created at runtime

   SQLite Documentation: https://www.sqlite.org/docs.html

9. **Development & Testing Tools**

   Visual Studio Code: https://code.visualstudio.com/

   PyCharm Community Edition: https://www.jetbrains.com/pycharm/

   Postman (API Testing): https://www.postman.com/

## 10. Browser Requirements

A modern browser is required to access the frontend dashboard and monitoring interface.
Recommended browsers:

- Google Chrome
- Microsoft Edge
- Mozilla Firefox

Google Chrome: https://www.google.com/chrome/

Mozilla Firefox: https://www.mozilla.org/firefox/

## Project Flow:

1. **Environment Setup and Dependency Configuration:**
    - **Activity 1.1:** Obtain Groq / LLaMA API Key for enabling AI reasoning and email generation.
    - **Activity 1.2:** Apollo Account Creation for accessing real-time company and lead data.
    - **Activity 1.3:** Environment Setup & Dependency Installation for configuring backend frontend and API services.
2. **Core AI Workflow & Lead Intelligence Pipeline**
    - **Activity 2.1:** Company Discovery & Research Pipeline using Apollo search and intelligent web crawling.
    - **Activity 2.2:** Lead Research & Enrichment extracting emails roles industry intent and pain points.
    - **Activity 2.3:** Lead Qualification Engine applying ICP rules scoring and qualification logic.
3. **AI-Powered Email Generation & Outreach Automation**
    - **Activity 3.1:** Cold Email Generation creating personalized outreach emails using LLM prompts.
    - **Activity 3.2:** Human-in-the-Loop Approval allowing manual review before sending outreach emails.
    - **Activity 3.3:** Email Sending Engine delivering approved emails securely via SMTP servers.
4. **Monitoring, Follow-ups & Meeting Scheduling**
    - **Activity 4.1:** Reply Monitoring System tracking email replies and updating campaign status.
    - **Activity 4.2:** Automated Follow-up Workflow sending follow-ups based on time and response status.
    - **Activity 4.3:** Meeting Scheduling Automation creating calendar events and Google Meet links.
5. **MILESTONE 5: Frontend Integration & User Experience**
    - **Activity 5.1:** Campaign Setup & Configuration UI collecting campaign query sender and mode details.

- **Activity 5.2:** Campaign Dashboard & Lead Visualization presenting leads scores progress and analytics.
- **Activity 5.3:** Email Preview & Approval Interface displaying drafted emails for user confirmation.

6. **Monitoring Dashboard, Testing & Deployment**

- **Activity 6.1:** Campaign UI Input & Execution Flow Validation validating forms navigation and execution flow.
- **Activity 6.2:** Email Approval, Sending & Follow-up Visibility confirming sent emails follow-ups and mailbox view.
- **Activity 6.3:** Monitoring, Reply Detection & Meeting Scheduling validating reply detection and meeting workflows.
- **Activity 6.4:** Deployment Preparation & Final Validation ensuring stable deployment and production readiness.

## MILESTONE 1: Environment Setup and Dependency Configuration

This milestone establishes the foundational technical environment required for the successful development and execution of the AI-powered B2B Lead Generation & Outreach Automation platform. It focuses on configuring all essential system prerequisites, including backend services, frontend frameworks, external APIs, and secure credential management. Proper setup of programming environments, dependency installation, and project structure ensures that all subsequent development stages operate on a stable, secure, and well-organized foundation.

### Activity 1.1: Obtain Groq / LLaMA API Key

- Visit the official Groq Console: https://console.groq.com/
- Sign in or create a new account to access LLaMA model APIs.

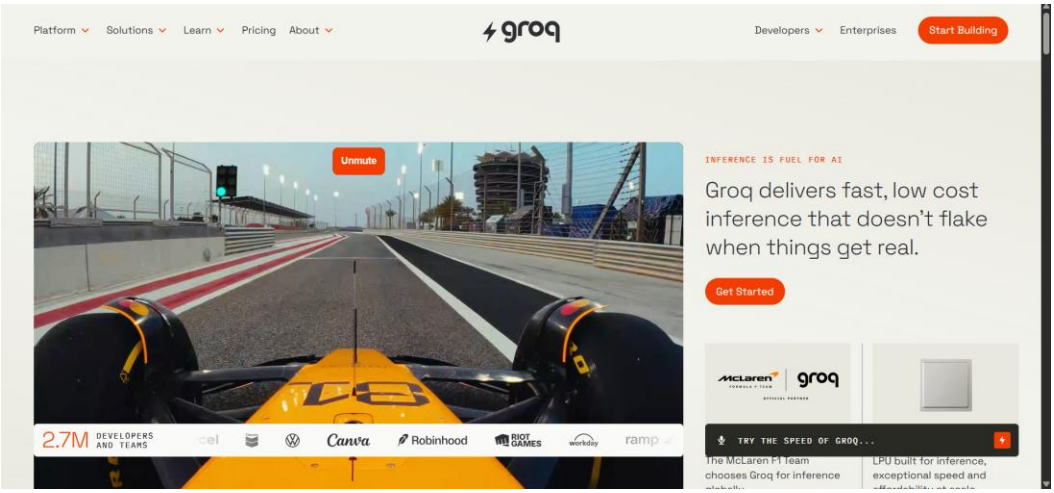



*Figure 2: Groq Home Page*

- Click on Start Building that navigate to dashboard.

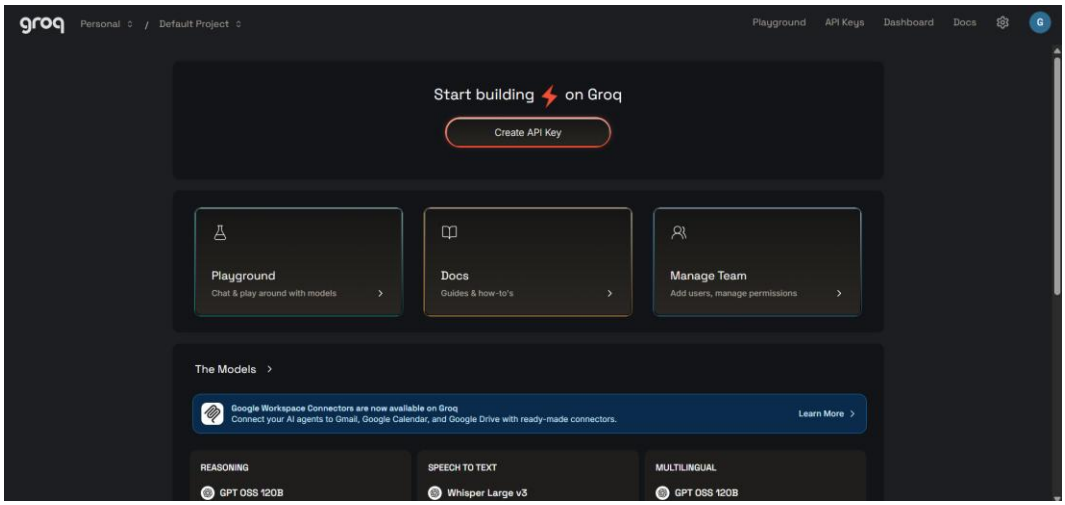



*Figure 3: Groq Dashboard*

- Navigate to the **API Keys** section in the dashboard.
- Click **Create API Key**, then assign a recognizable name such as B2B_Planner.

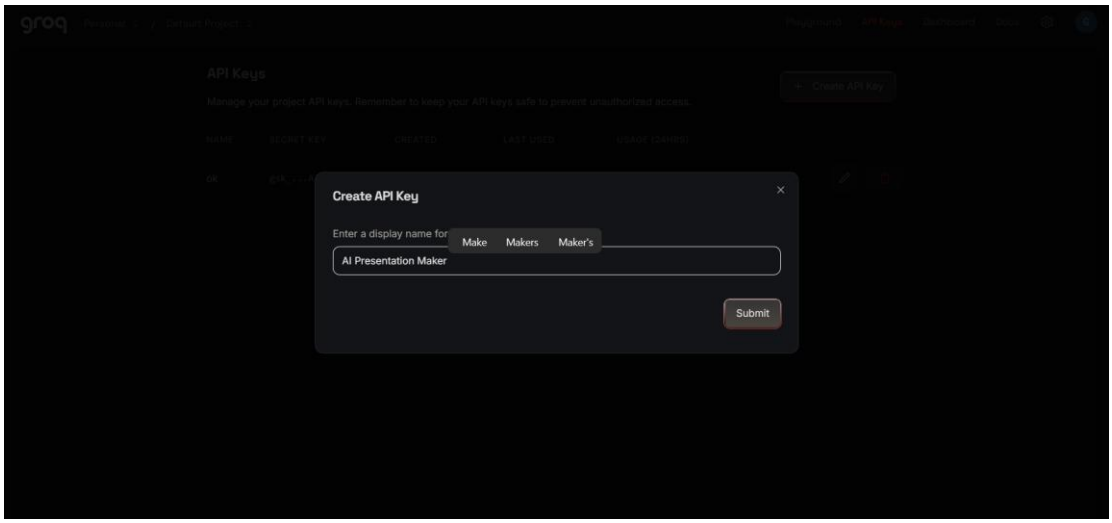



*Figure 4: Creating API Key*

- Copy the generated key and store it securely in your project's .env file.

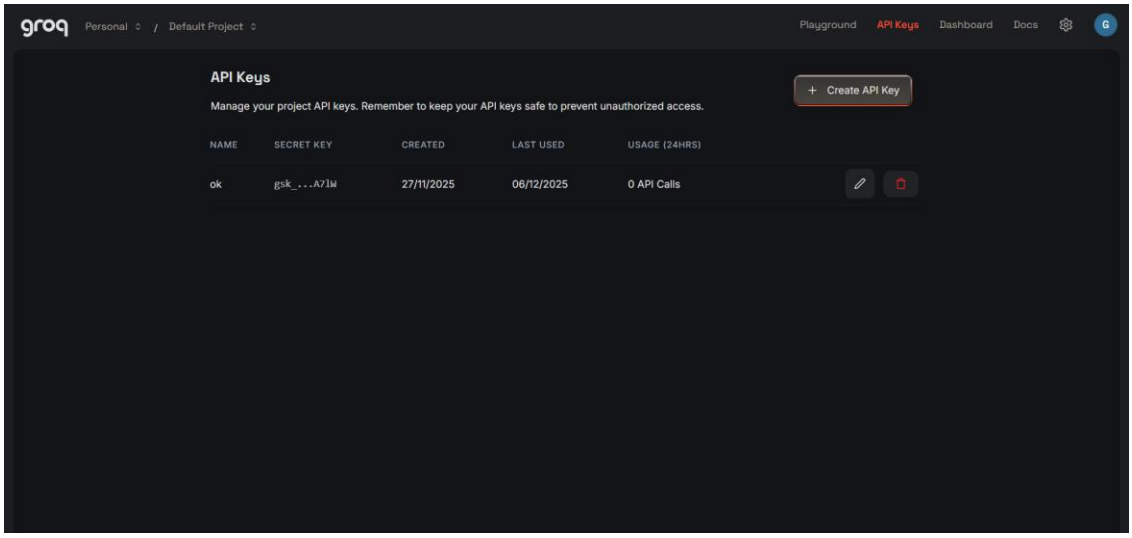



*Figure 5: API Key*

## Activity 1.2: Apollo Account Creation

- Open the official Apollo website: https://www.apollo.io/

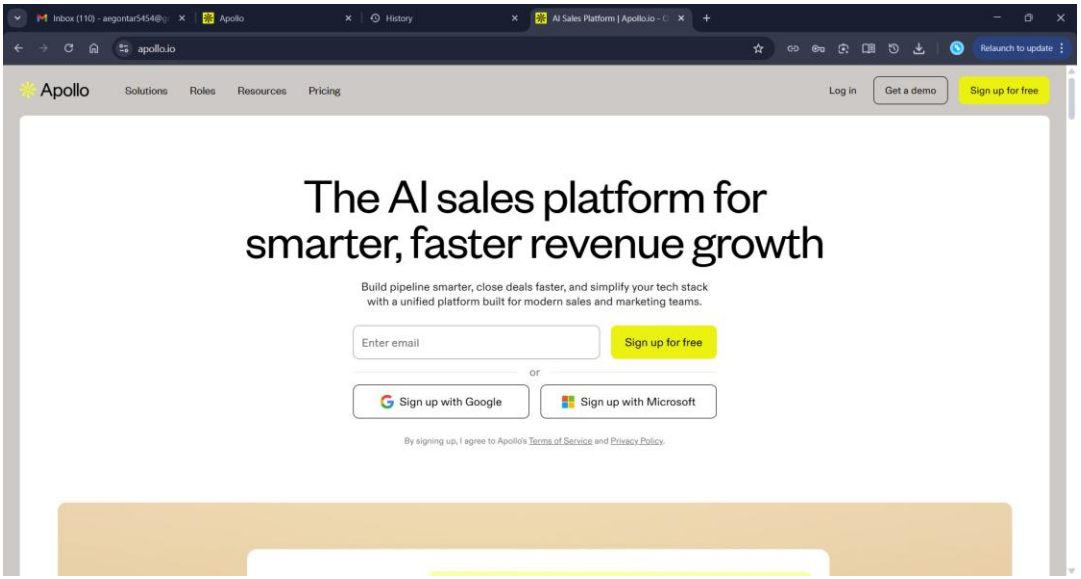



*Figure 6: Apollo home page*

- Click on the "Sign Up" button located at the top-right corner of the homepage.
- Register using:
  – Work email address
  – Google account (recommended)

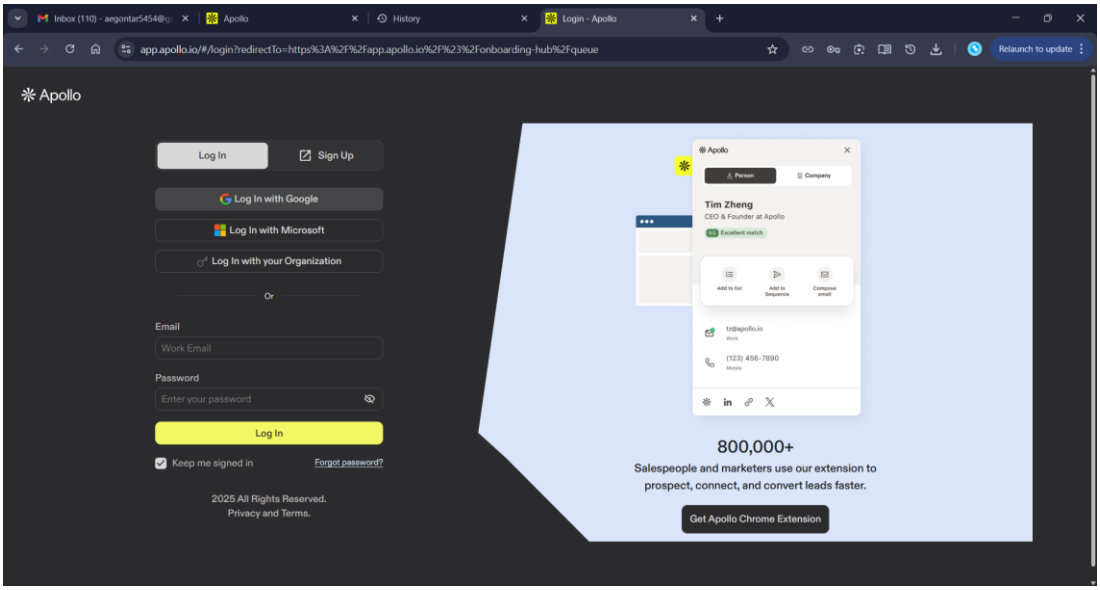



*Figure 7: Sign option*

- After successful registration, you will be redirected to the Apollo Dashboard.

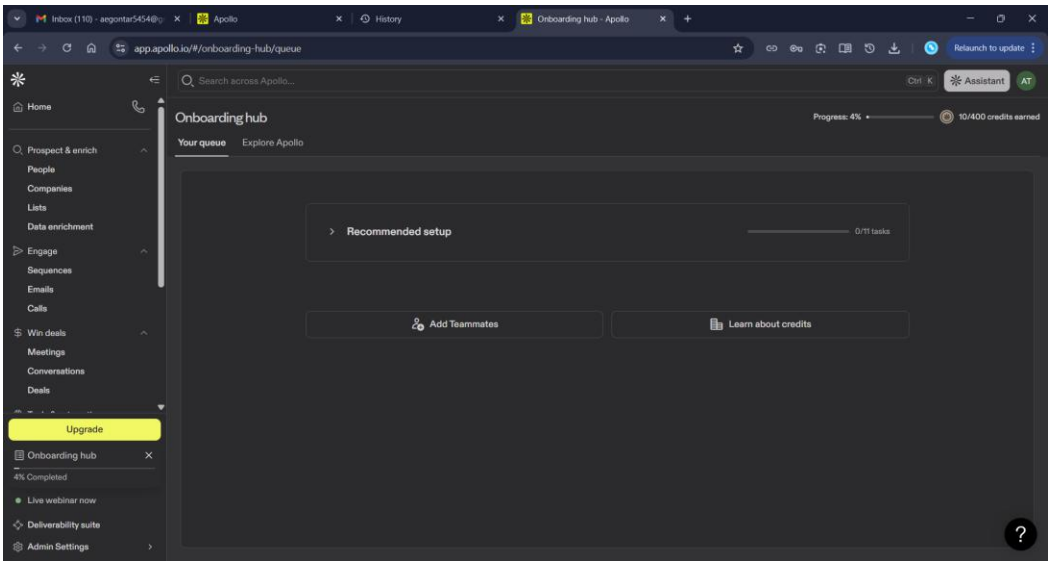



*Figure 8: Apollo Dashboard*

- From the Apollo dashboard, click on your profile icon (top-right corner).
  Navigate to: Settings → Integrations → API

- Locate the API Keys section:

  a) Click on "Create API Key". Provide a meaningful name for the key, for example: B2B_Lead_Generation_Project

  b) Click Generate to create the API key.

  c) Copy the generated API key and store it securely.

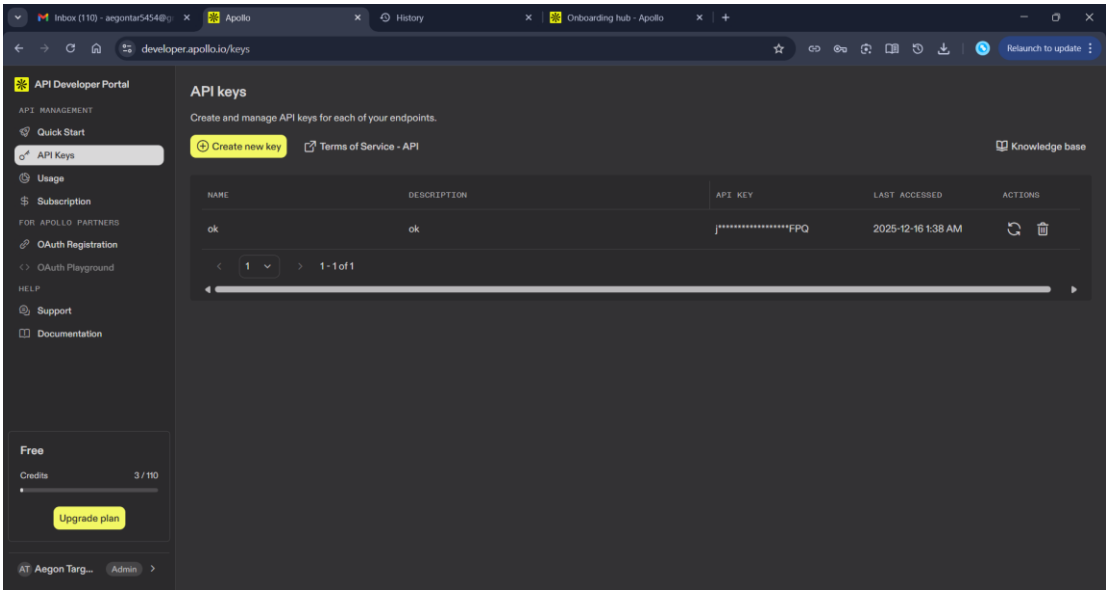



*Figure 9: API Key*

## Activity 1.3: Environment Setup & Dependency Installation

- Create a virtual environment for the project:

```
PS C:\Users\eengi\Desktop\New folder (5)\New folder (2)> python -m venv main
PS C:\Users\eengi\Desktop\New folder (5)\New folder (2)> main/Scripts/activate
```

*Figure 10: Creating & Activating Environment*

- Install project dependencies:

```
PS C:\Users\eengi\Desktop\New folder (5)\New folder (2)> python -m venv main
PS C:\Users\eengi\Desktop\New folder (5)\New folder (2)> main/Scripts/activate
(main) PS C:\Users\eengi\Desktop\New folder (5)\New folder (2)> pip install -r requirements.txt
```

*Figure 11: Installing requirements*

- Making a .env for securing the Gemini API Key as paste the api key here:

```
New folder > backend > .env
1   GROQ_API_KEY=your_groq_api_key_here
2   APOLLO_API_KEY=your_apollo_api_key_here
```

*Figure 12: .env File*

- Set up the project structure for modular development:

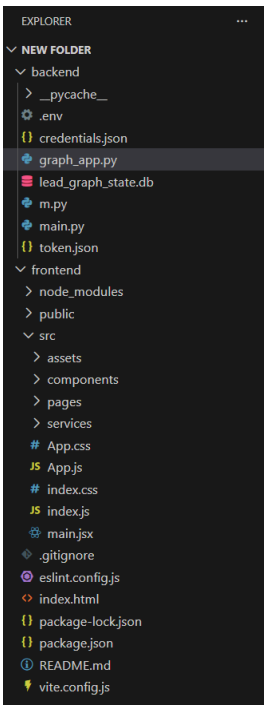


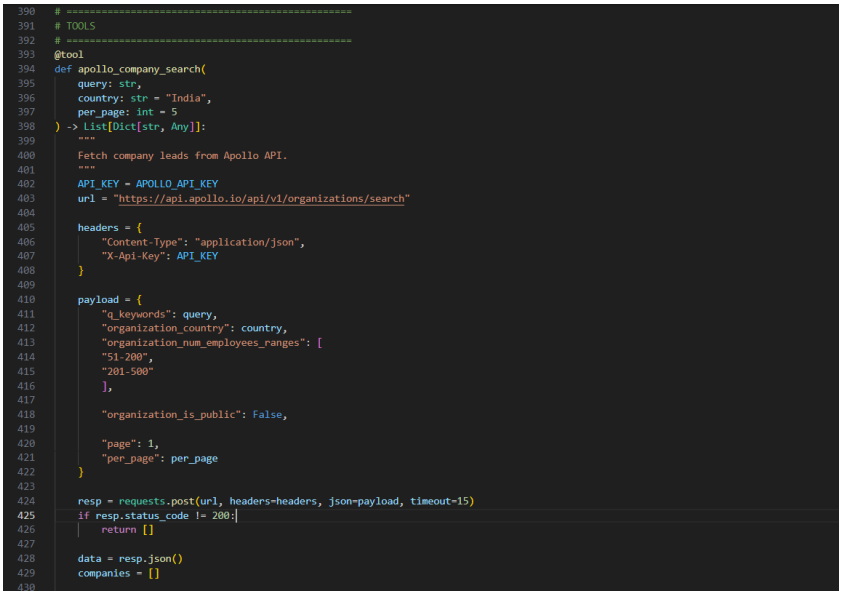

*Figure 13: Folder Structure*

## MILESTONE 2: Core AI Workflow & Lead Intelligence Pipeline

This milestone forms the backbone of the platform by implementing the core AI-driven workflow responsible for lead discovery and intelligence generation. It focuses on transforming raw company data into structured, actionable insights through automated research, enrichment, and qualification. By leveraging a LangGraph-based orchestration system, this stage ensures consistent state management, explainable decision-making, and scalable processing of lead intelligence across campaigns.

### Activity 2.1: Company Discovery & Research Pipeline

- Implement company search using Apollo API and web search fallback.



```
# ================================================
# TOOLS
# ================================================
@tool
def apollo_company_search(
    query: str,
    country: str = "India",
    per_page: int = 5
) -> List[Dict[str, Any]]:
    """
    Fetch company leads from Apollo API.
    """
    API_KEY = APOLLO_API_KEY
    url = "https://api.apollo.io/api/v1/organizations/search"

    headers = {
        "Content-Type": "application/json",
        "X-Api-Key": API_KEY
    }

    payload = {
        "q_keywords": query,
        "organization_country": country,
        "organization_num_employees_ranges": [
        "51-200",
        "201-500"
        ],

        "organization_is_public": False,

        "page": 1,
        "per_page": per_page
    }

    resp = requests.post(url, headers=headers, json=payload, timeout=15)
    if resp.status_code != 200:
        return []

    data = resp.json()
    companies = []
```

*Figure 14: Apollo tool*

```
    resp = requests.post(url, headers=headers, json=payload, timeout=15)
    if resp.status_code != 200:
        return []

    data = resp.json()
    companies = []

    for org in data.get("organizations", []):
        if not org.get("website_url") or not org.get("primary_domain"):
            continue

        companies.append({
            "company_name": org.get("name"),
            "company_website": org.get("website_url"),
            "domain": org.get("primary_domain"),
            "industry": org.get("industry"),
            "estimated_employees": org.get("estimated_num_employees"),
            "keywords": org.get("keywords", []),
            "source": "apollo"
        })

    return companies
```

*Figure 15: Setting the sates*

- Filter non-business domains and invalid websites.

```python
@tool
def web_company_search(query: str) -> List[Dict[str, str]]:
    """
    Search the web for real B2B company websites matching the query.
    Filters out media, blogs, directories, and aggregators.
    """
    companies = []
    with DDGS() as ddgs:
        for r in ddgs.text(query, max_results=15):
            if r.get("href"):
                domain = urlparse(r["href"]).netloc.lower()
                if (
                    domain
                    and not any(bad in domain for bad in BAD_DOMAINS)
                    and is_real_company_site(domain)
                ):
                    parts = domain.split(".")
                    if len(parts) >= 2:
                        company_key = parts[-2]
                    else:
                        company_key = parts[0]

                    companies.append({
                        "company_name": company_key.replace("-", " ").title(),
                        "company_website": f"https://{domain}",
                        "domain": domain
                    })
    return companies[:5]
```

*Figure 16: Web Tool*

- Crawl company websites to extract structured textual content.

```python
@tool
def deep_crawl_site(url: str, max_pages: int = 6) -> str:
    """
    Crawl a company website across internal pages
    and return combined clean text.
    """
    visited: Set[str] = set()
    to_visit = [url]
    base_domain = urlparse(url).netloc
    combined = ""

    while to_visit and len(visited) < max_pages:
        current = to_visit.pop(0)
        if current in visited:
            continue
        visited.add(current)

        try:
            r = requests.get(current, headers=HEADERS, timeout=10)
            soup = BeautifulSoup(r.text, "html.parser")
            for t in soup(["script", "style", "noscript"]):
                t.decompose()
            combined += " " + soup.get_text(" ", strip=True)

            for a in soup.find_all("a", href=True):
                link = urljoin(current, a["href"])
                if urlparse(link).netloc == base_domain:
                    to_visit.append(link)
        except:
            pass

    return combined[:6000]
```

*Figure 17: Deep Search tool*

### Activity 2.2: Lead Research & Enrichment

- Extract validated email addresses from company websites.

```python
@tool
def extract_and_validate_emails(text: str, domain: str) -> List[str]:
    """
    Extract emails from website text and validate domain using MX records.
    Falls back to role-based emails if none found.
    """

    parts = domain.lower().split(".")
    if len(parts) >= 2:
        root_domain = parts[-2] + "." + parts[-1]
    else:
        root_domain = domain.lower()

    found = set(re.findall(EMAIL_REGEX, text))
    emails = []

    try:
        dns.resolver.resolve(root_domain, "MX")
    except:
        return []

    for e in found:
        e = e.lower()
        if e.endswith(f"@{root_domain}") and root_domain not in DISPOSABLE_DOMAINS:
            emails.append(e)

    if not emails:
        emails = [f"{p}@{root_domain}" for p in ROLE_PREFIXES]

    return list(set(emails))

```

*Figure 18: Email extract and validate*

- Detect decision-maker roles such as CEO, CTO, Founder, and Directors.

```python
@tool
def detect_decision_maker_roles(text: str) -> List[str]:
    """
    Detect decision-maker roles mentioned on the company website.
    Returns roles only (no names).
    """
    roles = [
        "ceo", "cto", "cfo",
        "coo", "founder", "co-founder",
        "director", "head organizer", "vp",
        "vice president"
    ]

    text_lower = text.lower()
    found = []

    for role in roles:
        if role in text_lower:
            found.append(role)

    return list(set(found))
```

*Figure 19: Position Finder*

- Identify industry, company size, intent signals, and pain points.

```
    while state["companies"]:
        company = state["companies"].pop(0)

        site_text = deep_crawl_site.invoke(company["company_website"])

        emails = extract_and_validate_emails.invoke({
            "text": site_text,
            "domain": company["domain"]
        })

        decision_maker_roles = detect_decision_maker_roles.invoke(site_text)

        industry = company.get("industry", "unknown")

        company_size = normalize_company_size(
            company.get("estimated_employees")
        )

        intent_signals = map_keywords_to_intent(
            company.get("keywords", [])
        )

        pain_points = []

        prompt = f"""
Extract business facts ONLY from text.
DO NOT invent names for decision_makers.
Try to fill the intent_signals & pain_points field with help of text.
IF intent_signals:
  - Add short phrases ONLY if supported by phrases in the TEXT
    (e.g. "ai platform", "enterprise software", "automation solution"
  - If no clear signals exist, return unknown.
IF pain_points:
  - Add short phrases ONLY if supported by phrases in the TEXT
  - If no pain points are mentioned, return unknown.

Return JSON ONLY.

{{
  "industry": "ai | saas | fintech | ecommerce | other | health-care",
  "company_size": "small | medium | large | unknown",
  "intent_signals": [],
  "pain_points": []
}}

TEXT:
{site_text[:3500]}
"""

        response = model.invoke([HumanMessage(content=prompt)])
        raw = response.content.replace("```", "").replace("json", "").strip()
```

*Figure 20: Extracting information using the LLM*

```
        try:
            enriched = json.loads(raw)
            industry = normalize_industry(
                enriched.get("industry", industry)
            )

            company_size = normalize_company_size_llm(
                enriched.get("company_size", company_size)
            )
            pain_points = enriched.get("pain_points", [])

            if not intent_signals:
                intent_signals = enriched.get("intent_signals", [])

        except:
            pain_points = []


        summary_prompt = f"""
Summarize the following company website text in a clear, descriptive, business-focused way.
Do NOT say "Here is", "Summary:", or similar phrases.
You must return ONLY the summary text.

Rules:
- Use ONLY the provided text
- Do NOT add assumptions
- Do NOT add opinions
- Do NOT invent products or services
- Write 4–6 concise sentences
- Focus on: what the company does, who it serves, and key offerings

TEXT:
{site_text[:3000]}
"""

        summary_response = model.invoke(
            [HumanMessage(content=summary_prompt)]
        )

        website_summary = summary_response.content.strip()


        email_quality = get_email_quality(emails)

        intent_confidence = get_intent_confidence(intent_signals)

        research_confidence = calculate_research_confidence(
            industry=industry,
            company_size=company_size,
            emails=emails,
            people=decision_maker_roles,
            intent_signals=intent_signals
        )

        state["leads"].append({
            "company_name": company["company_name"],
            "company_website": company["company_website"],
            "domain": company["domain"],

            "industry": industry,
            "company_size": company_size,

            "intent_signals": intent_signals,
            "intent_confidence": intent_confidence,
```

*Figure 21: Summary for the extracted text*

```
        state["leads"].append({
            "company_name": company["company_name"],
            "company_website": company["company_website"],
            "domain": company["domain"],

            "industry": industry,
            "company_size": company_size,

            "intent_signals": intent_signals,
            "intent_confidence": intent_confidence,

            "pain_points": pain_points,

            "decision_makers": decision_maker_roles,

            "validated_emails": emails,
            "email_quality": email_quality,

            "website_summary": website_summary,
            "website_text_sample": site_text[:900],

            "research_confidence": research_confidence,

            "source": company.get("source", state.get("source", "web")),
        })

    return state
```

*Figure 22: Storing the values in state*

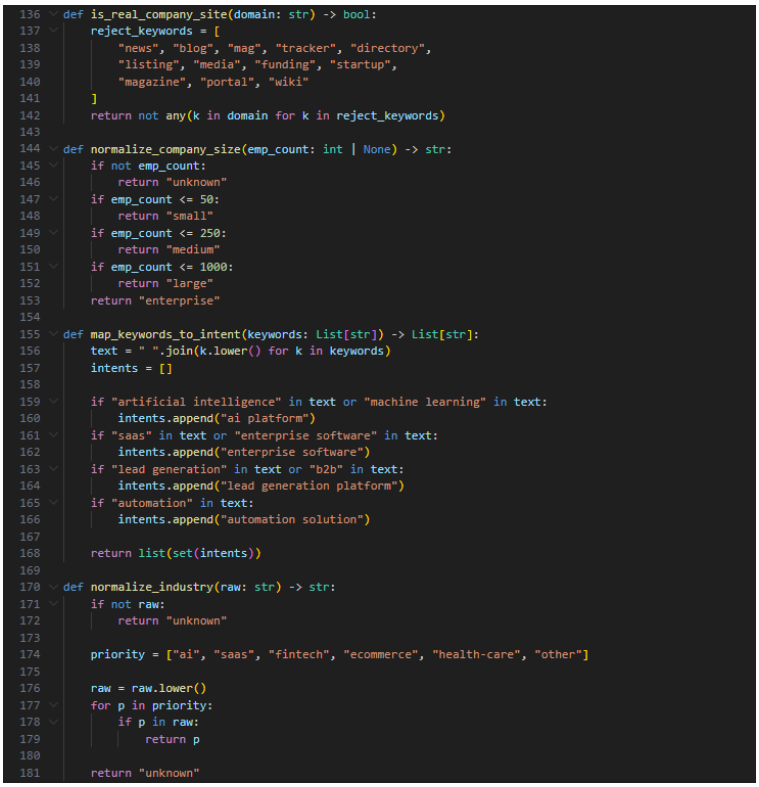
### Activity 2.3: Lead Qualification Engine

- Implement rule-based ICP (Ideal Customer Profile) scoring logic.

```
def calculate_research_confidence(
    industry: str,
    company_size: str,
    emails: List[str],
    people: List[str],
    intent_signals: List[str]
) -> float:
    score = 0.0

    if industry != "unknown":
        score += 0.2
    if company_size != "unknown":
        score += 0.2
    if emails:
        score += 0.2
    if intent_signals:
        score += 0.2
    if people:
        score += 0.2

    return round(score, 2)
```

*Figure 23: ICP Scoring*

- Extracting leads based on industry match, company size, intent confidence, from apollo search.



```python
def is_real_company_site(domain: str) -> bool:
    reject_keywords = [
        "news", "blog", "mag", "tracker", "directory",
        "listing", "media", "funding", "startup",
        "magazine", "portal", "wiki"
    ]
    return not any(k in domain for k in reject_keywords)

def normalize_company_size(emp_count: int | None) -> str:
    if not emp_count:
        return "unknown"
    if emp_count <= 50:
        return "small"
    if emp_count <= 250:
        return "medium"
    if emp_count <= 1000:
        return "large"
    return "enterprise"

def map_keywords_to_intent(keywords: List[str]) -> List[str]:
    text = " ".join(k.lower() for k in keywords)
    intents = []

    if "artificial intelligence" in text or "machine learning" in text:
        intents.append("ai platform")
    if "saas" in text or "enterprise software" in text:
        intents.append("enterprise software")
    if "lead generation" in text or "b2b" in text:
        intents.append("lead generation platform")
    if "automation" in text:
        intents.append("automation solution")

    return list(set(intents))

def normalize_industry(raw: str) -> str:
    if not raw:
        return "unknown"

    priority = ["ai", "saas", "fintech", "ecommerce", "health-care", "other"]

    raw = raw.lower()
    for p in priority:
        if p in raw:
            return p

    return "unknown"
```

*Figure 24: Extracting information from Apollo*

```python
def normalize_company_size_llm(raw: str) -> str:
    if not raw:
        return "unknown"

    priority = ["small", "medium", "large", "unknown"]

    raw = raw.lower()
    for p in priority:
        if p in raw:
            return p

    return "unknown"

def get_email_quality(emails: List[str]) -> str:
    if not emails:
        return "none"

    for e in emails:
        local = e.split("@")[0]
        if local not in ROLE_PREFIXES:
            return "personal"

    return "role_based"


def get_intent_confidence(intent_signals: List[str]) -> str:
    if not intent_signals:
        return "low"
    if len(intent_signals) == 1:
        return "medium"
    return "high"
```

*Figure 25: Extractor*

- Classify leads as qualified or unqualified with explainable scoring.

```python
def qualifier_node(state: LeadState) -> LeadState:
    """
    Rule-based qualification of researched leads.
    """
    if state.get("phase") == "monitor":
        return state
    if state.get("start_from_writer"):
        return state

    qualified_results = []

    for lead in state["leads"]:
        score = 0
        reasons = []

        if lead["industry"] in ICP_CONFIG["industries"]:
            score += 25
            reasons.append("Industry matches ICP")

        if lead["company_size"] in ICP_CONFIG["company_sizes"]:
            score += 20
            reasons.append("Company size matches ICP")

        if lead.get("decision_makers"):
            score += 20
            reasons.append("Decision-maker role found")

        if lead["email_quality"] == "personal":
            score += 15
            reasons.append("Personal email found")
        elif lead["email_quality"] == "role_based":
            score += 10
            reasons.append("Role-based email found")

        if lead["intent_confidence"] == "high":
            score += 20
            reasons.append("High intent detected")
        elif lead["intent_confidence"] == "medium":
            score += 10
            reasons.append("Medium intent detected")

        qualified_results.append({
            "company_name": lead["company_name"],
            "domain": lead["domain"],
            "qualification_score": score,
            "qualified": score >= ICP_CONFIG["min_score"],
            "qualification_reason": reasons
        })

    state["qualification"] = qualified_results

    return state
```

*Figure 26: Qualifiers node*

## MILESTONE 3: AI-Powered Email Generation & Outreach Automation

This milestone introduces intelligent communication capabilities that automate the outreach process while preserving human control. It focuses on generating personalized, context-aware emails using large language models and ensuring that all outbound communication is reviewed and approved by the user. Secure email delivery, traceable message handling, and transparent workflow execution are emphasized to ensure professional and reliable outreach operations.

### Activity 3.1: Cold Email Generation

- Design structured prompts for LLM-based email drafting.

```python

# ================================================
# LLM
# ================================================
model = ChatGroq(
    api_key=GROQ_API_KEY,
    model="llama-3.1-8b-instant",
    temperature=0
)

# ================================================
# Structure LLM
# ================================================
class EmailDraft(BaseModel):
    subject: str = Field(
        description="Short, Professional & Attractive subject line under 60 characters"
    )
    body: str = Field(
        description="Plain text B2B cold email body, 100-200 words, no placeholders"
    )

structured_email_model = model.with_structured_output(EmailDraft)

```

*Figure 27: LLM Setup*

- Generate professional subject lines and email bodies using Groq LLM.

```python
def writer_node(state: LeadState) -> LeadState:
    """
    Writes outreach emails ONLY for qualified leads
    using schema-enforced structured output.
    """
    if state.get("phase") == "monitor":
        return state

    emails_to_send = []

    for q in state["qualification"]:
        if not q["qualified"]:
            continue

        lead = next(
            (l for l in state["leads"] if l["company_name"] == q["company_name"]),
            None
        )
        if not lead:
            continue

        sender = state["sender_profile"]

        prompt = f"""
Write a short, professional B2B cold email.

Rules:
- No emojis
- No hype
- No placeholders like [Your Name] or [Recipient]
- Plain text only
- 100 - 200 words
- Mention company context naturally
- Clear, soft CTA
- Professional tone
- Must have 3-4 paragraph


Sender Information:
- Name: {sender["sender_name"]}
- Role: {sender["sender_role"]}
- Company: {sender["company_name"]}
- What we do: {sender["company_description"]}

Example:
Hi {lead["company_name"]},\n\n

Dear Recipient, I came across Test AI Corp and was impressed by your company's commitment to leveraging AI for business growth.

As a small business owner, I'm sure you're aware of the challenges that come with manual reporting and slow decision-making. Our AI-based analytics solutions can help automate these processes, freeing up your t

I'd be happy to schedule a call to discuss further.
```

*Figure 28: Prompting for LLM*

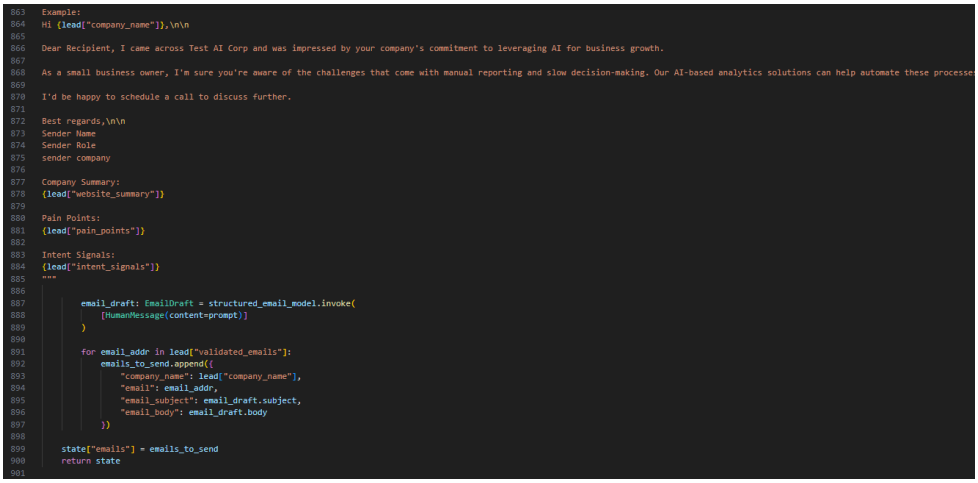



*Figure 29: Updating the state*

- Personalize content using company context, pain points, and sender profile.

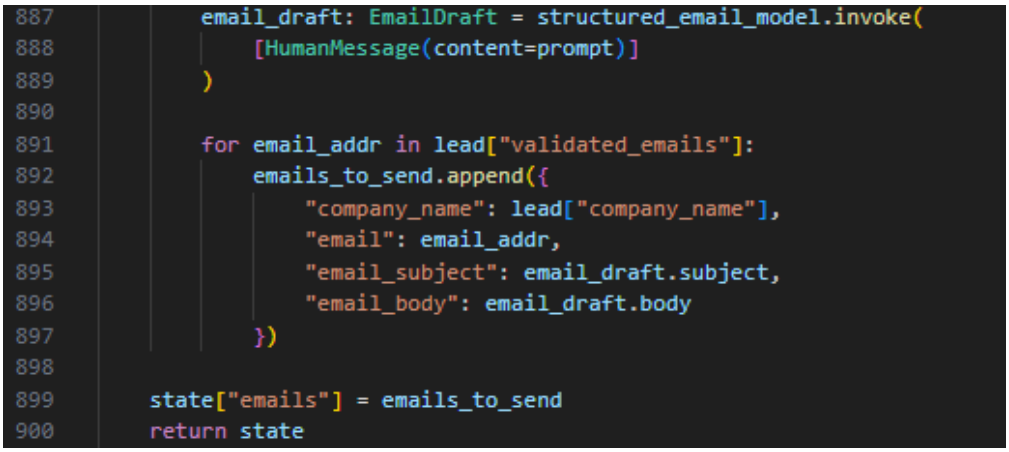



*Figure 30: Updating State*

## Activity 3.2: Human-in-the-Loop Approval

- Implement approval checkpoints for outbound email sending.


*Figure 31: Approval node*

- Pause workflow execution until explicit user approval is received with human decision.

```python
def human_meeting_decision_node(state: LeadState) -> LeadState:
    m = state["active_monitor"]

    if not m:
        return state

    if state["human_decision"].get("send_meeting_email") is not None:
        return state

    return state

def human_meeting_router(state: LeadState):
    if state["human_decision"].get("send_meeting_email") == "yes":
        return "meeting"
    return "monitor"
```

*Figure 32: Meet Approval*

### Activity 3.3: Email Sending Engine

- Implement secure SMTP-based email delivery.

```python
def sender_node(state: LeadState) -> LeadState:
    """
    Sender Agent: Sends emails and logs Message-IDs for monitoring.
    """
    if state.get("phase") == "monitor":
        return state

    sent_logs = []

    with smtplib.SMTP(SMTP_CONFIG["host"], SMTP_CONFIG["port"]) as server:
        server.starttls()
        server.login(
            SMTP_CONFIG["username"],
            SMTP_CONFIG["password"]
        )

        for item in state.get("emails", []):
            try:
                msg = MIMEMultipart()

                msg["From"] = f'{SMTP_CONFIG["from_name"]} <{SMTP_CONFIG["username"]}>'
                msg["To"] = item["email"]
                msg["Subject"] = item["email_subject"]

                message_id = make_msgid()
                msg["Message-ID"] = message_id

                msg.attach(MIMEText(item["email_body"], "plain"))

                server.send_message(msg)

                sent_logs.append({
                    "company_name": item["company_name"],
                    "email": item["email"],
                    "message_id": message_id,
                    "status": "sent",
                    "sent_at": datetime.utcnow().isoformat()
                })

            except Exception as e:
                sent_logs.append({
                    "company_name": item["company_name"],
                    "email": item["email"],
                    "message_id": message_id,
                    "status": "failed",
                    "error": str(e),
                    "sent_at": datetime.utcnow().isoformat()
                })

    state["email_send_logs"] = sent_logs

    now = datetime.now(timezone.utc).isoformat()

    state["monitoring"] = []
```

*Figure 33: SMTP*

```
    state["email_send_logs"] = sent_logs

    now = datetime.now(timezone.utc).isoformat()

    state["monitoring"] = []

    for log in state["email_send_logs"]:
        if log["status"] != "sent":
            continue

        state["monitoring"].append({
            "company_name": log["company_name"],
            "email": log["email"],
            "message_id": log["message_id"],

            "monitor_started_at": now,
            "last_checked_at": None,

            "reply_received": False,
            "meeting_scheduled": False,

            "followup_1_sent": False,
            "followup_2_sent": False,

            "monitor_status": "active"
        })

    state["phase"] = "monitor"
    return state
```

*Figure 34: Sate Update*

- Attach unique message IDs for tracking replies.

```
                msg["From"] = f'{SMTP_CONFIG["from_name"]} <{SMTP_CONFIG["username"]}>'
                msg["To"] = item["email"]
                msg["Subject"] = item["email_subject"]

                message_id = make_msgid()
                msg["Message-ID"] = message_id

                msg.attach(MIMEText(item["email_body"], "plain"))

                server.send_message(msg)

                sent_logs.append({
                    "company_name": item["company_name"],
                    "email": item["email"],
                    "message_id": message_id,
                    "status": "sent",
                    "sent_at": datetime.utcnow().isoformat()
                })
```

*Figure 35: Unique ID*

- Log sent email metadata for monitoring and reporting.


*Figure 36: Log Setup*

## MILESTONE 4: Monitoring, Follow-ups & Meeting Scheduling

This milestone manages post-outreach engagement by continuously tracking responses, triggering follow-up communication, and facilitating meeting coordination. It ensures that email replies are detected accurately, follow-ups are executed ethically and strategically, and meeting requests are handled seamlessly. This stage bridges the gap between initial outreach and meaningful business interaction, maintaining engagement continuity and visibility.

### Activity 4.1: Reply Monitoring System

- Monitor inbox using IMAP to detect replies using message headers.

```python
def check_reply_for_message_id(message_id: str) -> bool:
    """
    Check Gmail inbox for replies to a specific Message-ID
    using IMAP HEADER search (In-Reply-To / References).
    """

    # Gmail IMAP requires Message-ID WITHOUT <>
    search_id = message_id.strip("<>")

    mail = imaplib.IMAP4_SSL(IMAP_CONFIG["host"])
    mail.login(IMAP_CONFIG["username"], IMAP_CONFIG["password"])
    mail.select("inbox")

    # Search by In-Reply-To
    status, data = mail.search(
        None,
        f'HEADER "In-Reply-To" "{search_id}"'
    )

    if status == "OK" and data[0]:
        mail.logout()
        return True

    # Fallback: search by References
    status, data = mail.search(
        None,
        f'HEADER "References" "{search_id}"'
    )

    if status == "OK" and data[0]:
        mail.logout()
        return True

    mail.logout()
    return False
```

*Figure 37: Monitoring the email*

- Update campaign state when replies are received.

```python
def monitor_node(state: LeadState) -> LeadState:
    now = datetime.now(timezone.utc)

    for m in state["monitoring"]:
        if m["monitor_status"] != "active":
            continue

        start = datetime.fromisoformat(m["monitor_started_at"])
        elapsed = (now - start).total_seconds()

        if not m["reply_received"]:
            if check_reply_for_message_id(m["message_id"]):
                m["reply_received"] = True
                m["last_checked_at"] = now.isoformat()
                state["active_monitor"] = m
                return state

        if elapsed >= 60 and not m["followup_1_sent"]:
            state["active_monitor"] = m
            return state

        if elapsed >= 420 and not m["followup_2_sent"]:
            state["active_monitor"] = m
            return state

        if elapsed >= 600:
            m["monitor_status"] = "expired"

    state["active_monitor"] = {}
    return state
```

*Figure 38: Monitor node*

## Activity 4.2: Automated Follow-up Workflow

- Trigger follow-up emails based on time thresholds.

```python
def followup_node(state: LeadState) -> LeadState:
    m = state["active_monitor"]

    lead = next(
        (l for l in state["leads"] if l["company_name"] == m["company_name"]),
        {}
    )

    followup_no = 1 if not m["followup_1_sent"] else 2

    draft = generate_followup_email(lead, followup_no, state["sender_profile"])

    with smtplib.SMTP(SMTP_CONFIG["host"], SMTP_CONFIG["port"]) as server:
        server.starttls()
        server.login(
            SMTP_CONFIG["username"],
            SMTP_CONFIG["password"]
        )

        msg = MIMEMultipart()
        msg["From"] = f'{SMTP_CONFIG["from_name"]} <{SMTP_CONFIG["username"]}>'
        msg["To"] = m["email"]
        msg["Subject"] = draft.subject

        msg.attach(MIMEText(draft.body, "plain"))

        server.send_message(msg)

    if not m["followup_1_sent"]:
        m["followup_1_sent"] = True
    else:
        m["followup_2_sent"] = True

    state["active_monitor"] = {}
    return state
```

*Figure 39: Follow up emails*

- Generate polite, context-aware follow-up messages using LLM.

```python
def generate_followup_email(lead: Dict[str, Any], followup_no: int, sender: Dict[str, Any]) -> EmailDraft:

    prompt = f"""
Write a professional B2B follow-up email.

Rules:
- No emojis
- No hype
- No placeholders like [Your Name]
- Plain text only
- 60–90 words
- Polite and respectful
- Not pushy
- Clear soft CTA
- This is FOLLOW-UP #{followup_no}

Context:
Company: {lead["company_name"]}
Industry: {lead.get("industry")}
Pain Points: {lead.get("pain_points")}
Original Intent: {lead.get("intent_signals")}


Sender Information:
- Name: {sender["sender_name"]}
- Role: {sender["sender_role"]}
- Company: {sender["company_name"]}
- What we do: {sender["company_description"]}

Example:
Hi {lead["company_name"]},

Dear Recipient, I wanted to follow up on my previous email regarding how AI-driven analytics can help simplify reporting and improve decision-making.

I understand you may be busy, so I just wanted to briefly check if this is something worth exploring at this stage. We work with small teams to reduce manual effort and provide clear

If this sounds relevant, I'd be happy to share more or schedule a short call at your convenience.

Best regards,

Sender Name
Sender Role
Sender Company

Tone:
- Follow-up 1 = gentle reminder
- Follow-up 2 = final polite check-in
"""

    return structured_email_model.invoke(
        [HumanMessage(content=prompt)]
    )
```

*Figure 40: Generating cold email*

- Limit follow-ups to predefined stages to avoid spamming.

```
            if not m["reply_received"]:
                if check_reply_for_message_id(m["message_id"]):
                    m["reply_received"] = True
                    m["last_checked_at"] = now.isoformat()
                    state["active_monitor"] = m
                    return state

            if elapsed >= 60 and not m["followup_1_sent"]:
                state["active_monitor"] = m
                return state

            if elapsed >= 420 and not m["followup_2_sent"]:
                state["active_monitor"] = m
                return state

            if elapsed >= 600:
                m["monitor_status"] = "expired"

        state["active_monitor"] = {}
        return state
```

*Figure 41: Follow up conditions*

### Activity 4.3: Meeting Scheduling Automation

- Allow human confirmation for meeting scheduling.

```
def human_meeting_decision_node(state: LeadState) -> LeadState:
    m = state["active_monitor"]

    if not m:
        return state

    if state["human_decision"].get("send_meeting_email") is not None:
        return state

    return state

def human_meeting_router(state: LeadState):
    if state["human_decision"].get("send_meeting_email") == "yes":
        return "meeting"
    return "monitor"
```

*Figure 42: Meet Decision*

- Create Google Calendar events with Google Meet links.

```python
def meeting_node(state: LeadState) -> LeadState:
    m = state["active_monitor"]

    meeting_dt_str = state["human_decision"].get("meeting_datetime")
    if not meeting_dt_str:
        return state

    # Convert "YYYY-MM-DD HH:MM" → datetime
    start_dt = datetime.strptime(meeting_dt_str, "%Y-%m-%d %H:%M")
    end_dt = start_dt + timedelta(minutes=30)

    service = get_calendar_service()

    event = {
        "summary": f"Meeting with {m['company_name']}",
        "description": "Automated meeting created by B2B outreach system",
        "start": {
            "dateTime": start_dt.isoformat(),
            "timeZone": "Asia/Kolkata",
        },
        "end": {
            "dateTime": end_dt.isoformat(),
            "timeZone": "Asia/Kolkata",
        },
        "conferenceData": {
            "createRequest": {
                "requestId": str(uuid.uuid4())
            }
        },
        "attendees": [
            {"email": m["email"]}
        ]
    }

    created_event = service.events().insert(
        calendarId="primary",
        body=event,
        conferenceDataVersion=1,
        sendUpdates="all"
    ).execute()

    meet_link = created_event["conferenceData"]["entryPoints"][0]["uri"]

    m["meet_link"] = meet_link
    m["calendar_event_id"] = created_event["id"]
    m["monitor_status"] = "meeting_created"
    m["meeting_scheduled"] = True

    state["active_monitor"] = {}
    return state
```

*Figure 43: Meet Setup*

- Store meeting metadata in campaign state.

```
1134        meet_link = created_event["conferenceData"]["entryPoints"][0]["uri"]
1135
1136        m["meet_link"] = meet_link
1137        m["calendar_event_id"] = created_event["id"]
1138        m["monitor_status"] = "meeting_created"
1139        m["meeting_scheduled"] = True
1140
1141        state["active_monitor"] = {}
1142        return state
```

*Figure 44: Storing the data*

## MILESTONE 5: Frontend Integration & User Experience

This milestone integrates all backend intelligence into a cohesive, user-friendly frontend interface. It focuses on providing users with intuitive tools for campaign configuration, lead visualization, email review, and progress monitoring. Emphasis is placed on real-time updates, responsive design, and clarity of information presentation, enabling users to interact confidently with complex AI-driven workflows.

### Activity 5.1: Campaign Setup & Configuration UI

- Build campaign creation interface with mode selection.
- Capture sender profile details for email personalization.
- Support resuming existing campaign threads.

New folder > frontend > src > pages > JS CampaignSetup.js > default

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { campaignAPI } from '../services/api';
import './CampaignSetup.css';

const CampaignSetup = () => {
  const navigate = useNavigate();

  // Form state
  const [mode, setMode] = useState('test');
  const [query, setQuery] = useState('');
  const [threadId, setThreadId] = useState('');
  const [senderProfile, setSenderProfile] = useState({
    company_name: '',
    sender_name: '',
    sender_role: '',
    company_description: ''
  });

  // UI state
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const [useExistingThread, setUseExistingThread] = useState(false);

  const handleSenderProfileChange = (e) => {
    const { name, value } = e.target;
    setSenderProfile(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    // Validation
    if (!query.trim()) {
      setError('Please enter a search query');
      return;
    }

    if (!senderProfile.company_name.trim() ||
        !senderProfile.sender_name.trim() ||
        !senderProfile.sender_role.trim() ||
        !senderProfile.company_description.trim()) {
      setError('Please fill in all sender profile fields');
      return;
    }

    setLoading(true);
    setError('');
```

*Figure 45: Campaign Setup*

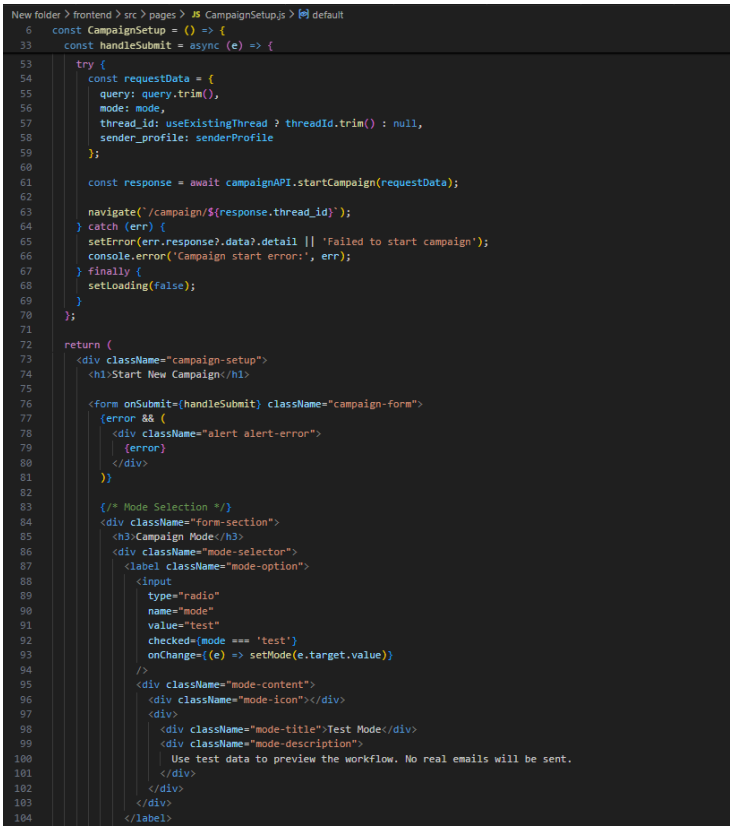



*Figure 46: Campaign Setup*


*Figure 47: Campaign Setup*

```jsx
            <div className="form-group">
              <label className="form-label">What does your company do?</label>
              <textarea
                className="form-control"
                name="company_description"
                value={senderProfile.company_description}
                onChange={handleSenderProfileChange}
                placeholder="Briefly describe your company's products/services (1-2 sentences)"
                rows="3"
                required
              />
              <small className="form-hint">
                This will be used in email templates to personalize outreach.
              </small>
            </div>
          </div>

          {/* Submit */}
          <div className="form-actions">
            <button
              type="submit"
              className="btn btn-primary btn-lg"
              disabled={loading}
            >
              {loading ? (
                <>
                  <div className="spinner spinner-sm"></div>
                  Starting Campaign...
                </>
              ) : (
                'Start Campaign'
              )}
            </button>

            <button
              type="button"
              className="btn btn-secondary"
              onClick={() => navigate('/dashboard')}
              disabled={loading}
            >
              Cancel
            </button>
          </div>
        </form>
      </div>
    );
};

export default CampaignSetup;
```

*Figure 48: Campaign Setup*

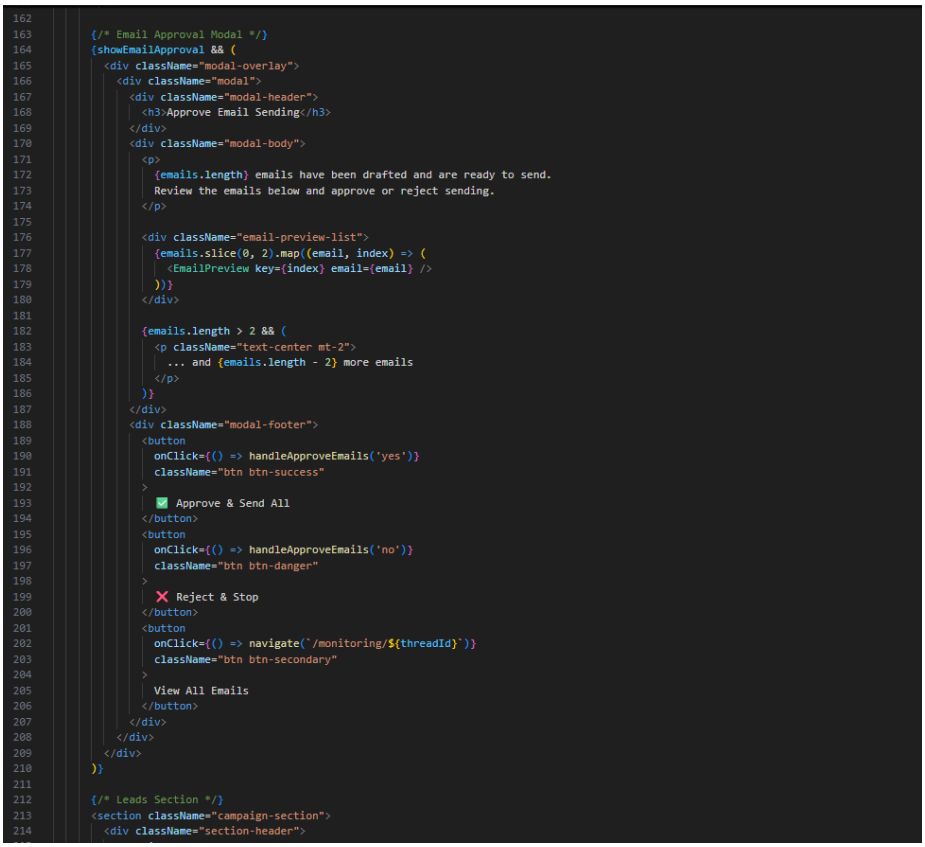
New folder > frontend > src > pages > JS CampaignView.js > CampaignView

```jsx
import React, { useState, useEffect } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
import { campaignAPI, WebSocketService } from '../services/api';
import LeadCard from '../components/LeadCard';
import EmailPreview from '../components/EmailPreview';
import './CampaignView.css';

const CampaignView = () => {
  const { threadId } = useParams();
  const navigate = useNavigate();

  // State
  const [campaign, setCampaign] = useState(null);
  const [leads, setLeads] = useState([]);
  const [emails, setEmails] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [phase, setPhase] = useState('');
  const [showEmailApproval, setShowEmailApproval] = useState(false);
  const [wsService, setWsService] = useState(null);

  useEffect(() => {
    fetchCampaignData();
    setupWebSocket();

    return () => {
      if (wsService) {
        wsService.disconnect();
      }
    };
  }, [threadId]);

  const setupWebSocket = () => {
    const service = new WebSocketService();
    service.connect(threadId);

    service.addListener('message', handleWebSocketMessage);
    service.addListener('connected', () => {
      console.log('Connected to campaign updates');
    });

    setWsService(service);
  };

  const handleWebSocketMessage = (data) => {
    console.log('WebSocket update:', data);

    if (data.type === 'campaign_updated' ||
        data.type === 'campaign_started' ||
        data.type === 'emails_approved' ||
        data.type === 'meeting_scheduled') {
      fetchCampaignData();
    }
  };
```

*Figure 49: Campaign View*

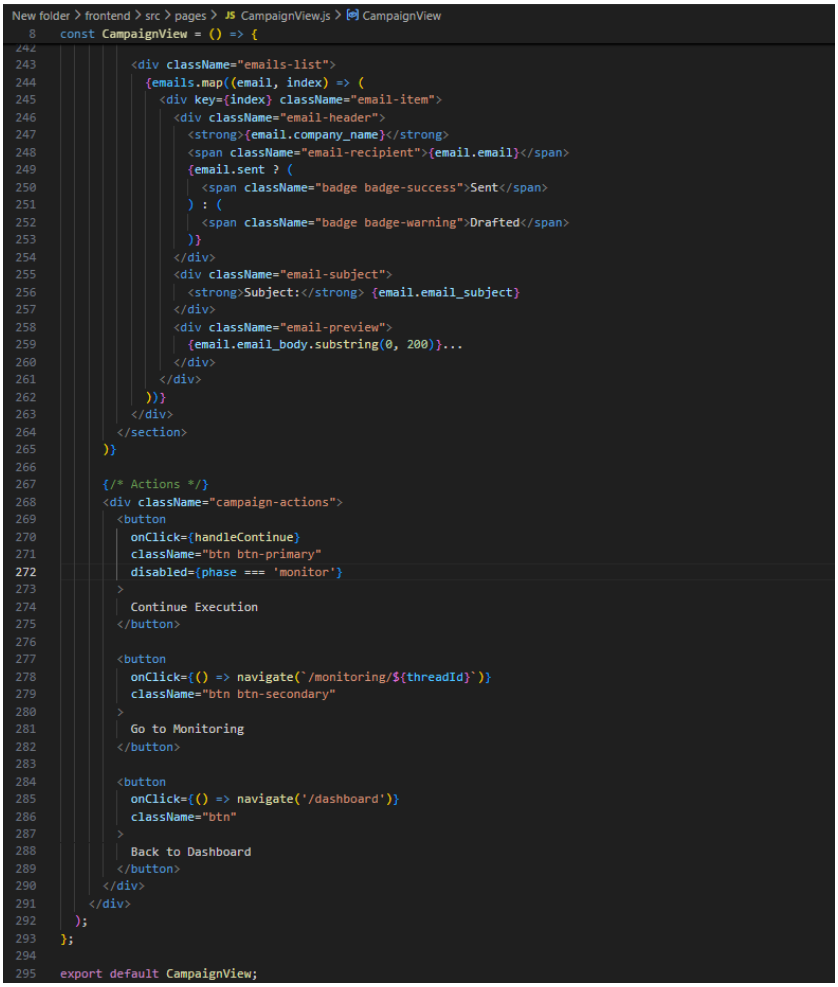
```
New folder > frontend > src > pages > JS CampaignView.js > CampaignView
  8  const CampaignView = () => {
 56    const fetchCampaignData = async () => {
 57      try {
 58        setLoading(true);
 59
 60        // Get campaign status
 61        const status = await campaignAPI.getCampaignStatus(threadId);
 62        setCampaign(status);
 63        setPhase(status.phase || '');
 64
 65        // Get leads
 66        const leadsData = await campaignAPI.getLeads(threadId);
 67        setLeads(leadsData.leads || []);
 68
 69        // Get emails
 70        const emailsData = await campaignAPI.getEmails(threadId);
 71        setEmails(emailsData.emails || []);
 72
 73        // Check if we need to show email approval
 74        if (status.current_state?.human_decision?.send_first_email === undefined &&
 75            emailsData.emails?.length > 0) {
 76          setShowEmailApproval(true);
 77        }
 78
 79      } catch (err) {
 80        setError('Failed to fetch campaign data');
 81        console.error(err);
 82      } finally {
 83        setLoading(false);
 84      }
 85    };
 86
 87    const handleApproveEmails = async (decision) => {
 88      try {
 89        await campaignAPI.approveEmails(threadId, decision);
 90        setShowEmailApproval(false);
 91        if (decision === 'yes') {
 92          setPhase('sending');
 93        }
 94      } catch (err) {
 95        setError('Failed to process email approval');
 96        console.error(err);
 97      }
 98    };
 99
100    const handleContinue = async () => {
101      try {
102        await campaignAPI.continueCampaign(threadId);
103        fetchCampaignData();
104      } catch (err) {
105        console.error('Failed to continue campaign:', err);
106      }
107    };
108
```

*Figure 50: Campaign View*



```
162
163       {/* Email Approval Modal */}
164       {showEmailApproval && (
165         <div className="modal-overlay">
166           <div className="modal">
167             <div className="modal-header">
168               <h3>Approve Email Sending</h3>
169             </div>
170             <div className="modal-body">
171               <p>
172                 {emails.length} emails have been drafted and are ready to send.
173                 Review the emails below and approve or reject sending.
174               </p>
175
176               <div className="email-preview-list">
177                 {emails.slice(0, 2).map((email, index) => (
178                   <EmailPreview key={index} email={email} />
179                 ))}
180               </div>
181
182               {emails.length > 2 && (
183                 <p className="text-center mt-2">
184                   ... and {emails.length - 2} more emails
185                 </p>
186               )}
187             </div>
188             <div className="modal-footer">
189               <button
190                 onClick={() => handleApproveEmails('yes')}
191                 className="btn btn-success"
192               >
193                 ✅ Approve & Send All
194               </button>
195               <button
196                 onClick={() => handleApproveEmails('no')}
197                 className="btn btn-danger"
198               >
199                 ❌ Reject & Stop
200               </button>
201               <button
202                 onClick={() => navigate(`/monitoring/${threadId}`)}
203                 className="btn btn-secondary"
204               >
205                 View All Emails
206               </button>
207             </div>
208           </div>
209         </div>
210       )}
211
212       {/* Leads Section */}
213       <section className="campaign-section">
214         <div className="section-header">
```

*Figure 51: Campaign View*



```
New folder > frontend > src > pages > JS CampaignView.js > CampaignView
  8  const CampaignView = () => {
242
243            <div className="emails-list">
244              {emails.map((email, index) => (
245                <div key={index} className="email-item">
246                  <div className="email-header">
247                    <strong>{email.company_name}</strong>
248                    <span className="email-recipient">{email.email}</span>
249                    {email.sent ? (
250                      <span className="badge badge-success">Sent</span>
251                    ) : (
252                      <span className="badge badge-warning">Drafted</span>
253                    )}
254                  </div>
255                  <div className="email-subject">
256                    <strong>Subject:</strong> {email.email_subject}
257                  </div>
258                  <div className="email-preview">
259                    {email.email_body.substring(0, 200)}...
260                  </div>
261                </div>
262              ))}
263            </div>
264          </section>
265        )}
266
267        {/* Actions */}
268        <div className="campaign-actions">
269          <button
270            onClick={handleContinue}
271            className="btn btn-primary"
272            disabled={phase === 'monitor'}
273          >
274            Continue Execution
275          </button>
276
277          <button
278            onClick={() => navigate(`/monitoring/${threadId}`)}
279            className="btn btn-secondary"
280          >
281            Go to Monitoring
282          </button>
283
284          <button
285            onClick={() => navigate('/dashboard')}
286            className="btn"
287          >
288            Back to Dashboard
289          </button>
290        </div>
291      </div>
292    );
293  };
294
295  export default CampaignView;
```

*Figure 52: Campaign View*

## Activity 5.2: Campaign Dashboard & Lead Visualization

- Display discovered and qualified leads using visual cards.
- Show intent signals, pain points, scores, and summaries.
- Provide real-time campaign statistics and progress indicators.

New folder > frontend > src > pages > JS Dashboard.js > Dashboard > fetchThreads

```jsx
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import { campaignAPI } from '../services/api';
import './Dashboard.css';

const Dashboard = () => {
  const [threads, setThreads] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchThreads();
  }, []);

  const fetchThreads = async () => {
    try {
      setLoading(true);
      const data = await campaignAPI.listThreads();
      setThreads(data.threads || []);
    } catch (err) {
      setError('Failed to fetch campaigns');
      console.error(err);
    } finally {
      setLoading(false);
    }
  };

  const formatDate = (dateString) => {
    if (!dateString) return 'N/A';
    return new Date(dateString).toLocaleDateString();
  };

  return (
    <div className="dashboard">
      {/* Hero Section */}
      <div className="hero-section">
        <h1 className="hero-title">B2B Lead Generator</h1>
        <p className="hero-tagline">AI-powered lead generation and outreach automation</p>
      </div>

      {/* Create Campaign Section */}
      <div className="section create-campaign-section">
        <div className="section-content">
          <h2 className="section-title">Start Your Campaign</h2>
          <p className="section-description">
            Begin your B2B outreach journey with our AI-powered platform. Generate qualified leads,
            craft personalized emails, and automate follow-ups.
          </p>
          <Link to="/campaign/new" className="btn btn-primary btn-large">
            Create New Campaign
          </Link>
        </div>
      </div>
```

*Figure 53: Dashboard*

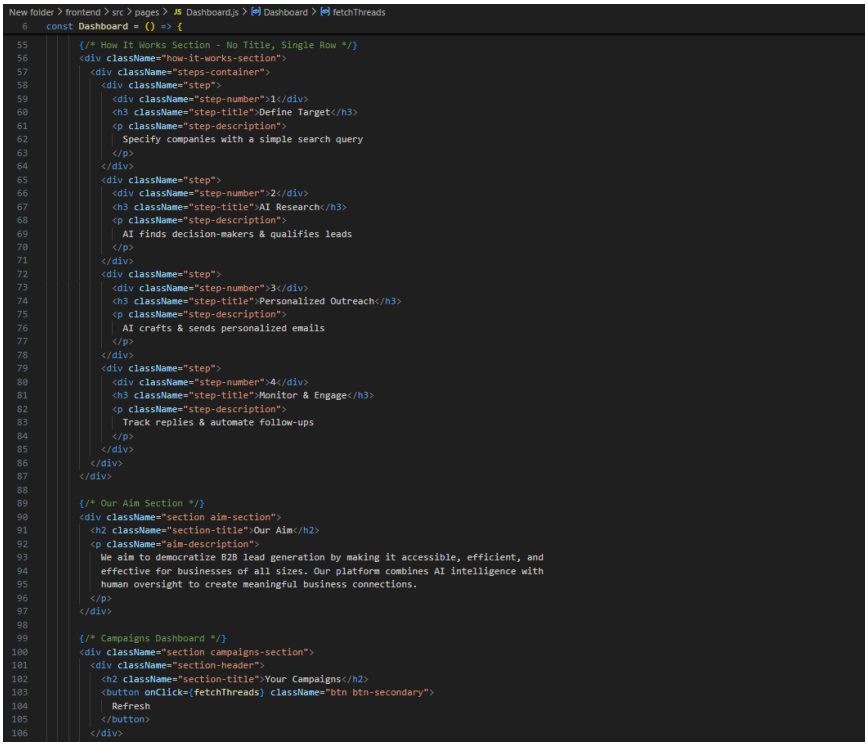



*Figure 54: Dashboard*

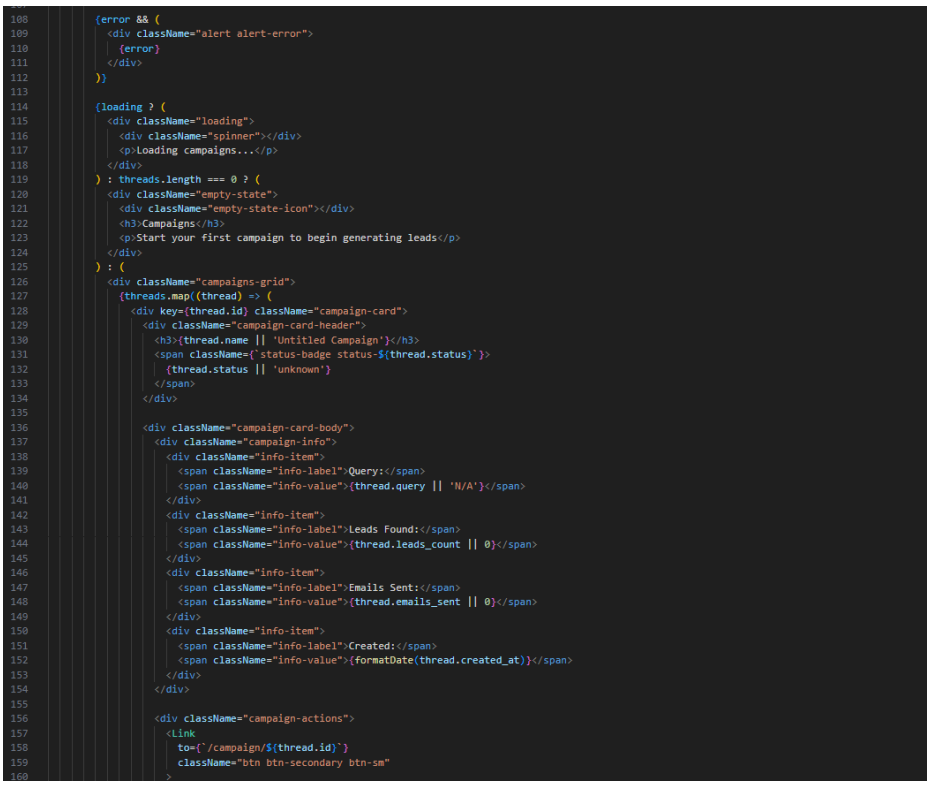
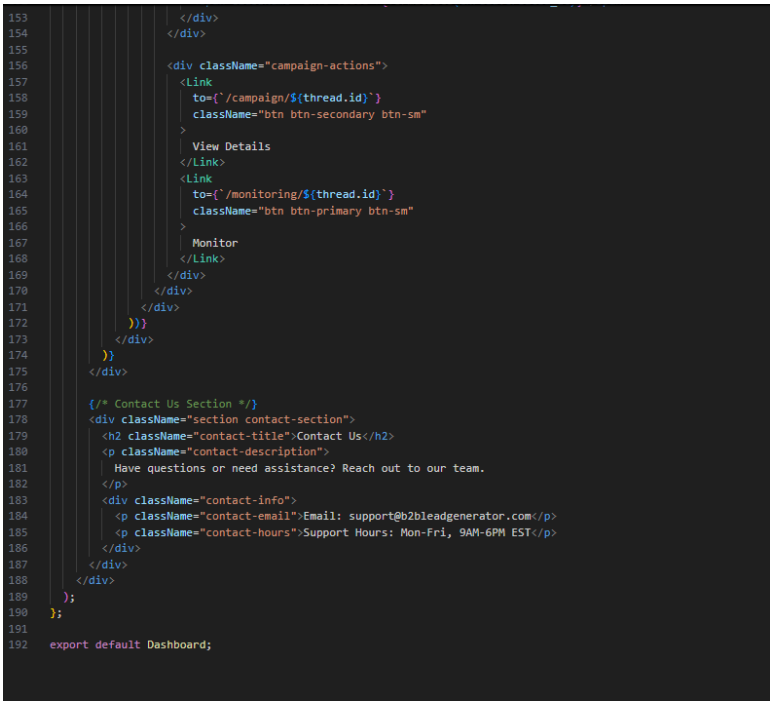
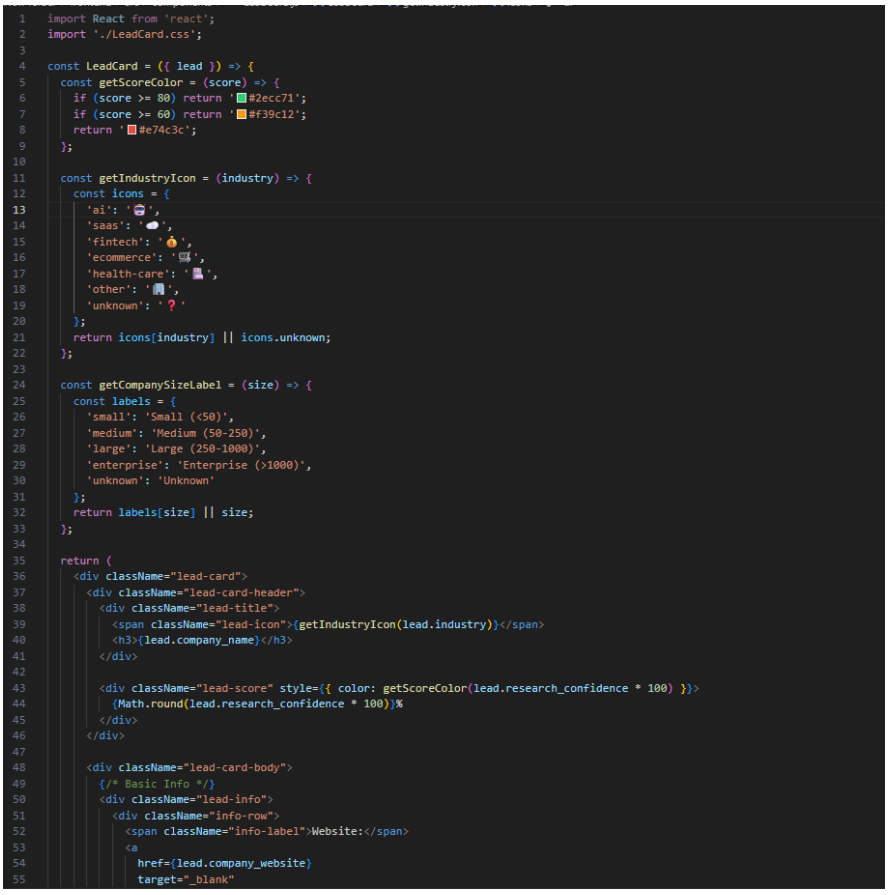



*Figure 55: Dashboard*



```
                    </div>
                  </div>

                  <div className="campaign-actions">
                    <Link
                      to={`/campaign/${thread.id}`}
                      className="btn btn-secondary btn-sm"
                    >
                      View Details
                    </Link>
                    <Link
                      to={`/monitoring/${thread.id}`}
                      className="btn btn-primary btn-sm"
                    >
                      Monitor
                    </Link>
                  </div>
                </div>
              </div>
            ))}
          </div>
        )}
      </div>

      {/* Contact Us Section */}
      <div className="section contact-section">
        <h2 className="contact-title">Contact Us</h2>
        <p className="contact-description">
          Have questions or need assistance? Reach out to our team.
        </p>
        <div className="contact-info">
          <p className="contact-email">Email: support@b2bleadgenerator.com</p>
          <p className="contact-hours">Support Hours: Mon-Fri, 9AM-6PM EST</p>
        </div>
      </div>
    </div>
  );
};

export default Dashboard;
```

*Figure 56: Dashboard*



```
import React from 'react';
import './LeadCard.css';

const LeadCard = ({ lead }) => {
  const getScoreColor = (score) => {
    if (score >= 80) return '#2ecc71';
    if (score >= 60) return '#f39c12';
    return '#e74c3c';
  };

  const getIndustryIcon = (industry) => {
    const icons = {
      'ai': '🤖',
      'saas': '☁️',
      'fintech': '💰',
      'ecommerce': '🛒',
      'health-care': '🏥',
      'other': '🏢',
      'unknown': '❓'
    };
    return icons[industry] || icons.unknown;
  };

  const getCompanySizeLabel = (size) => {
    const labels = {
      'small': 'Small (<50)',
      'medium': 'Medium (50-250)',
      'large': 'Large (250-1000)',
      'enterprise': 'Enterprise (>1000)',
      'unknown': 'Unknown'
    };
    return labels[size] || size;
  };

  return (
    <div className="lead-card">
      <div className="lead-card-header">
        <div className="lead-title">
          <span className="lead-icon">{getIndustryIcon(lead.industry)}</span>
          <h3>{lead.company_name}</h3>
        </div>

        <div className="lead-score" style={{ color: getScoreColor(lead.research_confidence * 100) }}>
          {Math.round(lead.research_confidence * 100)}%
        </div>
      </div>

      <div className="lead-card-body">
        {/* Basic Info */}
        <div className="lead-info">
          <div className="info-row">
            <span className="info-label">Website:</span>
            <a
              href={lead.company_website}
              target="_blank"
```

*Figure 57: Dashboard*

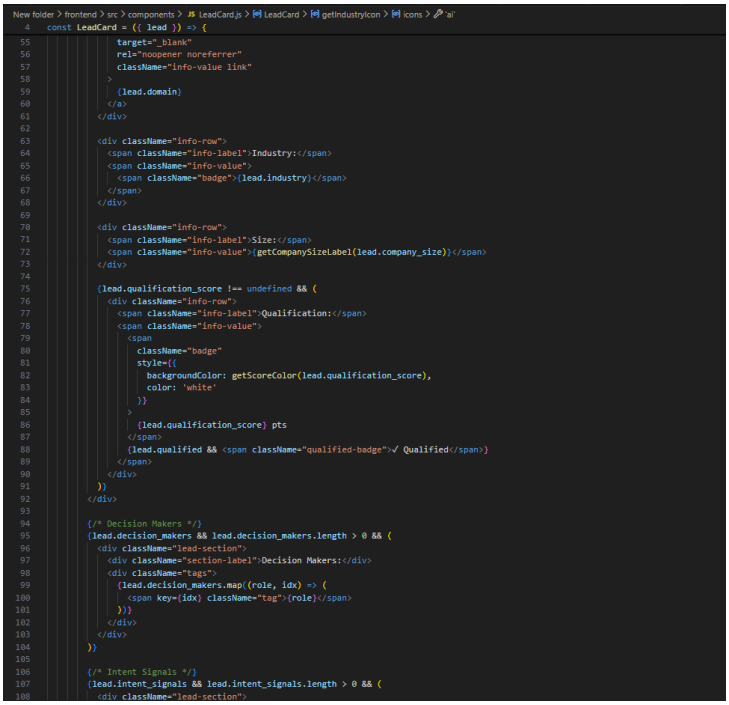



Figure 58: Dashboard


Figure 59: Dashboard

## Activity 5.3: Email Preview & Approval Interface

- Display drafted emails before sending.
- Enable approve/reject actions from the UI.
- Reflect real-time status updates using WebSockets.

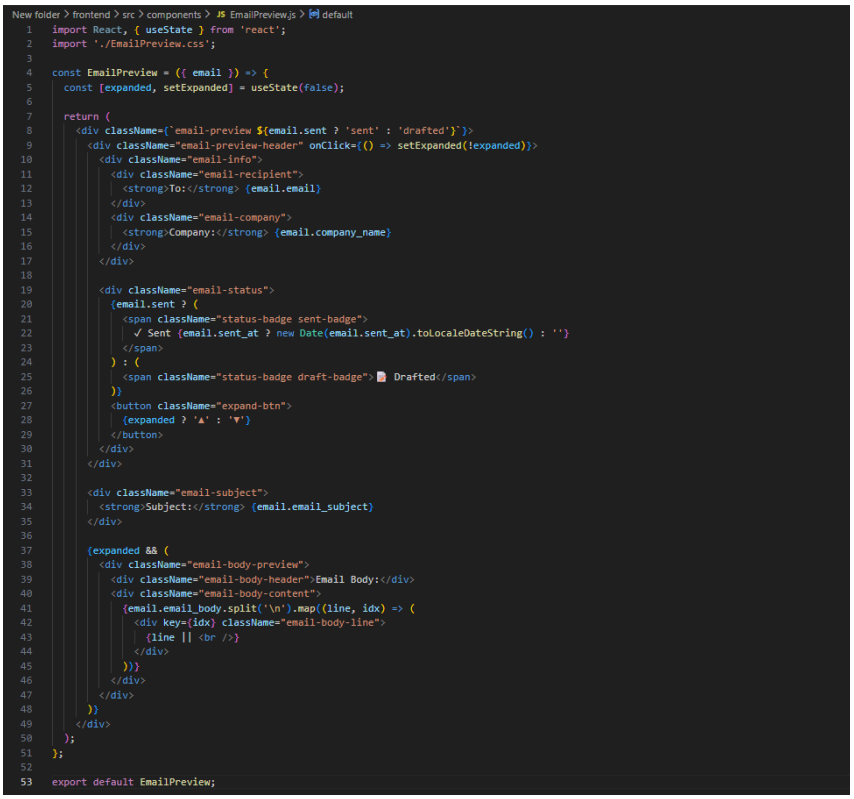



*Figure 60: Email Preview*

## MILESTONE 6: Monitoring Dashboard, Testing & Deployment

This milestone focuses on validating the end-user interaction flow of the AI-powered B2B Lead Generation & Outreach Automation platform. The objective is to ensure that all UI-driven actions such as campaign setup, email approval, sending emails, follow-ups, reply monitoring, and meeting scheduling behave correctly and reflect accurately across the dashboard, mailbox, and monitoring views.

### Activity 6.1: Campaign UI Input & Execution Flow Validation

- Validate the pages rendering correctly across the desktop.

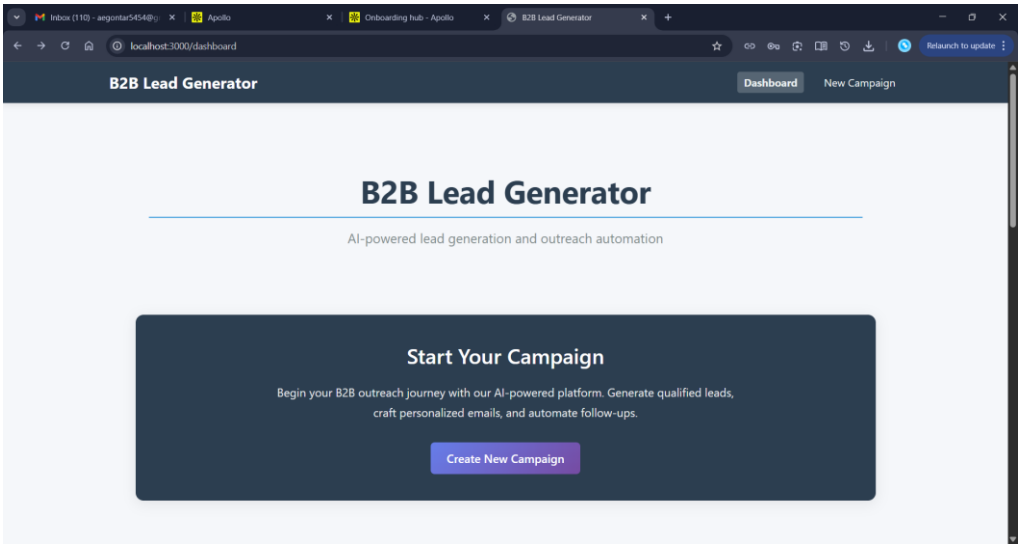



*Figure 61: Home Page*

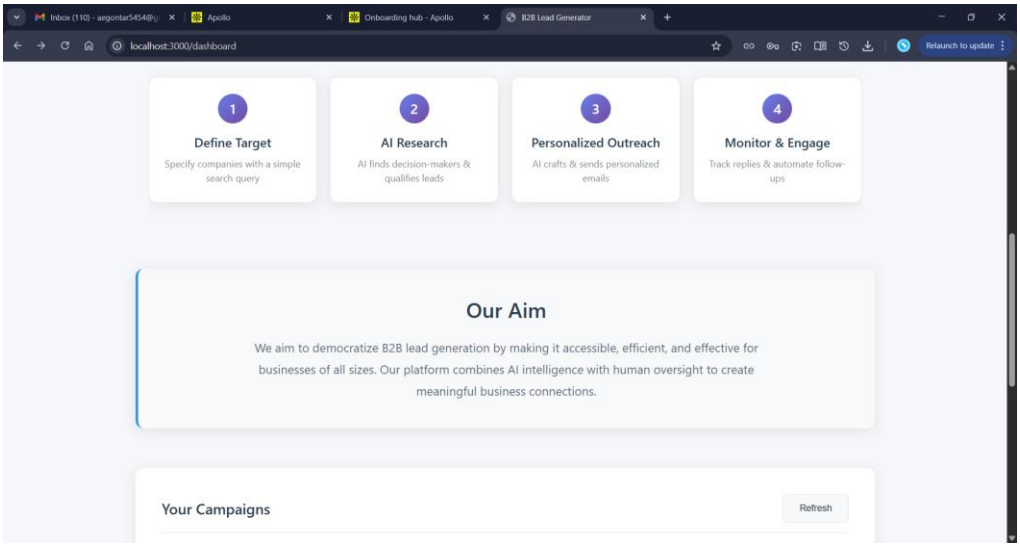



*Figure 62: About Section*

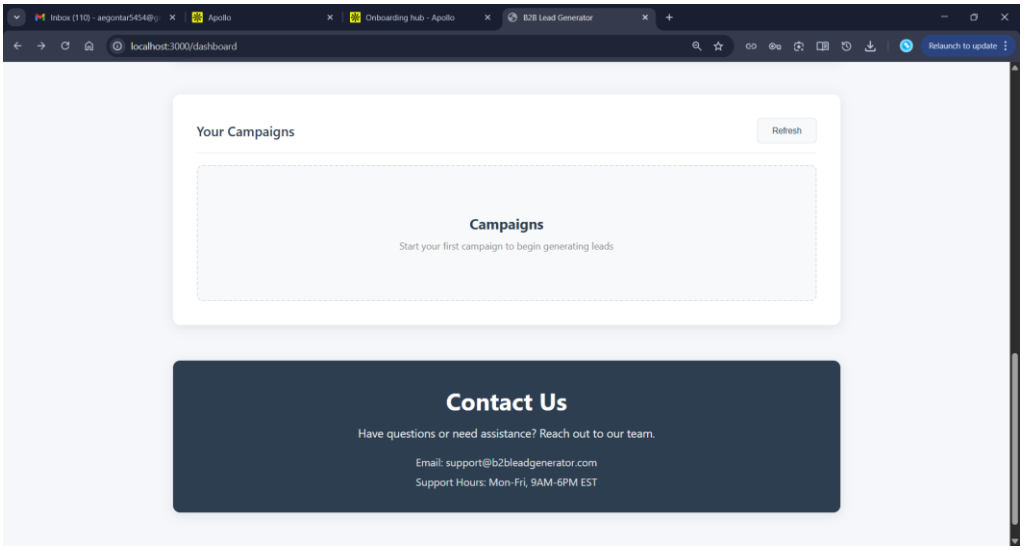



*Figure 63: Campaign History*

- Validate the **Campaign Setup UI**, ensuring users can:
  – Select campaign mode
  – Enter a valid search query
  – Provide sender profile details (company name, role, description)
  – Resume an existing campaign using a thread ID

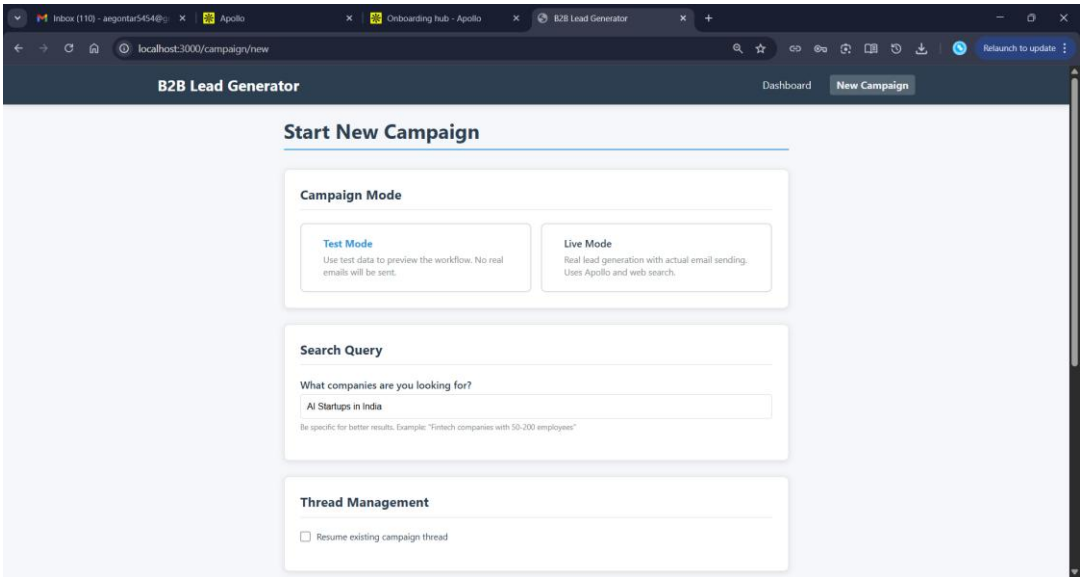



*Figure 64: Entering the user input*

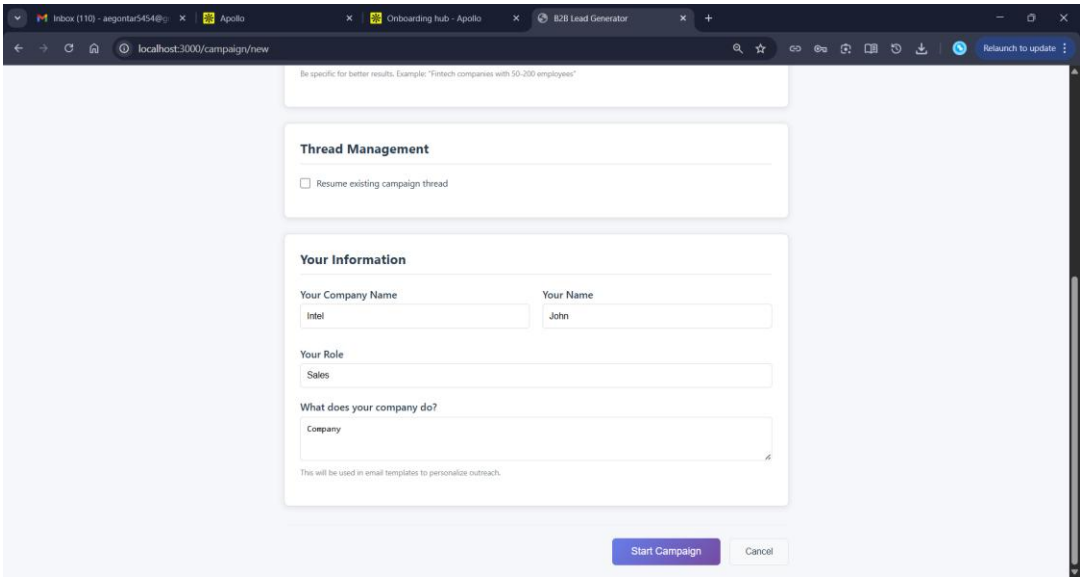



*Figure 65: User Detail*

- Verify form-level validations:
  – Empty query is rejected
  – Incomplete sender profile fields are flagged
  – Invalid thread ID input is handled gracefully

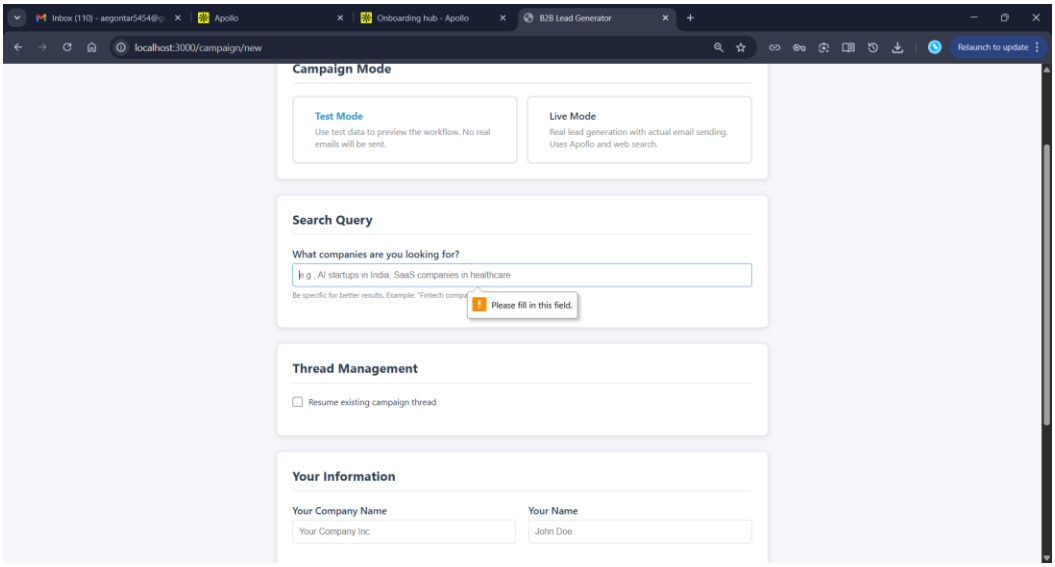



*Figure 66: Validation*

**Activity 6.2: Email Approval, Sending & Follow-up Visibility**

- Validate **email drafting display** in the UI before sending:
  – Subject line and body preview
  – Recipient email address
  – Drafted status indicator

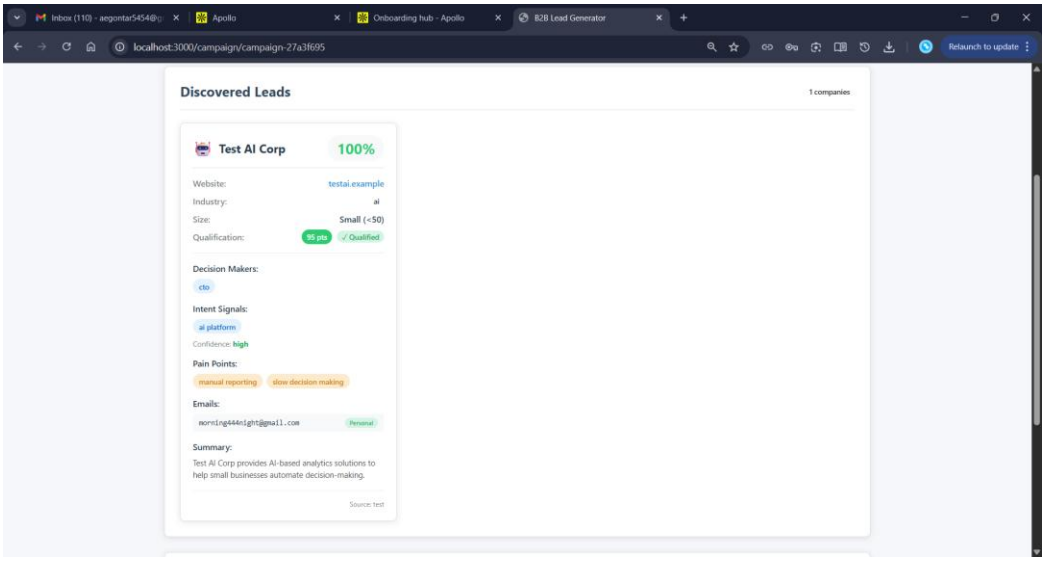



*Figure 67: Lead Boards*

- Test **Approve / Reject email options**:
  – Approve → emails are sent via SMTP
  – Reject → workflow terminates safely without sending emails

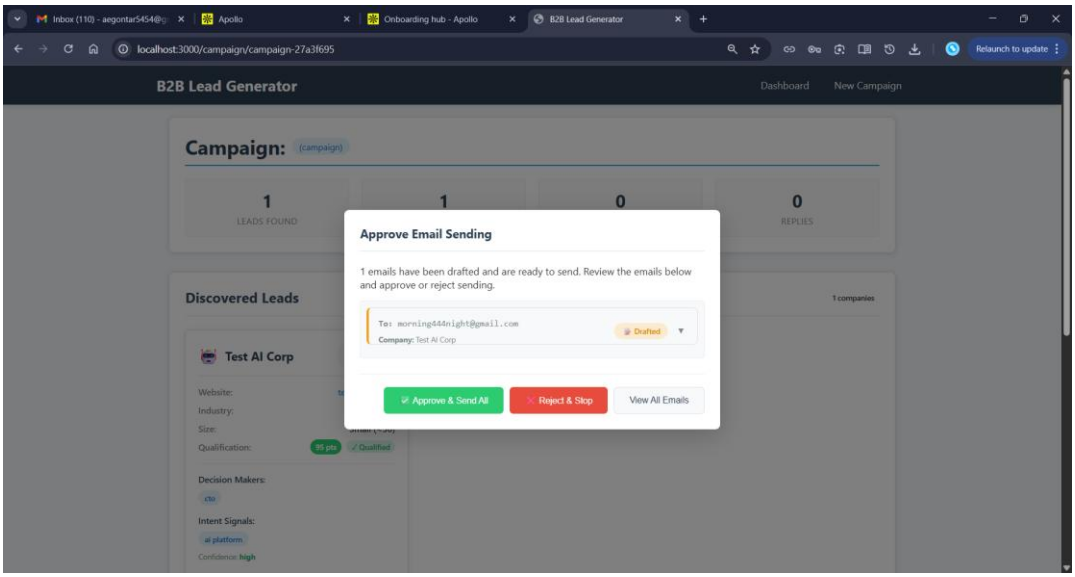



*Figure 68: Approve email*

- Verify **sent emails appear in the mailbox**:
  – Check Gmail "Sent" folder for outbound emails
  – Validate subject, content, and recipient accuracy
  – Confirm message IDs are attached for reply tracking

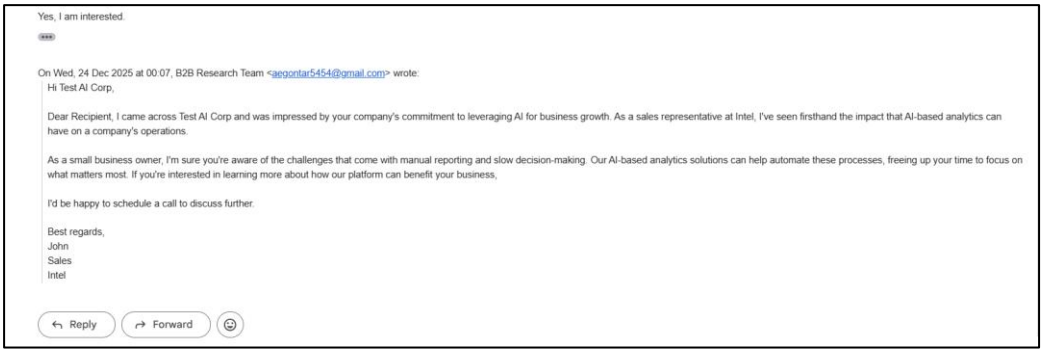



*Figure 69: Email form inbox*

- Validate **follow-up email workflow**:
  – Follow-up 1 is triggered after defined time threshold
  – Follow-up 2 is sent if no response is received
  – Follow-up emails are visible in the mailbox with correct threading

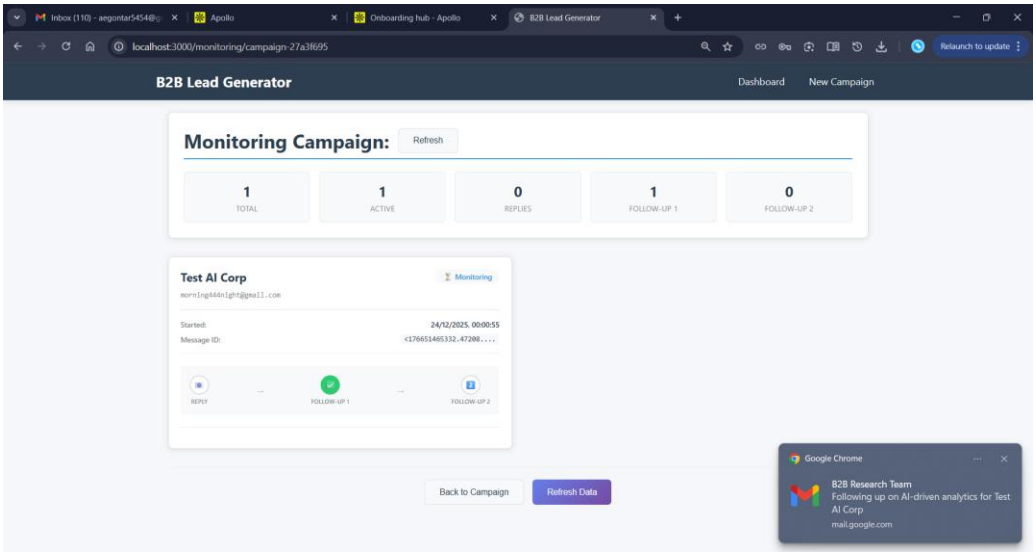



*Figure 70: Tracking Progress*

- Confirm follow-up status is reflected in:
  – Monitoring dashboard
  – Progress indicators (Follow-up 1 / Follow-up 2)

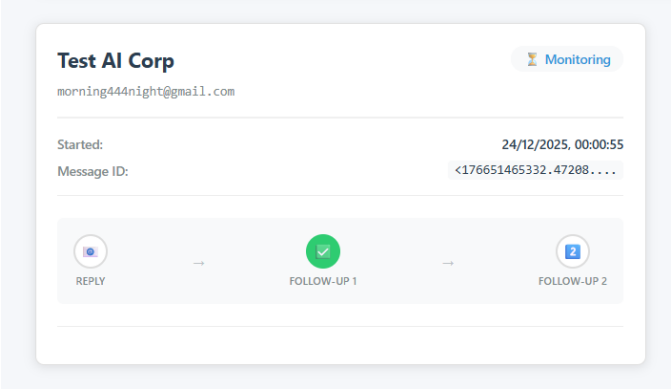



*Figure 71: Monitor Board*

## Activity 6.3: Monitoring, Reply Detection & Meeting Scheduling

- Validate automated **reply detection** using IMAP:
  – Replies are correctly identified using Message-ID headers
  – Reply status updates in real time on the monitoring dashboard

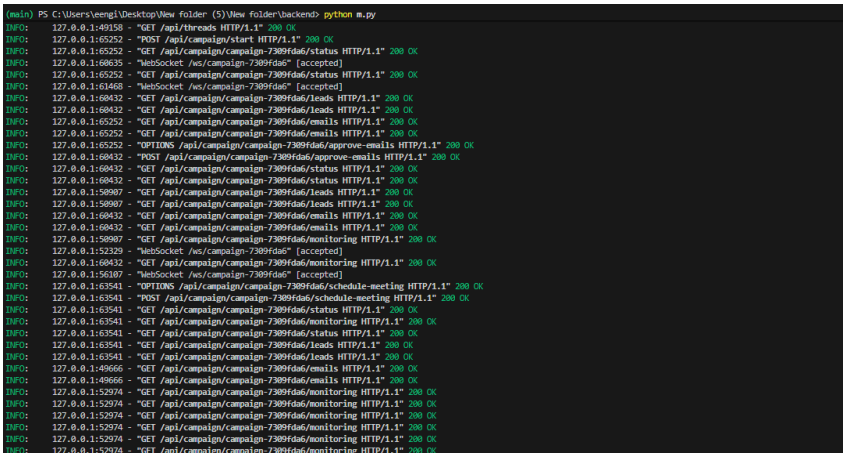



*Figure 72: VS Code Backend Log*

- Validate **Google Meet creation flow**:
  – Meeting date and time input validation
  – Google Calendar event creation
  – Google Meet link generation and storage

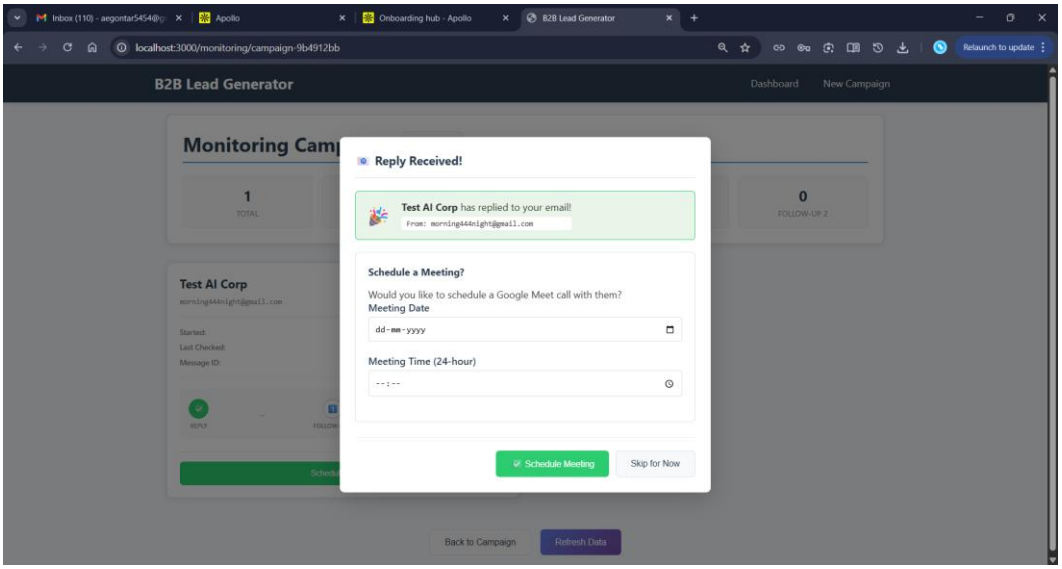



*Figure 73: Setting time and Date*

- Confirm meeting details are displayed correctly:
  – Meeting status indicator
  – Join Meet link visibility
  – Calendar event reference

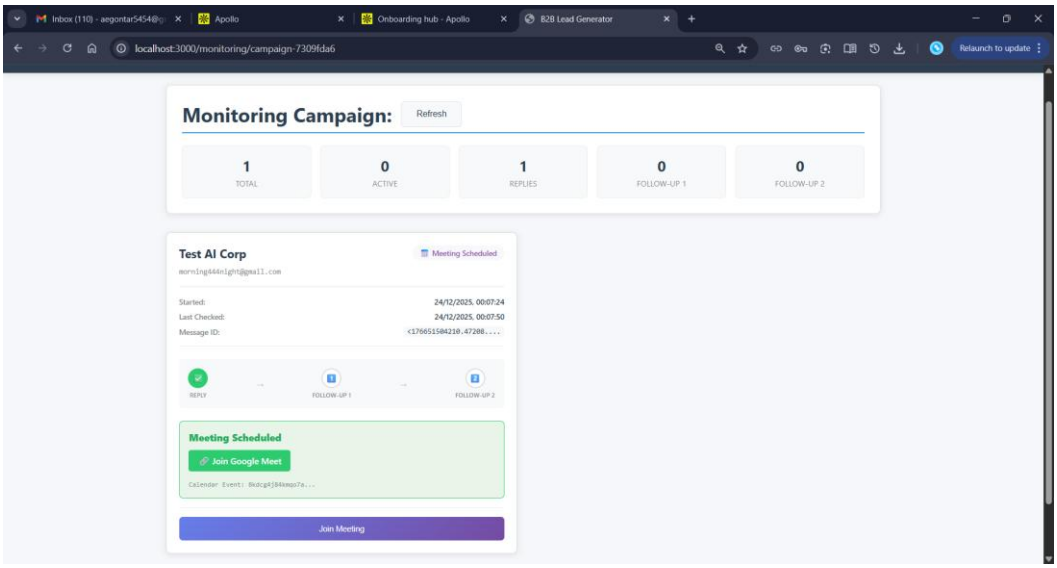



*Figure 74: Meeting is scheduled*

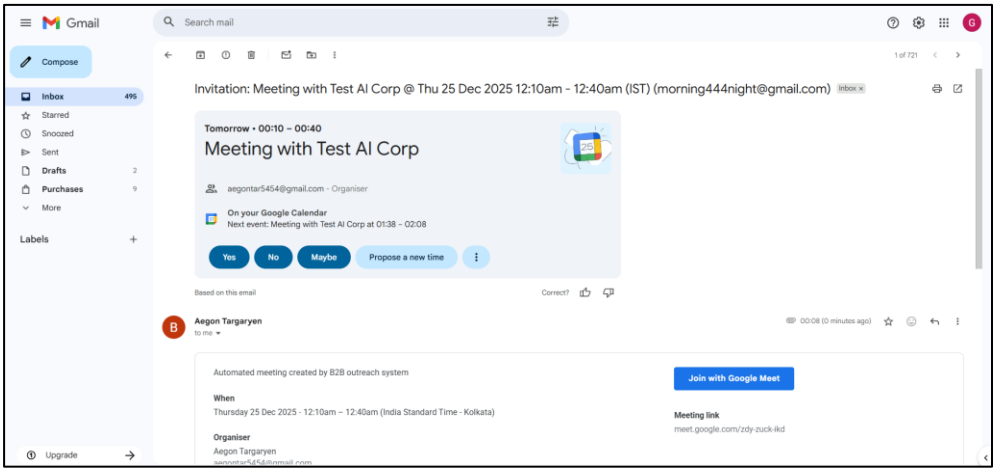



*Figure 75: Email conformation*

- Ensure monitoring lifecycle transitions correctly:
  – Active → Replied → Meeting Scheduled / Expired

### Activity 6.4: Deployment Preparation & Final Validation

- Configure environment variables for production deployment.
- Validate CORS settings and frontend-backend connectivity.
- Perform full end-to-end testing on deployed environment.
- Ensure stable execution across browsers and screen sizes.

# Conclusion

The AI-Powered B2B Lead Generation & Outreach Automation Platform demonstrates how intelligent automation can significantly transform traditional outbound sales and business development workflows. By combining AI-driven lead research, rule-based qualification, LLM-powered email generation, and human-in-the-loop control, the system delivers a structured, scalable, and reliable approach to B2B outreach. The platform eliminates the need for manual lead research and repetitive email drafting, allowing teams to focus on strategic engagement and relationship building.

Through its modular architecture built on LangGraph, FastAPI, and React, the system ensures clear separation of responsibilities while maintaining end-to-end workflow consistency. From campaign creation and lead discovery to email approval, follow-ups, reply monitoring, and meeting scheduling, each stage of the pipeline is transparent, traceable, and user-controlled. Real-time updates via WebSockets and persistent state management using SQLite further enhance system reliability and usability.

The integration of external services such as Apollo, SMTP/IMAP, and Google Calendar enables real-world operational readiness, ensuring that emails are delivered, replies are tracked directly from mailboxes, and meetings are scheduled seamlessly. Validation of UI flows, approval mechanisms, follow-up visibility, and monitoring dashboards confirms that the platform operates as a complete production-ready solution rather than a conceptual prototype.

Overall, this project provides a strong foundation for modern AI-assisted sales automation systems. It can be further extended with features such as advanced lead scoring models, CRM and ATS integrations, multilingual outreach, analytics dashboards, and adaptive AI follow-up strategies. With its scalable design and human-centric control model, the platform showcases how AI can enhance decision-making, improve outreach efficiency, and set a new standard for intelligent B2B engagement systems.