

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.

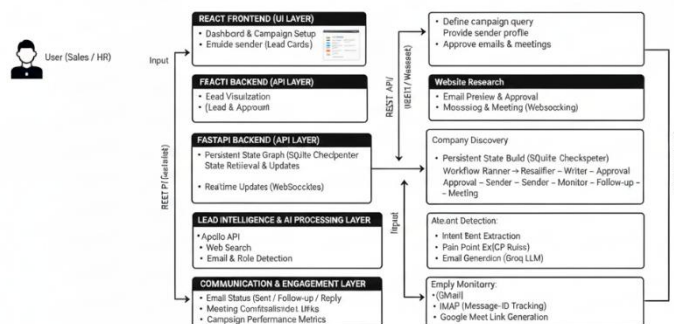


Figure 1: System architecture diagram

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.

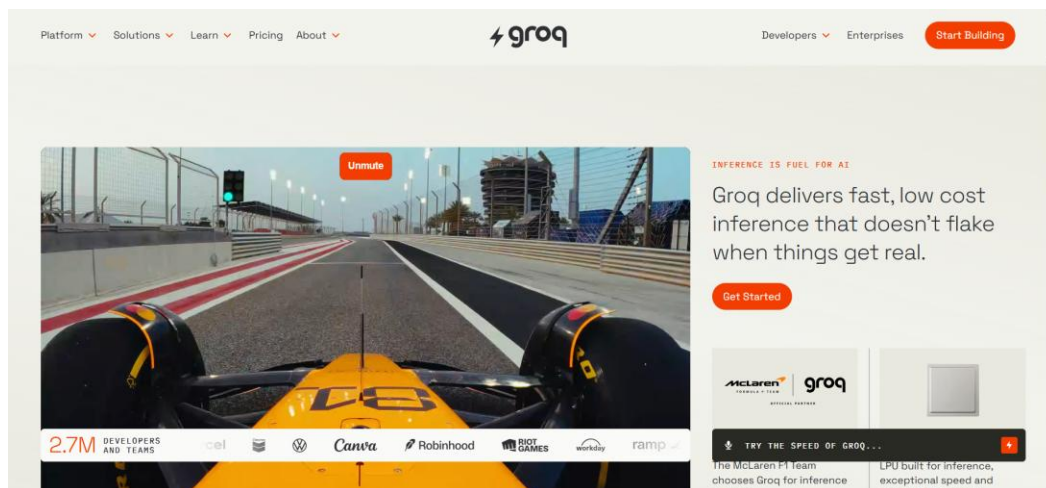


Figure 2: Groq Home Page

- Click on Start Building that navigate to dashboard.

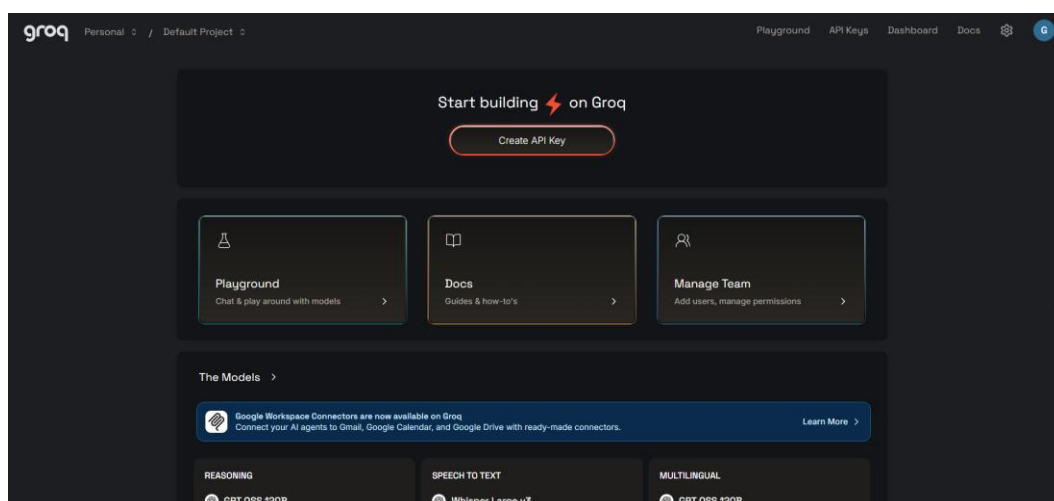


Figure 3: Groq Dashboard

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

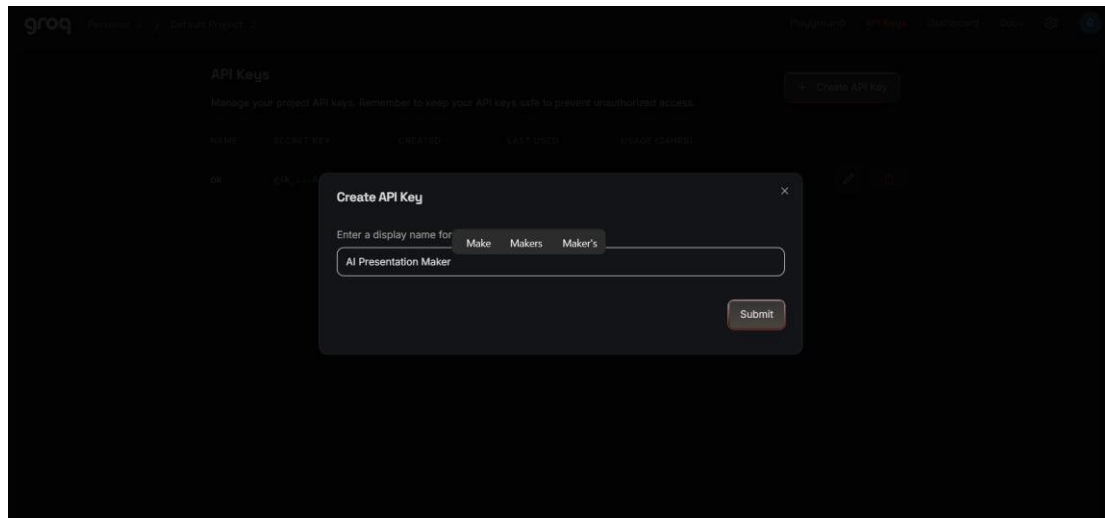


Figure 4: Creating API Key

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

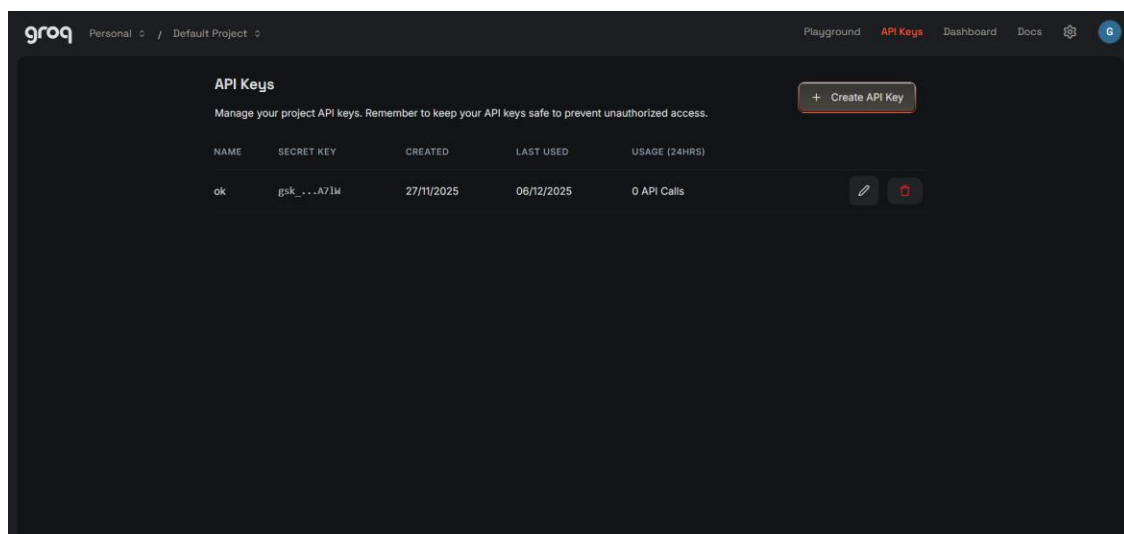


Figure 5: API Key

Activity 1.2: Apollo Account Creation

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

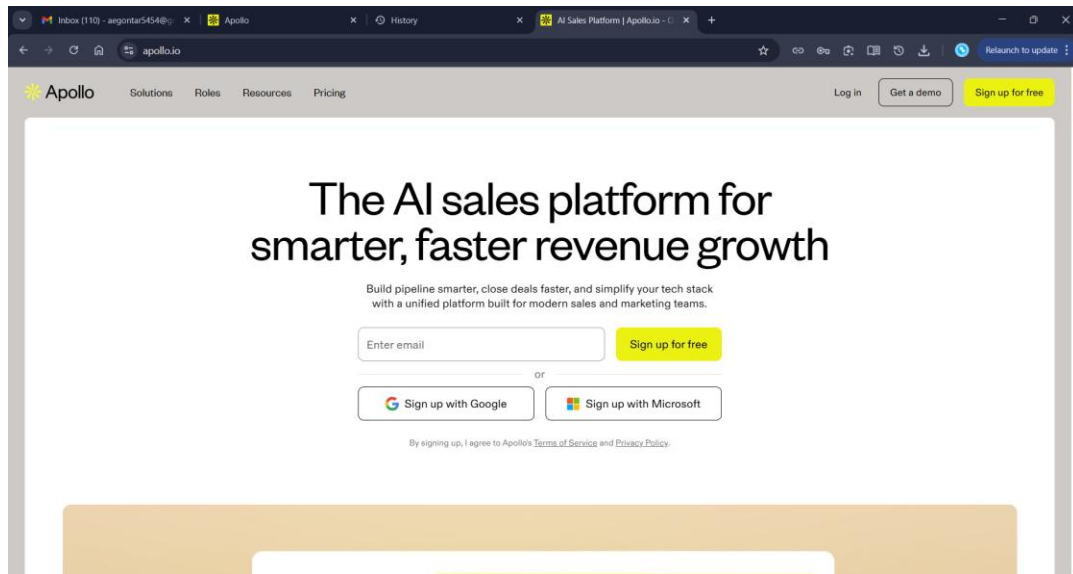


Figure 6: Apollo home page

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

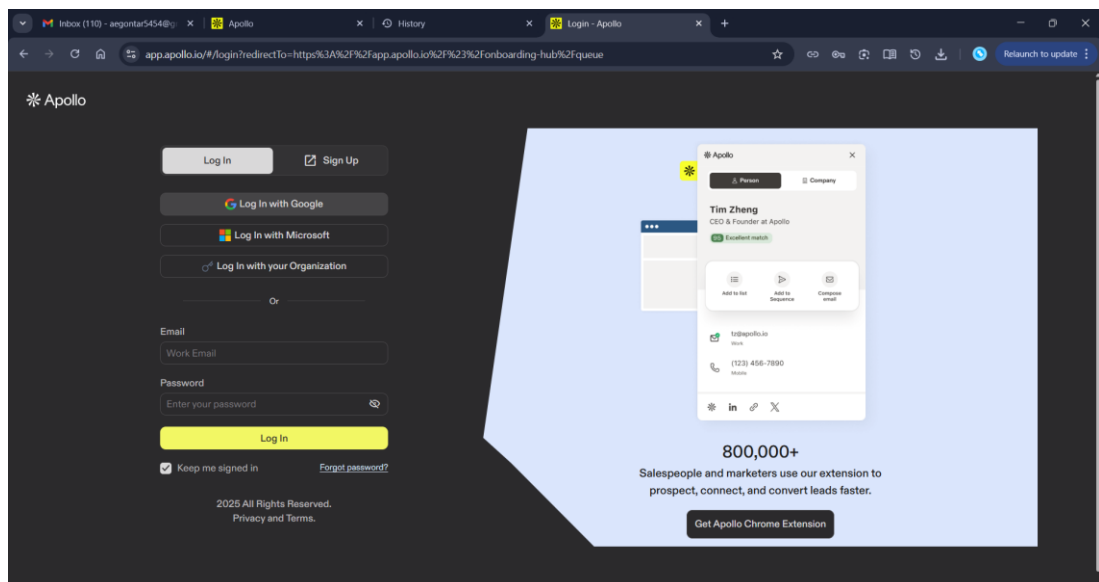


Figure 7: Sign option

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

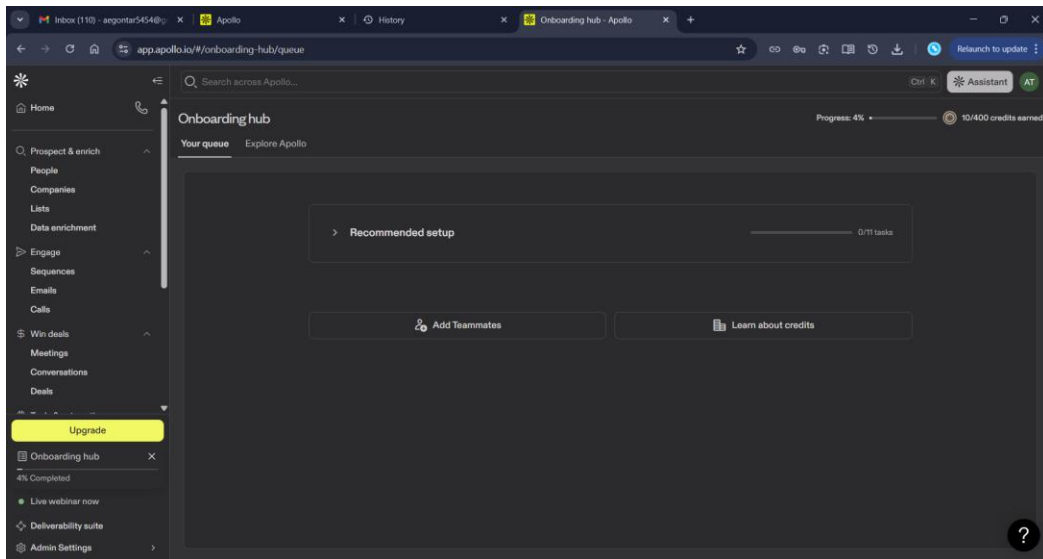


Figure 8: Apollo Dashboard

- 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.

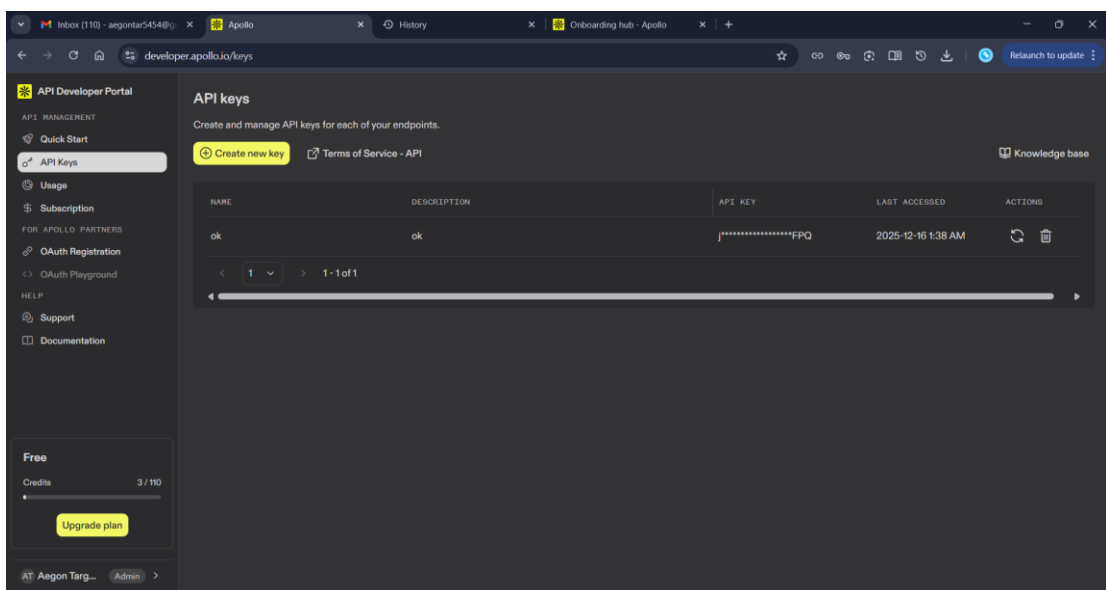


Figure 9: API Key

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:

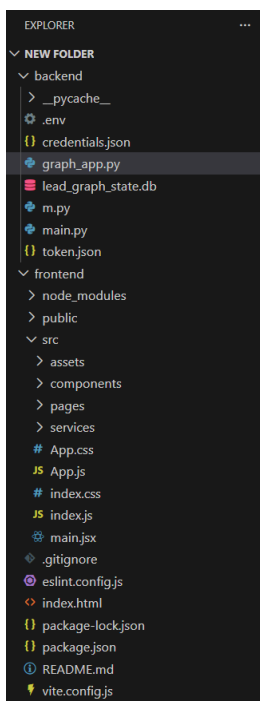


Figure 13: Folder Structure

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.

```

390 # -----
391 # TOOLS
392 # -----
393 @tool
394 def apollo_company_search(
395     query: str,
396     country: str = "India",
397     per_page: int = 5
398 ) -> List[Dict[str, Any]]:
399     """
400     Fetch company leads from Apollo API.
401     """
402     API_KEY = APOLLO_API_KEY
403     url = "https://api.apollo.io/api/v1/organizations/search"
404
405     headers = {
406         "Content-Type": "application/json",
407         "X-API-Key": API_KEY
408     }
409
410     payload = {
411         "q_keywords": query,
412         "organization_country": country,
413         "organization_num_employees_ranges": [
414             "51-200",
415             "201-500"
416         ],
417         "organization_is_public": False,
418         "page": 1,
419         "per_page": per_page
420     }
421
422     resp = requests.post(url, headers=headers, json=payload, timeout=15)
423
424     if resp.status_code != 200:
425         return []
426
427     data = resp.json()
428     companies = []
429
430

```

Figure 14: Apollo tool

```

424     resp = requests.post(url, headers=headers, json=payload, timeout=15)
425     if resp.status_code != 200:
426         return []
427
428     data = resp.json()
429     companies = []
430
431     for org in data.get("organizations", []):
432         if not org.get("website_url") or not org.get("primary_domain"):
433             continue
434
435         companies.append({
436             "company_name": org.get("name"),
437             "company_website": org.get("website_url"),
438             "domain": org.get("primary_domain"),
439             "industry": org.get("industry"),
440             "estimated_employees": org.get("estimated_num_employees"),
441             "keywords": org.get("keywords", []),
442             "source": "apollo"
443         })
444
445     return companies
446

```

Figure 15: Setting the sates

- Filter non-business domains and invalid websites.

```

447 @tool
448 def web_company_search(query: str) -> List[Dict[str, str]]:
449     """
450     Search the web for real B2B company websites matching the query.
451     Filters out media, blogs, directories, and aggregators.
452     """
453     companies = []
454     with DDGS() as ddgs:
455         for r in ddgs.text(query, max_results=15):
456             if r.get("href"):
457                 domain = urlparse(r["href"]).netloc.lower()
458                 if (
459                     domain
460                     and not any(bad in domain for bad in BAD_DOMAINS)
461                     and is_real_company_site(domain)
462                 ):
463                     parts = domain.split(".")
464                     if len(parts) >= 2:
465                         company_key = parts[-2]
466                     else:
467                         company_key = parts[0]
468                     companies.append({
469                         "company_name": company_key.replace("-", " ").title(),
470                         "company_website": f"https://{domain}",
471                         "domain": domain
472                     })
473     return companies[:5]
474

```

Figure 16: Web Tool

- Crawl company websites to extract structured textual content.

```

477 @tool
478 def deep_crawl_site(url: str, max_pages: int = 6) -> str:
479     """
480     Crawl a company website across internal pages
481     and return combined clean text.
482     """
483     visited: Set[str] = set()
484     to_visit = [url]
485     base_domain = urlparse(url).netloc
486     combined = ""
487
488     while to_visit and len(visited) < max_pages:
489         current = to_visit.pop(0)
490         if current in visited:
491             continue
492         visited.add(current)
493
494         try:
495             r = requests.get(current, headers=HEADERS, timeout=10)
496             soup = BeautifulSoup(r.text, "html.parser")
497             for t in soup(["script", "style", "noscript"]):
498                 t.decompose()
499             combined += " " + soup.get_text(" ", strip=True)
500
501             for a in soup.find_all("a", href=True):
502                 link = urljoin(current, a["href"])
503                 if urlparse(link).netloc == base_domain:
504                     to_visit.append(link)
505         except:
506             pass
507
508     return combined[:6000]
509

```

Figure 17: Deep Search tool

Activity 2.2: Lead Research & Enrichment

- Extract validated email addresses from company websites.

```

511 @tool
512 def extract_and_validate_emails(text: str, domain: str) -> List[str]:
513     """
514     Extract emails from website text and validate domain using MX records.
515     Falls back to role-based emails if none found.
516     """
517
518     parts = domain.lower().split(".")
519     if len(parts) >= 2:
520         root_domain = parts[-2] + "." + parts[-1]
521     else:
522         root_domain = domain.lower()
523
524     found = set(re.findall(EMAIL_REGEX, text))
525     emails = []
526
527     try:
528         dns.resolver.resolve(root_domain, "MX")
529     except:
530         return []
531
532     for e in found:
533         e = e.lower()
534         if e.endswith(f"@{root_domain}") and root_domain not in DISPOSABLE_DOMAINS:
535             emails.append(e)
536
537     if not emails:
538         emails = [f"{p}@{root_domain}" for p in ROLE_PREFIXES]
539
540     return list(set(emails))
541

```

Figure 18: Email extract and validate

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

```

543 @tool
544 def detect_decision_maker_roles(text: str) -> List[str]:
545     """
546     Detect decision-maker roles mentioned on the company website.
547     Returns roles only (no names).
548     """
549     roles = [
550         "ceo", "cto", "cfo",
551         "coo", "founder", "co-founder",
552         "director", "head organizer", "vp",
553         "vice president"
554     ]
555
556     text_lower = text.lower()
557     found = []
558
559     for role in roles:
560         if role in text_lower:
561             found.append(role)
562
563     return list(set(found))

```

Figure 19: Position Finder

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

```

600 while state["companies"]:
601     company = state["companies"].pop(0)
602
603     site_text = deep_crawl_site.invoke(company["company_website"])
604
605     emails = extract_and_validate_emails.invoke({
606         "text": site_text,
607         "domain": company["domain"]
608     })
609
610     decision_maker_roles = detect_decision_maker_roles.invoke(site_text)
611
612     industry = company.get("industry", "unknown")
613
614     company_size = normalize_company_size(
615         company.get("estimated_employees")
616     )
617
618     intent_signals = map_keywords_to_intent(
619         company.get("keywords", [])
620     )
621
622     pain_points = []
623
624     prompt = f"""
625     Extract business facts ONLY from text.
626     DO NOT invent names for decision_makers.
627     Try to fill the intent_signals & pain_points field with help of text.
628     If intent_signals:
629         - Add short phrases ONLY if supported by phrases in the TEXT
630         (e.g. "ai platform", "enterprise software", "automation solution"
631         - If no clear signals exist, return unknown.
632     If pain_points:
633         - Add short phrases ONLY if supported by phrases in the TEXT
634         - If no pain points are mentioned, return unknown.
635
636     Return JSON ONLY.
637
638     {{
639         "industry": "ai | saas | fintech | ecommerce | other | health-care",
640         "company_size": "small | medium | large | unknown",
641         "intent_signals": [],
642         "pain_points": []
643     }}
644
645     TEXT:
646     {site_text[:3500]}
647     """
648
649     response = model.invoke([HumanMessage(content=prompt)])
650     raw = response.content.replace("'", "").replace('"', "").strip()
651 
```

Figure 20: Extracting information using the LLM

```

673 try:
674     enriched = json.loads(raw)
675     industry = normalize_industry(
676         enriched.get("industry", industry)
677     )
678
679     company_size = normalize_company_size_llm(
680         enriched.get("company_size", company_size)
681     )
682     pain_points = enriched.get("pain_points", [])
683
684     if not intent_signals:
685         intent_signals = enriched.get("intent_signals", [])
686
687 except:
688     pain_points = []
689
690
691     summary_prompt = f"""
692     Summarize the following company website text in a clear, descriptive, business-focused way.
693     Do NOT say "Here is", "Summary:", or similar phrases.
694     You must return ONLY the summary text.
695
696     Rules:
697     - Use ONLY the provided text
698     - Do NOT add assumptions
699     - Do NOT add opinions
700     - Do NOT invent products or services
701     - Write 4-6 concise sentences
702     - Focus on: what the company does, who it serves, and key offerings
703
704     TEXT:
705     {site_text[:3000]}
706     """
707
708     summary_response = model.invoke(
709         [HumanMessage(content=summary_prompt)]
710     )
711
712     website_summary = summary_response.content.strip()
713
714     email_quality = get_email_quality(emails)
715
716     intent_confidence = get_intent_confidence(intent_signals)
717
718     research_confidence = calculate_research_confidence(
719         industry=industry,
720         company_size=company_size,
721         emails=emails,
722         people=decision_maker_roles,
723         intent_signals=intent_signals
724     )
725
726     state["leads"].append({
727         "company_name": company["company_name"],
728         "company_website": company["company_website"],
729         "domain": company["domain"],
730
731         "industry": industry,
732         "company_size": company_size,
733
734         "intent_signals": intent_signals,
735         "intent_confidence": intent_confidence,
736 
```

Figure 21: Summary for the extracted text

```
727     state["leads"].append({
728         "company_name": company["company_name"],
729         "company_website": company["company_website"],
730         "domain": company["domain"],
731
732         "industry": industry,
733         "company_size": company_size,
734
735         "intent_signals": intent_signals,
736         "intent_confidence": intent_confidence,
737
738         "pain_points": pain_points,
739
740         "decision_makers": decision_maker_roles,
741
742         "validated_emails": emails,
743         "email_quality": email_quality,
744
745         "website_summary": website_summary,
746         "website_text_sample": site_text[:900],
747
748         "research_confidence": research_confidence,
749
750         "source": company.get("source", state.get("source", "web")),
751     })
752
753     return state
754
```

Figure 22: Storing the values in state

Activity 2.3: Lead Qualification Engine

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

```
216 def calculate_research_confidence(
217     industry: str,
218     company_size: str,
219     emails: List[str],
220     people: List[str],
221     intent_signals: List[str]
222 ) -> float:
223     score = 0.0
224
225     if industry != "unknown":
226         score += 0.2
227     if company_size != "unknown":
228         score += 0.2
229     if emails:
230         score += 0.2
231     if intent_signals:
232         score += 0.2
233     if people:
234         score += 0.2
235
236     return round(score, 2)
```

Figure 23: ICP Scoring

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

```

136 def is_real_company_site(domain: str) -> bool:
137     reject_keywords = [
138         "news", "blog", "mag", "tracker", "directory",
139         "listing", "media", "funding", "startup",
140         "magazine", "portal", "wiki"
141     ]
142     return not any(k in domain for k in reject_keywords)
143
144 def normalize_company_size(emp_count: int | None) -> str:
145     if not emp_count:
146         return "unknown"
147     if emp_count <= 50:
148         return "small"
149     if emp_count <= 250:
150         return "medium"
151     if emp_count <= 1000:
152         return "large"
153     return "enterprise"
154
155 def map_keywords_to_intent(keywords: List[str]) -> List[str]:
156     text = " ".join(k.lower() for k in keywords)
157     intents = []
158
159     if "artificial intelligence" in text or "machine learning" in text:
160         intents.append("ai platform")
161     if "saas" in text or "enterprise software" in text:
162         intents.append("enterprise software")
163     if "lead generation" in text or "b2b" in text:
164         intents.append("lead generation platform")
165     if "automation" in text:
166         intents.append("automation solution")
167
168     return list(set(intents))
169
170 def normalize_industry(raw: str) -> str:
171     if not raw:
172         return "unknown"
173
174     priority = ["ai", "saas", "fintech", "ecommerce", "health-care", "other"]
175
176     raw = raw.lower()
177     for p in priority:
178         if p in raw:
179             return p
180
181     return "unknown"

```

Figure 24: Extracting information from Apollo

```

183 def normalize_company_size_llm(raw: str) -> str:
184     if not raw:
185         return "unknown"
186
187     priority = ["small", "medium", "large", "unknown"]
188
189     raw = raw.lower()
190     for p in priority:
191         if p in raw:
192             return p
193
194     return "unknown"
195
196 def get_email_quality(emails: List[str]) -> str:
197     if not emails:
198         return "none"
199
200     for e in emails:
201         local = e.split("@")[0]
202         if local not in ROLE_PREFIXES:
203             return "personal"
204
205     return "role_based"
206
207
208 def get_intent_confidence(intent_signals: List[str]) -> str:
209     if not intent_signals:
210         return "low"
211     if len(intent_signals) == 1:
212         return "medium"
213     return "high"
214
215

```

Figure 25: Extractor

- Classify leads as qualified or unqualified with explainable scoring.

```

758 def qualifier_node(state: LeadState) -> LeadState:
759     """
760     Rule-based qualification of researched leads.
761     """
762     if state.get("phase") == "monitor":
763         return state
764     if state.get("start_from_writer"):
765         return state
766
767     qualified_results = []
768
769     for lead in state["leads"]:
770         score = 0
771         reasons = []
772
773         if lead["industry"] in ICP_CONFIG["industries"]:
774             score += 25
775             reasons.append("Industry matches ICP")
776
777         if lead["company_size"] in ICP_CONFIG["company_sizes"]:
778             score += 20
779             reasons.append("Company size matches ICP")
780
781         if lead.get("decision_makers"):
782             score += 20
783             reasons.append("Decision-maker role found")
784
785         if lead["email_quality"] == "personal":
786             score += 15
787             reasons.append("Personal email found")
788         elif lead["email_quality"] == "role_based":
789             score += 10
790             reasons.append("Role-based email found")
791
792         if lead["intent_confidence"] == "high":
793             score += 20
794             reasons.append("High intent detected")
795         elif lead["intent_confidence"] == "medium":
796             score += 10
797             reasons.append("Medium intent detected")
798
799         qualified_results.append({
800             "company_name": lead["company_name"],
801             "domain": lead["domain"],
802             "qualification_score": score,
803             "qualified": score >= ICP_CONFIG["min_score"],
804             "qualification_reason": reasons
805         })
806
807     state["qualification"] = qualified_results
808
809     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.

```

43
44 # =====
45 # LLM
46 # =====
47 model = ChatGroq(
48     api_key=GROQ_API_KEY,
49     model="llama-3.1-8b-instant",
50     temperature=0
51 )
52
53 # =====
54 # Structure LLM
55 # =====
56 class EmailDraft(BaseModel):
57     subject: str = Field(
58         description="Short, Professional & Attractive subject line under 60 characters"
59     )
60     body: str = Field(
61         description="Plain text 828 cold email body, 100-200 words, no placeholders"
62     )
63
64 structured_email_model = model.with_structured_output(EmailDraft)
65

```

Figure 27: LLM Setup

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

```

100 def writer_node(state: LeadData) -> LeadData:
101     """
102     Write outreach emails: ONLY for qualified leads
103     using schema-enforced structured output.
104     """
105     if state.get("phase") == "monitor":
106         return state
107
108     emails_to_send = []
109
110     for q in state["qualification"]:
111         if not q["qualified"]:
112             continue
113
114         lead = next(
115             (l for l in state["leads"] if l["company_name"] == q["company_name"]),
116             None
117         )
118         if not lead:
119             continue
120
121         sender = state["sender_profile"]
122
123         prompt = """
124         Write a short, professional 828 cold email.
125
126         Rules:
127         - No emojis
128         - No hype
129         - No placeholders like [Your Name] or [Recipient]
130         - Plain text only
131         - 100 - 200 words
132         - Mention company context naturally
133         - Clear, soft CTA
134         - Professional tone
135         - Must have 3-4 paragraph
136
137         Sender Information:
138         - Name: {sender["sender_name"]}
139         - Role: {sender["sender_role"]}
140         - Company: {sender["company_name"]}
141         - What we do: {sender["company_description"]}
142
143         Example:
144         Hi {lead["company_name"]} team,
145
146         Dear Recipient, I came across Test AI Corp and was impressed by your company's commitment to leveraging AI for business growth.
147
148         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 time.
149
150         I'd be happy to schedule a call to discuss further.
151

```

Figure 28: Prompting for LLM

```

963 Example:
964 Hi (lead["company_name"]),\n\n
965
966 Dear Recipient, I came across Test AI Corp and was impressed by your company's commitment to leveraging AI for business growth.
967
968 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.
969
970 I'd be happy to schedule a call to discuss further.
971
972 Best regards,\n\n
973 Sender Name
974 Sender Role
975 sender_company
976
977 Company Summary:
978 (lead["website_summary"])
979
980 Pain Points:
981 (lead["pain_points"])
982
983 Intent Signals:
984 (lead["intent_signals"])
985
986
987 email_draft: EmailDraft = structured_email_model.invoke(
988     [HumanMessage(content=prompt)]
989 )
990
991 for email_addr in lead["validated_emails"]:
992     emails_to_send.append({
993         "company_name": lead["company_name"],
994         "email": email_addr,
995         "email_subject": email_draft.subject,
996         "email_body": email_draft.body
997     })
998
999 state["emails"] = emails_to_send
1000 return state
1001

```

Figure 29: Updating the state

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

```

887 email_draft: EmailDraft = structured_email_model.invoke(
888     [HumanMessage(content=prompt)]
889 )
890
891 for email_addr in lead["validated_emails"]:
892     emails_to_send.append({
893         "company_name": lead["company_name"],
894         "email": email_addr,
895         "email_subject": email_draft.subject,
896         "email_body": email_draft.body
897     })
898
899 state["emails"] = emails_to_send
900 return state

```

Figure 30: Updating State

Activity 3.2: Human-in-the-Loop Approval

- Implement approval checkpoints for outbound email sending.

```

983 def human_send_approval_node(state: LeadState) -> LeadState:
984     """
985     Pause graph and wait for human input:
986     send_first_email: yes / no
987     """
988     # If decision already exists, just continue
989     if state.get("human_decision", {}).get("send_first_email") is not None:
990         return state
991
992     return state
993
994 def human_send_router(state: LeadState):
995     decision = state["human_decision"].get("send_first_email")
996     if decision == "yes":
997         return "sender"
998     return END
999

```

Figure 31: Approval node

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

```

1035 def human_meeting_decision_node(state: LeadState) -> LeadState:
1036     m = state["active_monitor"]
1037
1038     if not m:
1039         return state
1040
1041     if state["human_decision"].get("send_meeting_email") is not None:
1042         return state
1043
1044     return state
1045
1046 def human_meeting_router(state: LeadState):
1047     if state["human_decision"].get("send_meeting_email") == "yes":
1048         return "meeting"
1049     return "monitor"
1050

```

Figure 32: Meet Approval

Activity 3.3: Email Sending Engine

- Implement secure SMTP-based email delivery.

```

904 def sender_node(state: LeadState) -> LeadState:
905     """
906     Sender Agent: Sends emails and logs Message-IDs for monitoring.
907     """
908     if state.get("phase") == "monitor":
909         return state
910
911     sent_logs = []
912
913     with smtplib.SMTP(SMTP_CONFIG["host"], SMTP_CONFIG["port"]) as server:
914         server.starttls()
915         server.login(
916             SMTP_CONFIG["username"],
917             SMTP_CONFIG["password"]
918         )
919
920         for item in state.get("emails", []):
921             try:
922                 msg = MIMEText(item["email_body"], "plain")
923
924                 msg["From"] = f'{SMTP_CONFIG["from_name"]} <{SMTP_CONFIG["username"]}@>'
925                 msg["To"] = item["email"]
926                 msg["Subject"] = item["email_subject"]
927
928                 message_id = make_msgid()
929                 msg["Message-ID"] = message_id
930
931                 msg.attach(MIMEText(item["email_body"], "plain"))
932
933                 server.send_message(msg)
934
935                 sent_logs.append({
936                     "company_name": item["company_name"],
937                     "email": item["email"],
938                     "message_id": message_id,
939                     "status": "sent",
940                     "sent_at": datetime.utcnow().isoformat()
941                 })
942             except Exception as e:
943                 sent_logs.append({
944                     "company_name": item["company_name"],
945                     "email": item["email"],
946                     "message_id": message_id,
947                     "status": "failed",
948                     "error": str(e),
949                     "sent_at": datetime.utcnow().isoformat()
950                 })
951
952     state["email_send_logs"] = sent_logs
953
954     now = datetime.now(timezone.utc).isoformat()
955
956     state["monitoring"] = []
957

```

Figure 33: SMTP


```

953     state["email_send_logs"] = sent_logs
954
955     now = datetime.now(timezone.utc).isoformat()
956
957     state["monitoring"] = []
958
959     for log in state["email_send_logs"]:
960         if log["status"] != "sent":
961             continue
962
963         state["monitoring"].append({
964             "company_name": log["company_name"],
965             "email": log["email"],
966             "message_id": log["message_id"],
967
968             "monitor_started_at": now,
969             "last_checked_at": None,
970
971             "reply_received": False,
972             "meeting_scheduled": False,
973
974             "followup_1_sent": False,
975             "followup_2_sent": False,
976
977             "monitor_status": "active"
978         })
979
980     state["phase"] = "monitor"
981     return state

```

Figure 34: State Update

- Attach unique message IDs for tracking replies.

```

924     msg["From"] = f'{SMTP_CONFIG["from_name"]} <{SMTP_CONFIG["username"]} >'
925     msg["To"] = item["email"]
926     msg["Subject"] = item["email_subject"]
927
928     message_id = make_msgid()
929     msg["Message-ID"] = message_id
930
931     msg.attach(MIMEText(item["email_body"], "plain"))
932
933     server.send_message(msg)
934
935     sent_logs.append({
936         "company_name": item["company_name"],
937         "email": item["email"],
938         "message_id": message_id,
939         "status": "sent",
940         "sent_at": datetime.now().isoformat()
941     })
942

```

Figure 35: Unique ID

- Log sent email metadata for monitoring and reporting.

```
Click to add a breakpoint
935
936     sent_logs.append({
937         "company_name": item["company_name"],
938         "email": item["email"],
939         "message_id": message_id,
940         "status": "sent",
941         "sent_at": datetime.utcnow().isoformat()
942     })
943
944     except Exception as e:
945         sent_logs.append({
946             "company_name": item["company_name"],
947             "email": item["email"],
948             "message_id": message_id,
949             "status": "failed",
950             "error": str(e),
951             "sent_at": datetime.utcnow().isoformat()
952         })
953
954     state["email_send_logs"] = sent_logs
```

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.

```

249 def check_reply_for_message_id(message_id: str) -> bool:
250     """
251     Check Gmail inbox for replies to a specific Message-ID
252     using IMAP HEADER search (In-Reply-To / References).
253     """
254
255     # Gmail IMAP requires Message-ID WITHOUT <>
256     search_id = message_id.strip("<>")
257
258     mail = imaplib.IMAP4_SSL(IMAP_CONFIG["host"])
259     mail.login(IMAP_CONFIG["username"], IMAP_CONFIG["password"])
260     mail.select("inbox")
261
262     # Search by In-Reply-To
263     status, data = mail.search(
264         None,
265         f'HEADER "In-Reply-To" "{search_id}"'
266     )
267
268     if status == "OK" and data[0]:
269         mail.logout()
270         return True
271
272     # Fallback: search by References
273     status, data = mail.search(
274         None,
275         f'HEADER "References" "{search_id}"'
276     )
277
278     if status == "OK" and data[0]:
279         mail.logout()
280         return True
281
282     mail.logout()
283     return False
284
  
```

Figure 37: Monitoring the email

- Update campaign state when replies are received.

```

1003 def monitor_node(state: LeadState) -> LeadState:
1004     now = datetime.now(timezone.utc)
1005
1006     for m in state["monitoring"]:
1007         if m["monitor_status"] != "active":
1008             continue
1009
1010         start = datetime.fromisoformat(m["monitor_started_at"])
1011         elapsed = (now - start).total_seconds()
1012
1013         if not m["reply_received"]:
1014             if check_reply_for_message_id(m["message_id"]):
1015                 m["reply_received"] = True
1016                 m["last_checked_at"] = now.isoformat()
1017                 state["active_monitor"] = m
1018                 return state
1019
1020         if elapsed >= 60 and not m["followup_1_sent"]:
1021             state["active_monitor"] = m
1022             return state
1023
1024         if elapsed >= 420 and not m["followup_2_sent"]:
1025             state["active_monitor"] = m
1026             return state
1027
1028         if elapsed >= 600:
1029             m["monitor_status"] = "expired"
1030
1031     state["active_monitor"] = {}
1032     return state
1033
  
```

Figure 38: Monitor node

Activity 4.2: Automated Follow-up Workflow

- Trigger follow-up emails based on time thresholds.

```

1052 def followup_node(state: LeadState) -> LeadState:
1053     m = state["active_monitor"]
1054
1055     lead = next(
1056         (1 for l in state["leads"] if l["company_name"] == m["company_name"]),
1057         {}
1058     )
1059
1060     followup_no = 1 if not m["followup_1_sent"] else 2
1061
1062     draft = generate_followup_email(lead, followup_no, state["sender_profile"])
1063
1064     with smtplib.SMTP(SMTP_CONFIG["host"], SMTP_CONFIG["port"]) as server:
1065         server.starttls()
1066         server.login(
1067             SMTP_CONFIG["username"],
1068             SMTP_CONFIG["password"]
1069         )
1070
1071         msg = MIMEMultipart()
1072         msg["From"] = f'{SMTP_CONFIG["from_name"]} <{SMTP_CONFIG["username"]} >'
1073         msg["To"] = m["email"]
1074         msg["Subject"] = draft.subject
1075
1076         msg.attach(MIMEText(draft.body, "plain"))
1077
1078         server.send_message(msg)
1079
1080     if not m["followup_1_sent"]:
1081         m["followup_1_sent"] = True
1082     else:
1083         m["followup_2_sent"] = True
1084
1085     state["active_monitor"] = {}
1086     return state
1087

```

Figure 39: Follow up emails

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

```

105 def generate_followup_email(lead: Dict[str, Any], followup_no: int, sender: Dict[str, Any]) -> EmailDraft:
106     prompt = """
107     Write a professional B2B follow-up email.
108
109     Rules:
110     - No emojis
111     - No typos
112     - No placeholders like [Your Name]
113     - Plain text only
114     - No jargon
115     - Polite and respectful
116     - Not pushy
117     - Clear soft CTA
118     - This is FOLLOW-UP # {followup_no}
119
120     Context:
121     Company: {lead["company_name"]}
122     Industry: {lead.get("industry")}
123     Pain Points: {lead.get("pain_points")}
124     Original Intent: {lead.get("intent_signals")}
125
126     Sender Information:
127     - Name: {sender["sender_name"]}
128     - Role: {sender["sender_role"]}
129     - Company: {sender["company_name"]}
130     - What we do: {sender["company_description"]}
131
132     Example:
133     Hi {lead["company_name"]},
134
135     Dear Recipient, I wanted to follow up on my previous email regarding how AI-driven analytics can help simplify reporting and improve decision-making.
136
137     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
138
139     If this sounds relevant, I'd be happy to share more or schedule a short call at your convenience.
140
141     Best regards,
142
143     Sender Name
144     Sender Role
145     Sender Company
146
147     Tone:
148     - Follow-up 1 = gentle reminder
149     - Follow-up 2 = final polite check-in
150     """
151     return structured_email_model.invoke(
152         [HumanMessage(content=prompt)]
153     )

```

Figure 40: Generating cold email

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

```

1013         if not m["reply_received"]:
1014             if check_reply_for_message_id(m["message_id"]):
1015                 m["reply_received"] = True
1016                 m["last_checked_at"] = now.isoformat()
1017                 state["active_monitor"] = m
1018                 return state
1019
1020         if elapsed >= 60 and not m["followup_1_sent"]:
1021             state["active_monitor"] = m
1022             return state
1023
1024         if elapsed >= 420 and not m["followup_2_sent"]:
1025             state["active_monitor"] = m
1026             return state
1027
1028         if elapsed >= 600:
1029             m["monitor_status"] = "expired"
1030
1031         state["active_monitor"] = {}
1032         return state
1033

```

Figure 41: Follow up conditions

Activity 4.3: Meeting Scheduling Automation

- Allow human confirmation for meeting scheduling.

```

1035 def human_meeting_decision_node(state: LeadState) -> LeadState:
1036     m = state["active_monitor"]
1037
1038     if not m:
1039         return state
1040
1041     if state["human_decision"].get("send_meeting_email") is not None:
1042         return state
1043
1044     return state
1045
1046 def human_meeting_router(state: LeadState):
1047     if state["human_decision"].get("send_meeting_email") == "yes":
1048         return "meeting"
1049     return "monitor"
1050

```

Figure 42: Meet Decision

- Create Google Calendar events with Google Meet links.

```

1093 def meeting_node(state: LeadState) -> LeadState:
1094     m = state["active_monitor"]
1095
1096     meeting_dt_str = state["human_decision"].get("meeting_datetime")
1097     if not meeting_dt_str:
1098         return state
1099
1100     # Convert "YYYY-MM-DD HH:MM" -> datetime
1101     start_dt = datetime.strptime(meeting_dt_str, "%Y-%m-%d %H:%M")
1102     end_dt = start_dt + timedelta(minutes=30)
1103
1104     service = get_calendar_service()
1105
1106     event = {
1107         "summary": f"Meeting with {m['company_name']}",
1108         "description": "Automated meeting created by B2B outreach system",
1109         "start": {
1110             "dateTime": start_dt.isoformat(),
1111             "timeZone": "Asia/Kolkata",
1112         },
1113         "end": {
1114             "dateTime": end_dt.isoformat(),
1115             "timeZone": "Asia/Kolkata",
1116         },
1117         "conferenceData": {
1118             "createRequest": {
1119                 "requestId": str(uuid.uuid4())
1120             }
1121         },
1122         "attendees": [
1123             {"email": m["email"]}
1124         ]
1125     }
1126
1127     created_event = service.events().insert(
1128         calendarId="primary",
1129         body=event,
1130         conferenceDataVersion=1,
1131         sendUpdates="all"
1132     ).execute()
1133
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
1143

```

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
1143
```

Figure 44: Storing the data

Figure 45: Campaign Setup


```

New folder > frontend > src > pages > CampaignSetup.js > default
6  const CampaignSetup = () => {
33  const handleSubmit = async (e) => {
53      try {
54          const requestData = {
55              query: query.trim(),
56              mode: mode,
57              thread_id: useExistingThread ? threadId.trim() : null,
58              sender_profile: senderProfile
59          };
60
61          const response = await campaignAPI.startCampaign(requestData);
62
63          navigate(`/campaign/${response.thread_id}`);
64      } catch (err) {
65          setError(err.response?.data?.detail || 'Failed to start campaign');
66          console.error('Campaign start error:', err);
67      } finally {
68          setloading(false);
69      }
70  };
71
72  return (
73      <div className="campaign-setup">
74          <h1>Start New Campaign</h1>
75
76          <form onSubmit={handleSubmit} className="campaign-form">
77              {error && (
78                  <div className="alert alert-error">
79                      {error}
80                  </div>
81              )}
82
83              { /* Mode Selection */ }
84              <div className="form-section">
85                  <h3>Campaign Mode</h3>
86                  <div className="mode-selector">
87                      <label className="mode-option">
88                          <input
89                              type="radio"
90                              name="mode"
91                              value="test"
92                              checked={mode === 'test'}
93                              onChange={e => setMode(e.target.value)}
94                          />
95                      <div className="mode-content">
96                          <div className="mode-icon"></div>
97                          <div>
98                              <div className="mode-title">Test Mode</div>
99                              <div className="mode-description">
100                                  Use test data to preview the workflow. No real emails will be sent.
101                              </div>
102                          </div>
103                      </div>
104                  </label>

```

Figure 46: Campaign Setup

```

106          <label className="mode-option">
107              <input
108                  type="radio"
109                  name="mode"
110                  value="live"
111                  checked={mode === 'live'}
112                  onChange={e => setMode(e.target.value)}
113              />
114              <div className="mode-content">
115                  <div className="mode-icon"></div>
116                  <div>
117                      <div className="mode-title">Live Mode</div>
118                      <div className="mode-description">
119                          Real lead generation with actual email sending. Uses Apollo and web search.
120                      </div>
121                  </div>
122              </div>
123          </label>
124      </div>
125  </div>
126
127  { /* Search Query */ }
128  <div className="form-section">
129      <h3>Search Query</h3>
130      <div className="form-group">
131          <label className="form-label">
132              What companies are you looking for?
133          </label>
134          <input
135              type="text"
136              className="form-control"
137              value={query}
138              onChange={e => setQuery(e.target.value)}
139              placeholder="e.g., AI startups in India, SaaS companies in healthcare"
140              required
141          />
142          <small className="form-hint">
143              Be specific for better results. Example: "Fintech companies with 50-200 employees"
144          </small>
145      </div>
146  </div>
147
148  { /* Thread Management */ }
149  <div className="form-section">
150      <h3>Thread Management</h3>
151      <div className="form-group">
152          <label className="form-check">
153              <input
154                  type="checkbox"
155                  checked={useExistingThread}
156                  onChange={e => setUseExistingThread(e.target.checked)}
157              />
158              <span>Resume existing campaign thread</span>

```

Figure 47: Campaign Setup

```

228 <div className="form-group">
229   <label className="form-label">What does your company do?</label>
230   <textarea
231     className="form-control"
232     name="company_description"
233     value={senderProfile.company_description}
234     onChange={handleSenderProfileChange}
235     placeholder="Briefly describe your company's products/services (1-2 sentences)"
236     rows="3"
237     required
238   />
239   <small className="form-hint">
240     This will be used in email templates to personalize outreach.
241   </small>
242 </div>
243 </div>
244
245 /* Submit */
246 <div className="form-actions">
247   <button
248     type="submit"
249     className="btn btn-primary btn-lg"
250     disabled={loading}
251   >
252     {loading ? (
253       <>
254         <div className="spinner spinner-sm"></div>
255         Starting Campaign...
256       </>
257     ) : (
258       'Start Campaign'
259     )}
260   </button>
261
262   <button
263     type="button"
264     className="btn btn-secondary"
265     onClick={() => navigate('/dashboard')}
266     disabled={loading}
267   >
268     Cancel
269   </button>
270 </div>
271 </form>
272 </div>
273 );
274 };
275
276 export default CampaignSetup;

```

Figure 48: Campaign Setup

```

New folder > frontend > src > pages > # CampaignViews > CampaignView
1 import React, { useState, useEffect } from 'react';
2 import { useParams, useNavigate } from 'react-router-dom';
3 import { campaignAPI, WebSocketService } from '../services/api';
4 import LeadCard from '../components/LeadCard';
5 import EmailPreview from '../components/EmailPreview';
6 import './CampaignView.css';
7
8 const CampaignView = () => {
9   const { threadId } = useParams();
10   const navigate = useNavigate();
11
12   // State
13   const [campaign, setCampaign] = useState(null);
14   const [leads, setLeads] = useState([]);
15   const [emails, setEmails] = useState([]);
16   const [loading, setLoading] = useState(true);
17   const [error, setError] = useState('');
18   const [phase, setPhase] = useState('');
19   const [showEmailApproval, setShowEmailApproval] = useState(false);
20   const [wsService, setWsService] = useState(null);
21
22   useEffect(() => {
23     fetchCampaignData();
24     setupWebSocket();
25
26     return () => {
27       if (wsService) {
28         wsService.disconnect();
29       }
30     };
31   }, [threadId]);
32
33   const setupWebSocket = () => {
34     const service = new WebSocketService();
35     service.connect(threadId);
36
37     service.addListener('message', handleWebSocketMessage);
38     service.addListener('connected', () => {
39       console.log('Connected to campaign updates');
40     });
41
42     setWsService(service);
43   };
44
45   const handleWebSocketMessage = (data) => {
46     console.log('WebSocket update:', data);
47
48     if (data.type === 'campaign_updated' ||
49         data.type === 'campaign_started' ||
50         data.type === 'emails_approved' ||
51         data.type === 'meeting_scheduled') {
52       fetchCampaignData();
53     }
54   };

```

Figure 49: Campaign View

```

New folder > frontend > src > pages > .idea 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    } catch (err) {
79      setError('Failed to fetch campaign data');
80      console.error(err);
81    } finally {
82      setLoading(false);
83    }
84  };
85
86  const handleApproveEmails = async (decision) => {
87    try {
88      await campaignAPI.approveEmails(threadId, decision);
89      setShowEmailApproval(false);
90      if (decision === 'yes') {
91        setPhase('sending');
92      }
93    } catch (err) {
94      setError('Failed to process email approval');
95      console.error(err);
96    }
97  };
98
99  const handleContinue = async () => {
100    try {
101      await campaignAPI.continueCampaign(threadId);
102      fetchCampaignData();
103    } catch (err) {
104      console.error('Failed to continue campaign:', err);
105    }
106  };
107
108  // ... (rest of the component code)

```

Figure 50: Campaign View

```

102
103  {/* Email Approval Modal */}
104  {showEmailApproval && (
105    <div className="modal-overlay">
106      <div className="modal">
107        <div className="modal-header">
108          <h3>Approve Email Sending</h3>
109        </div>
110        <div className="modal-body">
111          <p>
112            {emails.length} emails have been drafted and are ready to send.
113            Review the emails below and approve or reject sending.
114          </p>
115
116          <div className="email-preview-list">
117            {emails.slice(0, 2).map((email, index) => (
118              <EmailPreview key={index} email={email} />
119            ))}
120          </div>
121
122          {emails.length > 2 && (
123            <p className="text-center mt-2">
124              ... and {emails.length - 2} more emails
125            </p>
126          )}
127        </div>
128        <div className="modal-footer">
129          <button
130            onClick={() => handleApproveEmails('yes')}
131            className="btn btn-success">
132            <input checked="" type="checkbox"/> Approve & Send All
133          </button>
134
135          <button
136            onClick={() => handleApproveEmails('no')}
137            className="btn btn-danger">
138            ✖ Reject & Stop
139          </button>
140
141          <button
142            onClick={() => navigate(`/monitoring/${threadId}`)}
143            className="btn btn-secondary">
144            View All Emails
145          </button>
146        </div>
147      </div>
148    )}
149  )}
150
151  {/* Leads Section */}
152  <section className="campaign-section">
153    <div className="section-header">

```

Figure 51: Campaign View

```

New folder > frontend > src > pages > JS CampaignViews > 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
1 import React, { useState, useEffect } from 'react';
2 import { Link } from 'react-router-dom';
3 import { campaignAPI } from '../services/api';
4 import './Dashboard.css';
5
6 const Dashboard = () => {
7   const [threads, setThreads] = useState([]);
8   const [loading, setLoading] = useState(true);
9   const [error, setError] = useState(null);
10
11   useEffect(() => {
12     fetchThreads();
13   }, []);
14
15   const fetchThreads = async () => {
16     try {
17       setLoading(true);
18       const data = await campaignAPI.listThreads();
19       setThreads(data.threads || []);
20     } catch (err) {}
21     setError('Failed to fetch campaigns');
22     console.error(err);
23   } finally {
24     setLoading(false);
25   }
26 };
27
28 const formatDate = (dateString) => {
29   if (!dateString) return 'N/A';
30   return new Date(dateString).toLocaleDateString();
31 };
32
33 return (
34   <div className="dashboard">
35     {/* Hero Section */}
36     <div className="hero-section">
37       <h1 className="hero-title">B2B Lead Generator</h1>
38       <p className="hero-tagline">AI-powered lead generation and outreach automation</p>
39     </div>
40
41     {/* Create Campaign Section */}
42     <div className="section create-campaign-section">
43       <div className="section-content">
44         <h2 className="section-title">Start Your Campaign</h2>
45         <p className="section-description">
46           Begin your B2B outreach journey with our AI-powered platform. Generate qualified leads,
47           craft personalized emails, and automate follow-ups.
48         </p>
49         <Link to="/campaign/new" className="btn btn-primary btn-large">
50           Create New Campaign
51         </Link>
52       </div>
53     </div>
54   </div>
```

Figure 53: Dashboard

```

New folder > frontend > src > pages > Dashboard > Dashboard > fetchThreads
6  const Dashboard = () => {
55    /* How It Works Section - No Title, Single Row */
56    <div className="how-it-works-section">
57      <div className="steps-container">
58        <div className="step">
59          <div className="step-number">1</div>
60          <h3 className="step-title">Define Target</h3>
61          <p className="step-description">
62            Specify companies with a simple search query
63          </p>
64        </div>
65        <div className="step">
66          <div className="step-number">2</div>
67          <h3 className="step-title">AI Research</h3>
68          <p className="step-description">
69            AI finds decision-makers & qualifies leads
70          </p>
71        </div>
72        <div className="step">
73          <div className="step-number">3</div>
74          <h3 className="step-title">Personalized Outreach</h3>
75          <p className="step-description">
76            AI crafts & sends personalized emails
77          </p>
78        </div>
79        <div className="step">
80          <div className="step-number">4</div>
81          <h3 className="step-title">Monitor & Engage</h3>
82          <p className="step-description">
83            Track replies & automate follow-ups
84          </p>
85        </div>
86      </div>
87    </div>
88
89    /* Our Aim Section */
90    <div className="section-aim-section">
91      <h2 className="section-title">Our Aim</h2>
92      <p className="aim-description">
93        We aim to democratize B2B lead generation by making it accessible, efficient, and
94        effective for businesses of all sizes. Our platform combines AI intelligence with
95        human oversight to create meaningful business connections.
96      </p>
97    </div>
98
99    /* Campaigns Dashboard */
100    <div className="section-campaigns-section">
101      <div className="section-header">
102        <h2 className="section-title">Your Campaigns</h2>
103        <button onClick={fetchThreads} className="btn btn-secondary">
104          Refresh
105        </button>
106      </div>

```

Figure 54: Dashboard

```

108    <error 88 {
109      <div className="alert alert-error">
110        {error}
111      </div>
112    }
113
114    <loading ? {
115      <div className="loading">
116        <div className="spinner"></div>
117        <p>Loading campaigns...</p>
118      </div>
119    } : threads.length === 0 ? {
120      <div className="empty-state">
121        <div className="empty-state-icon"></div>
122        <h3>Campaigns</h3>
123        <p>Start your first campaign to begin generating leads</p>
124      </div>
125    } : {
126      <div className="campaigns-grid">
127        {threads.map((thread) => {
128          <div key={thread.id} className="campaign-card">
129            <div className="campaign-card-header">
130              <h3>{thread.name || 'Untitled Campaign'}</h3>
131              <span className="status-badge status-${thread.status}">
132                {thread.status || 'unknown'}
133              </span>
134            </div>
135
136            <div className="campaign-card-body">
137              <div className="campaign-info">
138                <div className="info-item">
139                  <span className="info-label">Query:</span>
140                  <span className="info-value">{thread.query || 'N/A'}</span>
141                </div>
142                <div className="info-item">
143                  <span className="info-label">Leads Found:</span>
144                  <span className="info-value">{thread.leads_count || 0}</span>
145                </div>
146                <div className="info-item">
147                  <span className="info-label">Emails Sent:</span>
148                  <span className="info-value">{thread.emails_sent || 0}</span>
149                </div>
150                <div className="info-item">
151                  <span className="info-label">Created:</span>
152                  <span className="info-value">{formatDate(thread.created_at)}</span>
153                </div>
154              </div>
155
156              <div className="campaign-actions">
157                <Link
158                  to={`/campaign/${thread.id}`}
159                  className="btn btn-secondary btn-sm"
160                >

```

Figure 55: Dashboard

```

153     </div>
154   </div>
155
156   <div className="campaign-actions">
157     <Link>
158       to={` /campaign/${thread.id}`}
159       className="btn btn-secondary btn-sm"
160     >
161       View Details
162     </Link>
163     <Link>
164       to={` /monitoring/${thread.id}`}
165       className="btn btn-primary btn-sm"
166     >
167       Monitor
168     </Link>
169   </div>
170 </div>
171 </div>
172 ))}
173 </div>
174 )}
175 </div>
176
177 /* Contact Us Section */
178 <div className="section contact-section">
179   <h2 className="contact-title">Contact Us</h2>
180   <p className="contact-description">
181     Have questions or need assistance? Reach out to our team.
182   </p>
183   <div className="contact-info">
184     <p className="contact-email">Email: support@b2bleadgenerator.com</p>
185     <p className="contact-hours">Support Hours: Mon-Fri, 9AM-6PM EST</p>
186   </div>
187 </div>
188 </div>
189 );
190 };
191
192 export default Dashboard;

```

Figure 56: Dashboard

```

1  import React from 'react';
2  import './LeadCard.css';
3
4  const LeadCard = ({ lead }) => {
5    const getScoreColor = (score) => {
6      if (score >= 80) return '#2ecc71';
7      if (score >= 60) return '#f39c12';
8      return '#e74c3c';
9    };
10
11    const getIndustryIcon = (industry) => {
12      const icons = {
13        'ai': '🤖',
14        'saas': '💻',
15        'fintech': '💰',
16        'ecommerce': '🛒',
17        'health-care': '🏥',
18        'other': '🏠',
19        'unknown': '❓'
20      };
21      return icons[industry] || icons.unknown;
22    };
23
24    const getCompanySizeLabel = (size) => {
25      const labels = {
26        'small': 'Small (<50)',
27        'medium': 'Medium (50-250)',
28        'large': 'Large (250-1000)',
29        'enterprise': 'Enterprise (>1000)',
30        'unknown': 'Unknown'
31      };
32      return labels[size] || size;
33    };
34
35    return (
36      <div className="lead-card">
37        <div className="lead-card-header">
38          <div className="lead-title">
39            <span className="lead-icon">{getIndustryIcon(lead.industry)}</span>
40            <h3>{lead.company_name}</h3>
41          </div>
42
43          <div className="lead-score" style={{ color: getScoreColor(lead.research_confidence * 100) }}>
44            {Math.round(lead.research_confidence * 100)}%
45          </div>
46        </div>
47
48        <div className="lead-card-body">
49          { /* Basic Info */ }
50          <div className="lead-info">
51            <div className="info-row">
52              <span className="info-label">Website:</span>
53              <a
54                href={lead.company_website}
55                target="_blank"

```

Figure 57: Dashboard

```

New Folder > Frontend > src > components > # LeadCard.js > @ LeadCard > @ getIndustryIcon > @ kom > @ ai
4  const LeadCard = ({ lead }) => {
55      target="_blank"
56      rel="noopener noreferrer"
57      className="info-value link"
58      >
59      {lead.domain}
60      </a>
61      </div>
62
63      <div className="info-row">
64        <span className="info-label">Industry:</span>
65        <span className="info-value">
66          <span className="badge">{lead.industry}</span>
67        </span>
68      </div>
69
70      <div className="info-row">
71        <span className="info-label">Size:</span>
72        <span className="info-value">{getCompanySizeLabel(lead.company_size)}</span>
73      </div>
74
75      {lead.qualification_score !== undefined && (
76        <div className="info-row">
77          <span className="info-label">Qualification:</span>
78          <span className="info-value">
79            <span
80              className="badge"
81              style={{
82                backgroundColor: getScoreColor(lead.qualification_score),
83                color: 'white'
84              }}
85            >
86              {lead.qualification_score} pts
87            </span>
88            {lead.qualified && <span className="qualified-badge">✓ Qualified</span>}
89          </span>
90        </div>
91      )}
92    </div>
93
94    {/* Decision Makers */}
95    {lead.decision_makers && lead.decision_makers.length > 0 && (
96      <div className="lead-section">
97        <div className="section-label">Decision Makers:</div>
98        <div className="tags">
99          {lead.decision_makers.map((role, idx) => (
100            <span key={idx} className="tag">{role}</span>
101          ))}
102        </div>
103      </div>
104    )}
105
106    {/* Intent Signals */}
107    {lead.intent_signals && lead.intent_signals.length > 0 && (
108      <div className="lead-section">

```

Figure 58: Dashboard

```

121    </div>
122  })
123
124  {/* Pain Points */}
125  {lead.pain_points && lead.pain_points.length > 0 && (
126    <div className="lead-section">
127      <div className="section-label">Pain Points:</div>
128      <div className="tags">
129        {lead.pain_points.map((point, idx) => (
130          <span key={idx} className="tag tag-warning">{point}</span>
131        ))}
132      </div>
133    </div>
134  )}
135
136  {/* Emails */}
137  {lead.validated_emails && lead.validated_emails.length > 0 && (
138    <div className="lead-section">
139      <div className="section-label">Emails:</div>
140      <div className="emails">
141        {lead.validated_emails.map((email, idx) => (
142          <div key={idx} className="email">
143            <span className="email-address">{email}</span>
144            <span className="email-quality">{lead.email_quality}</span>
145          </div>
146        ))}
147      </div>
148    </div>
149  )}
150
151  {/* Website Summary */}
152  {lead.website_summary && (
153    <div className="lead-section">
154      <div className="section-label">Summary:</div>
155      <div className="summary-text">
156        {lead.website_summary}
157      </div>
158    </div>
159  )}
160
161  </div>
162
163  <div className="lead-card-footer">
164    <div className="source-badge">
165      Source: {lead.source || 'web'}
166    </div>
167  </div>
168  </div>
169  );
170  };
171
172  export default LeadCard;

```

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.

```

New folder > frontend > src > components > # EmailPreview.js | default
1 import React, { useState } from 'react';
2 import './EmailPreview.css';
3
4 const EmailPreview = ({ email }) => {
5   const [expanded, setExpanded] = useState(false);
6
7   return (
8     <div className="email-preview ${email.sent ? 'sent' : 'drafted'}">
9       <div className="email-preview-header" onClick={() => setExpanded(!expanded)}>
10        <div className="email-info">
11          <div className="email-recipient">
12            <strong>To:</strong> {email.email}
13          </div>
14          <div className="email-company">
15            <strong>Company:</strong> {email.company_name}
16          </div>
17        </div>
18        <div className="email-status">
19          {email.sent ? (
20            <span className="status-badge sent-badge">
21              ✓ Sent {email.sent_at ? new Date(email.sent_at).toLocaleDateString() : ''}
22            </span>
23          ) : (
24            <span className="status-badge draft-badge"> 📧 Drafted</span>
25          )}
26          <button className="expand-btn">
27            {expanded ? '▲' : '▼'}
28          </button>
29        </div>
30      </div>
31      <div className="email-subject">
32        <strong>Subject:</strong> {email.email_subject}
33      </div>
34      {expanded && (
35        <div className="email-body-preview">
36          <div className="email-body-header">Email Body:</div>
37          <div className="email-body-content">
38            {email.email_body.split('\n').map((line, idx) => (
39              <div key={idx} className="email-body-line">
40                {line || <br />}
41              </div>
42            ))}
43          </div>
44        </div>
45      )}
46    </div>
47  );
48
49  </div>
50);
51
52
53 export default EmailPreview;
  
```

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.

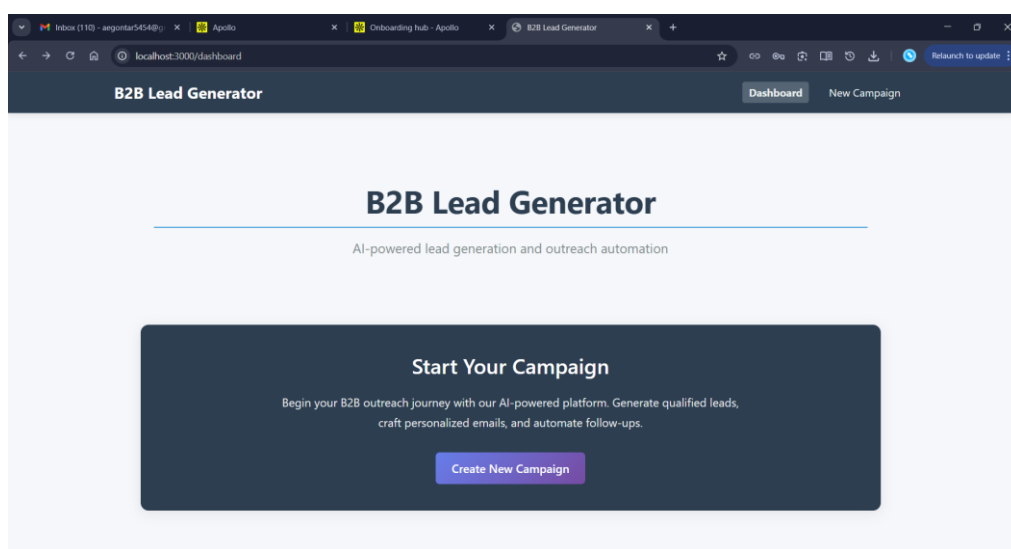


Figure 61: Home Page

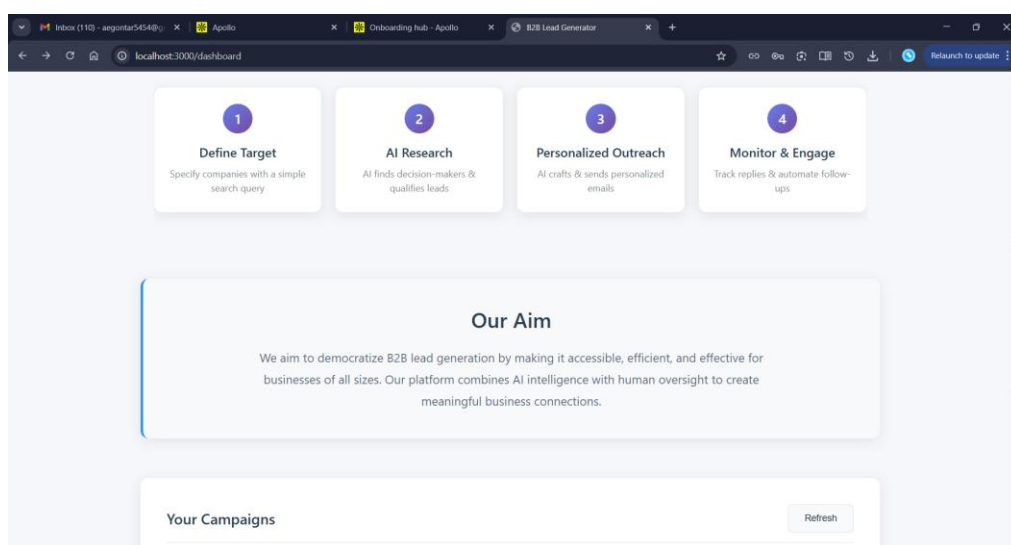


Figure 62: About Section

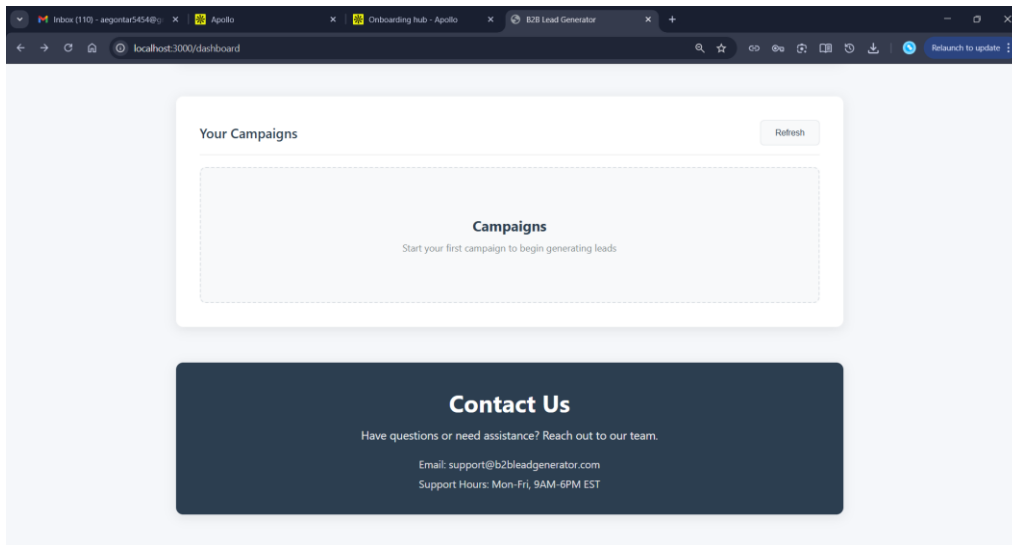


Figure 63: Campaign History

- 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

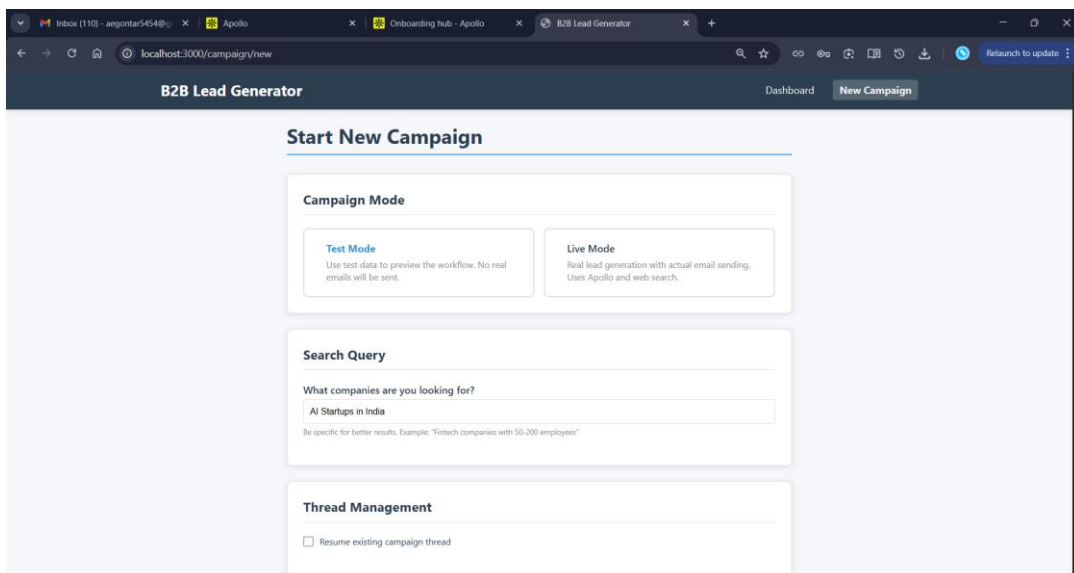
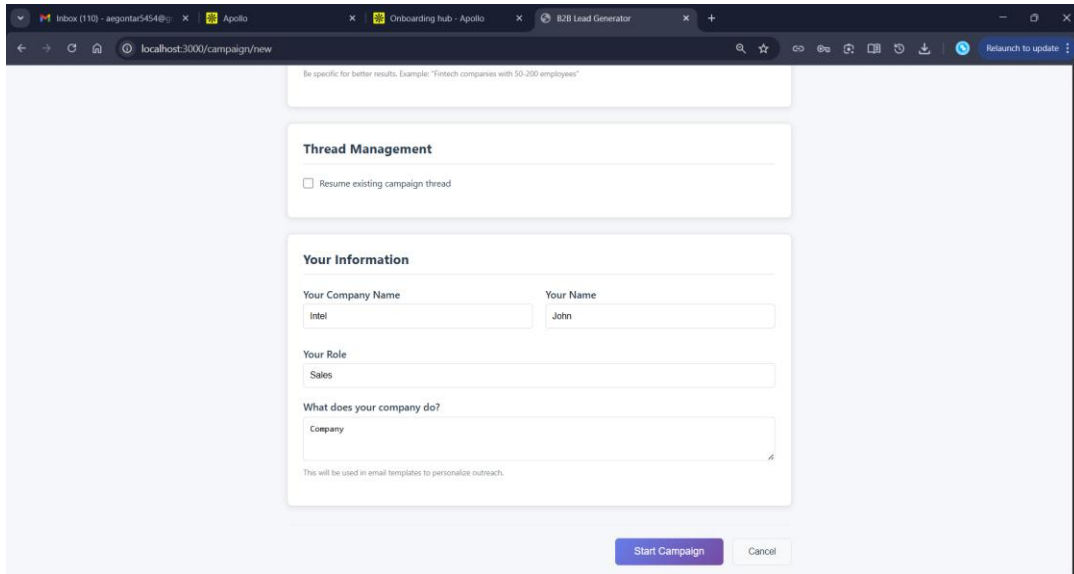


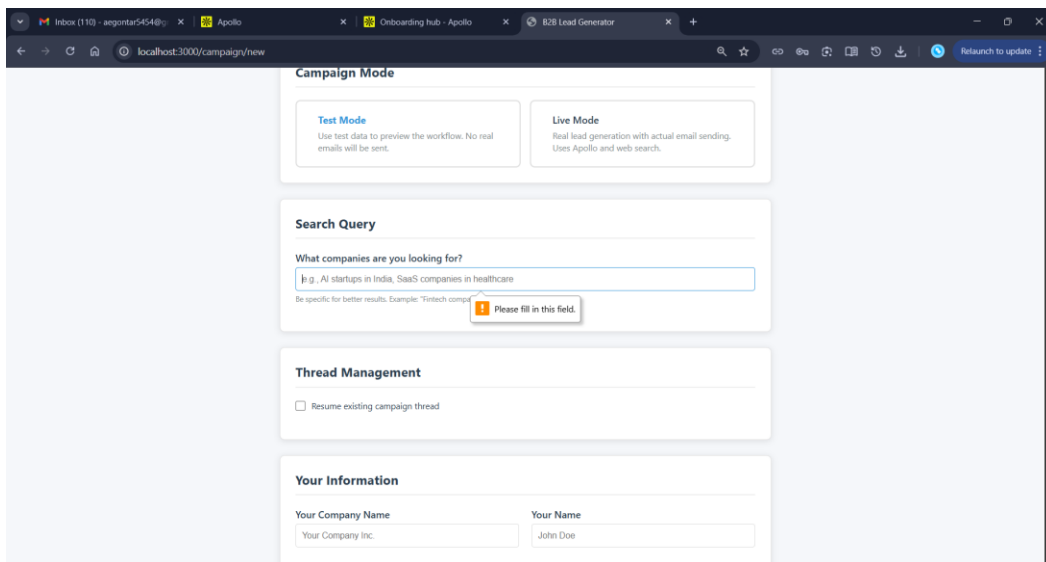
Figure 64: Entering the user input



The screenshot shows a web browser window with the URL `localhost:3000/campaign/new`. The page contains a form for creating a new campaign. At the top, there is a text input field with a placeholder: "Be specific for better results. Example: 'Fintech companies with 50-200 employees'". Below this is a section titled "Thread Management" with a checkbox labeled "Resume existing campaign thread". The next section is "Your Information", which includes two columns of input fields. The left column has "Your Company Name" (with "Intel" entered) and "What does your company do?" (with "Company" entered). The right column has "Your Name" (with "John" entered) and "Your Role" (with "Sales" entered). A small note at the bottom of the "Your Information" section states: "This will be used in email templates to personalize outreach." At the bottom right of the form are two buttons: "Start Campaign" (in purple) and "Cancel" (in light gray).

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



The screenshot shows the same web browser window, but now the form is in "Campaign Mode". At the top, there are two tabs: "Test Mode" (selected) and "Live Mode". Below the tabs is a "Search Query" section with a text input field. The input field contains the text "i.g., AI startups in India, SaaS companies in healthcare". A red error message "Please fill in this field." is displayed below the input field. Below the "Search Query" section is the "Thread Management" section with the "Resume existing campaign thread" checkbox. At the bottom is the "Your Information" section, which now shows "Your Company Name" with "Your Company Inc." entered and "Your Name" with "John Doe" entered. The "Start Campaign" and "Cancel" buttons are still present at the bottom right.

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

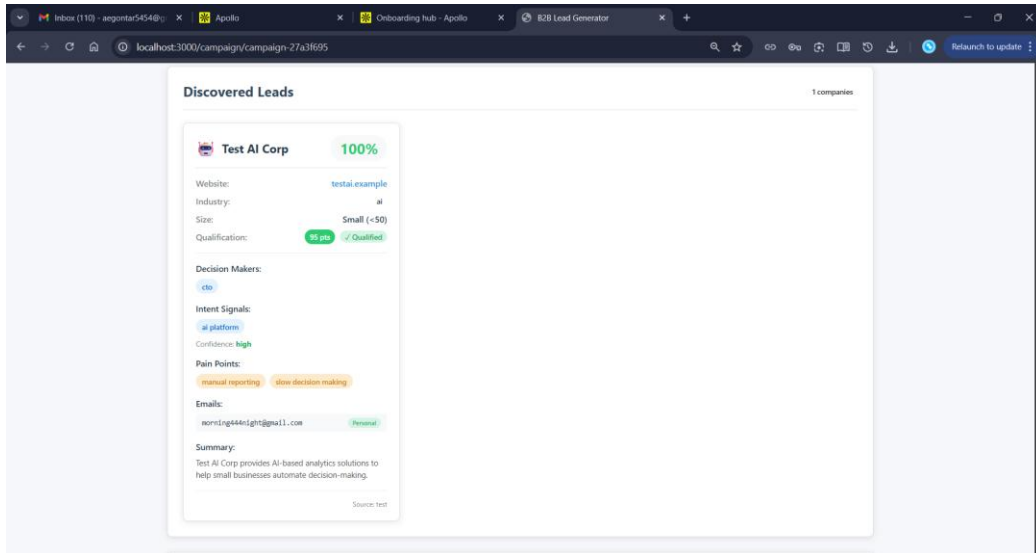


Figure 67: Lead Boards

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

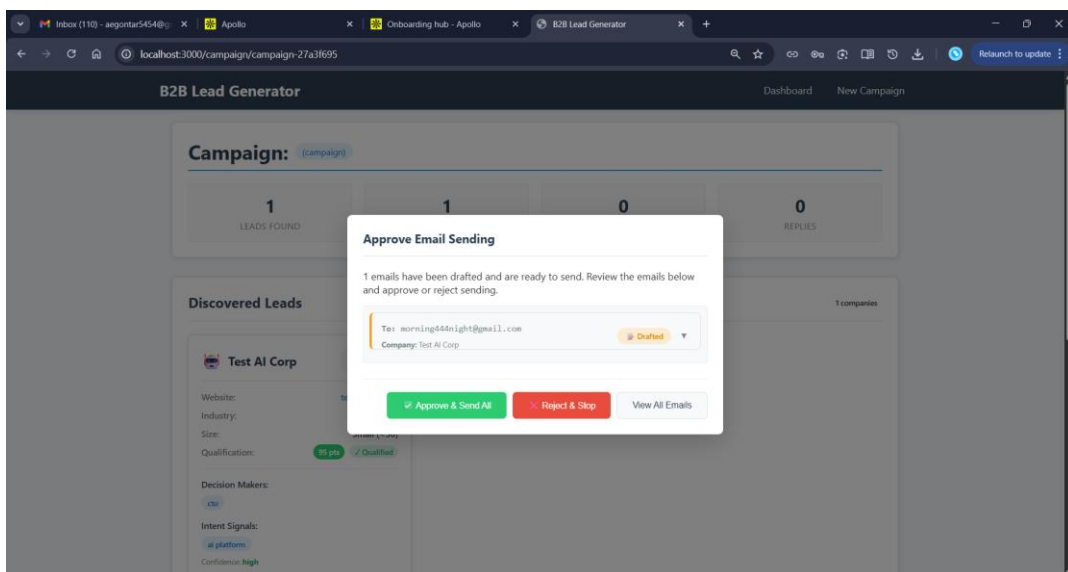


Figure 68: Approve email

- **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

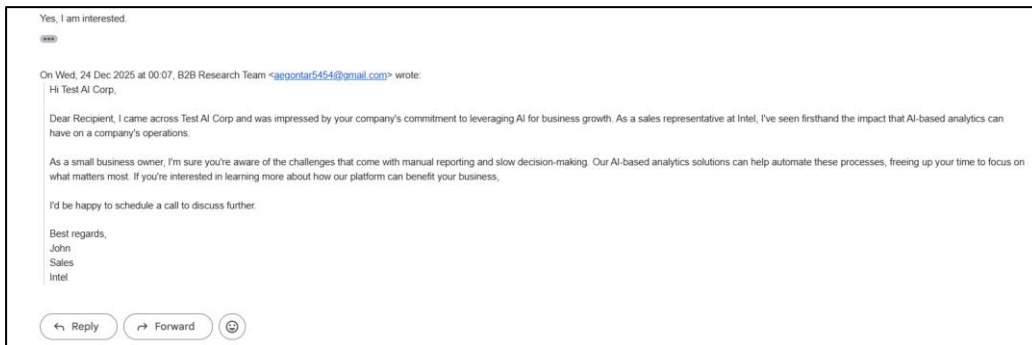


Figure 69: Email form inbox

- **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

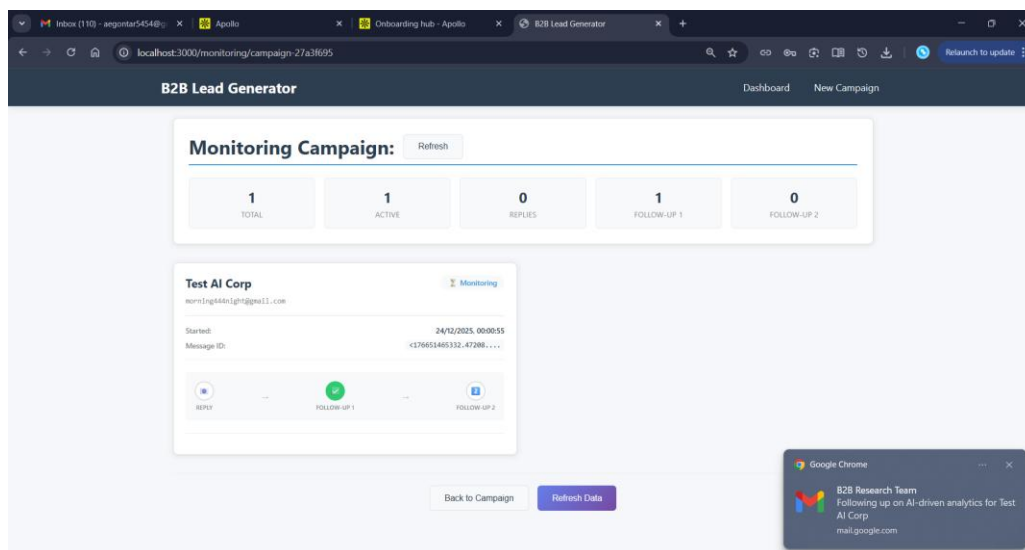


Figure 70: Tracking Progress

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

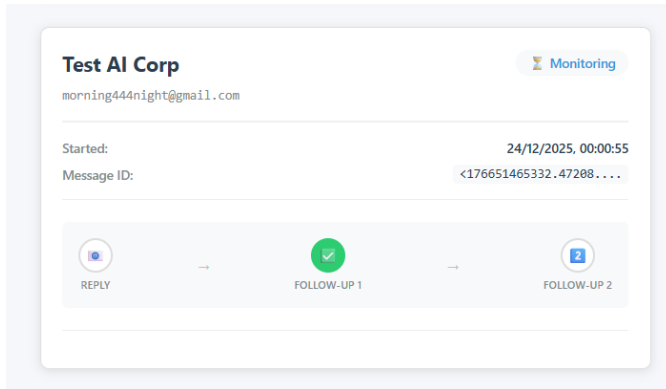


Figure 71: Monitor Board

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

```
(main) PS C:\Users\jeorgi\Desktop\New folder (5)\New folder\backend> python m.py
INFO: 127.0.0.1:49158 - "GET /api/threads HTTP/1.1" 200 OK
INFO: 127.0.0.1:165252 - "POST /api/campaign/start HTTP/1.1" 200 OK
INFO: 127.0.0.1:165252 - "GET /api/campaign/campaign-7309fdb6/status HTTP/1.1" 200 OK
INFO: 127.0.0.1:160435 - "WebSocket /ws/campaign-7309fdb6" [accepted]
INFO: 127.0.0.1:165252 - "GET /api/campaign/campaign-7309fdb6/status HTTP/1.1" 200 OK
INFO: 127.0.0.1:161468 - "WebSocket /ws/campaign-7309fdb6" [accepted]
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/leads HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/leads HTTP/1.1" 200 OK
INFO: 127.0.0.1:165252 - "GET /api/campaign/campaign-7309fdb6/emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:165252 - "GET /api/campaign/campaign-7309fdb6/emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:165252 - "OPTIONS /api/campaign/campaign-7309fdb6/approve-emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "POST /api/campaign/campaign-7309fdb6/approve-emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/status HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/status HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/leads HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/leads HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:152129 - "WebSocket /ws/campaign-7309fdb6" [accepted]
INFO: 127.0.0.1:160432 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:16187 - "WebSocket /ws/campaign-7309fdb6" [accepted]
INFO: 127.0.0.1:163541 - "OPTIONS /api/campaign/campaign-7309fdb6/schedule-meeting HTTP/1.1" 200 OK
INFO: 127.0.0.1:163541 - "POST /api/campaign/campaign-7309fdb6/schedule-meeting HTTP/1.1" 200 OK
INFO: 127.0.0.1:163541 - "GET /api/campaign/campaign-7309fdb6/status HTTP/1.1" 200 OK
INFO: 127.0.0.1:163541 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:163541 - "GET /api/campaign/campaign-7309fdb6/status HTTP/1.1" 200 OK
INFO: 127.0.0.1:163541 - "GET /api/campaign/campaign-7309fdb6/leads HTTP/1.1" 200 OK
INFO: 127.0.0.1:163541 - "GET /api/campaign/campaign-7309fdb6/leads HTTP/1.1" 200 OK
INFO: 127.0.0.1:49666 - "GET /api/campaign/campaign-7309fdb6/emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:49666 - "GET /api/campaign/campaign-7309fdb6/emails HTTP/1.1" 200 OK
INFO: 127.0.0.1:152974 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:152974 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:152974 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:152974 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
INFO: 127.0.0.1:152974 - "GET /api/campaign/campaign-7309fdb6/monitoring HTTP/1.1" 200 OK
```

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

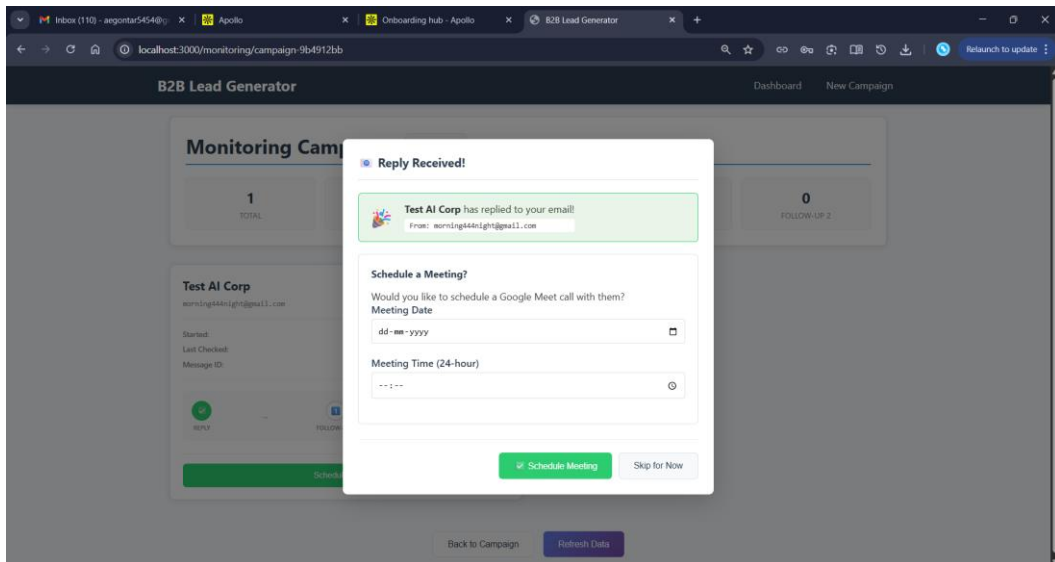


Figure 73: Setting time and Date

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

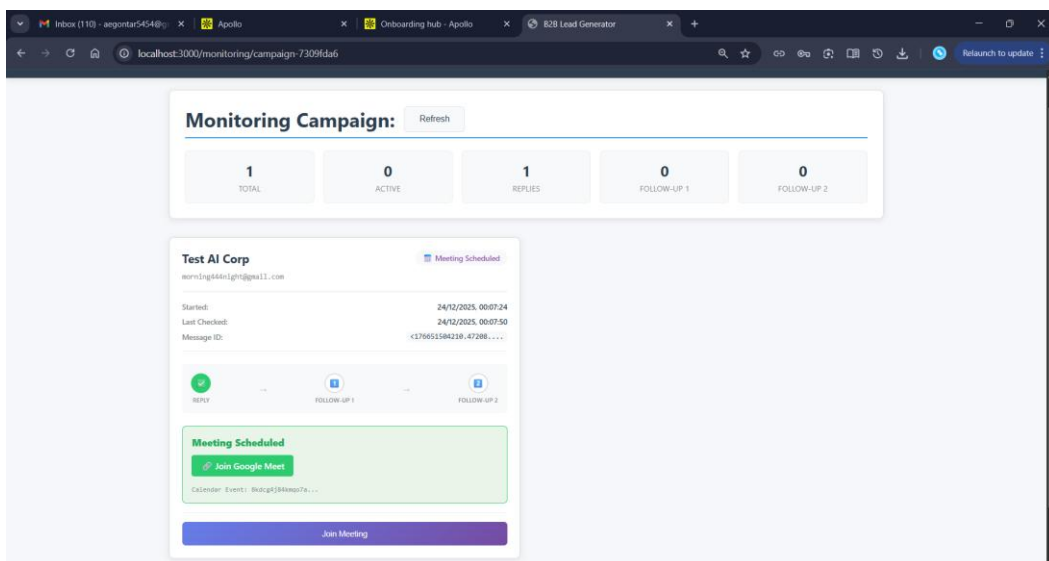


Figure 74: Meeting is scheduled

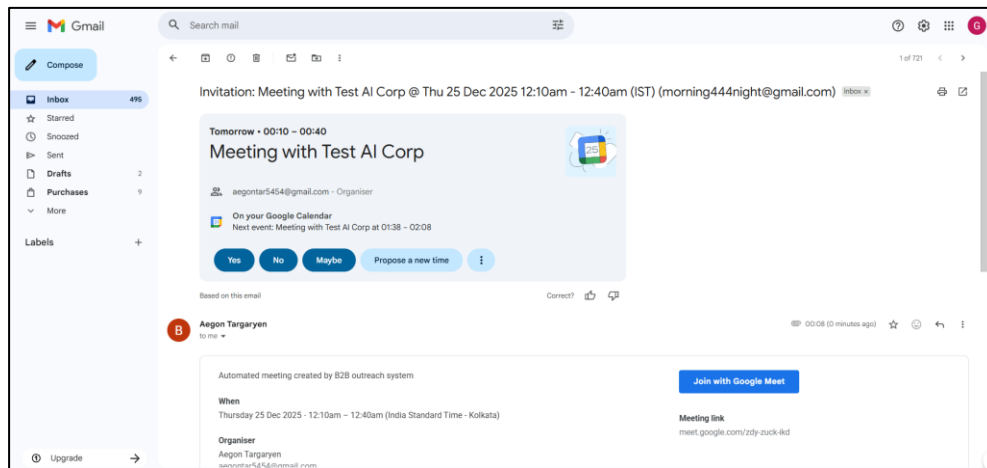


Figure 75: Email conformation

- 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.