*Project Title*: **HireBOT**

**Team Members:**

1. Boomika S
2. Sakthi Gowshick S
3. Sanjaykumar S
4. Sharmila M
5. Subhiksha Kodibass

# PROBLEM STATEMENT

# Currently, college placement teams manually send emails to multiple HRs, track responses, and follow up individually. This process is time-consuming, error-prone, and lacks consistency in communication tone. Moreover, tracking unreplied or partially replied mails requires significant manual effort. Hence, there is a need for a smart, automated system that can manage end-to-end communication, track reply statuses, and handle responses intelligently.

# ABSTRACT

The project **HireBOT** aims to automate the process of communication between a college placement cell and multiple HR representatives from various companies.   
The system automatically sends invitation mails to companies with final-year student details categorized by department. It monitors email responses, segregates them into replied and unreplied, and sends appropriate follow-up or acknowledgment mails until the company provides a job description (JD).   
By integrating AI with the Model Context Protocol (MCP), **HireBOT** intelligently classifies and prepares email responses, but dispatches them only after human (user) review and approval, ensuring accuracy and control in automated communication.

# OBJECTIVE

To design and develop an automated email communication system that can efficiently interact with HRs of various companies for campus placement coordination, utilizing Large Language Models (LLMs) integrated with MCP for intelligent processing, contextual reply generation, and continuous follow-up until successful placement collaboration.

# SYSTEM ARCHITECTURE

The **HireBOT** system consists of the following key layers:  
1. **Frontend (React.js):** Dashboard for managing HR contacts, viewing mail status, and monitoring JD collection progress.  
2. **Backend (FastAPI):** Core logic for sending/receiving mails, handling authentication, and storing data.  
3. **Database (Supabase):** Stores HR details, student data, and mail logs.  
4. **LLM Layer (Llama 3.1):** Processes and classifies email content, extracts JD details, and generates context-aware replies.  
5. **MCP Layer:** Bridges the LLM with backend functions, enforcing structured responses and safe tool access.  
6. **Mail Integration:** Uses Mailjet for sending and receiving mail threads.

# WORKFLOW

1. Placement team uploads student and HR data.  
2. System automatically sends invitation emails with student details.  
3. Incoming mails are analyzed by the LLM through MCP to classify as replied, unreplied, or JD received.  
4. MCP triggers automated actions: sending thank-you replies, scheduling follow-ups, or updating status.  
5. The process repeats periodically until JD is received.  
6. Dashboard displays progress metrics and communication history.

# TECHNOLOGY STACK

- **Frontend:** React.js, Tailwind CSS  
**-** **Backend:** FastAPI (Python)  
**- Database:** Supabase  
**- LLM:** Llama 3.1  
**- AI Orchestration:** Model Context Protocol (MCP)  
**- Mail APIs:** Mailjet API  
**- Deployment:** AWS

# IMPLEMENTATION PLAN

Phase 1: Requirement Analysis & Design  
Phase 2: Backend and Mail API Integration  
Phase 3: LLM + MCP Integration  
Phase 4: Frontend Dashboard Development  
Phase 5: Testing, Optimization & Deployment

# FUTURE SCOPE

- Integration with LinkedIn or job portals for direct HR outreach.  
- AI-based JD parsing to automatically match students to suitable openings.  
- Voice assistant integration for placement officers.

# REFERENCES

1. Anthropic Claude API Documentation.  
2. Model Context Protocol (MCP) Integration Guide.  
3. Gmail API Developer Documentation.  
4. FastAPI and MongoDB Integration Tutorials.