Section 1: System Architecture (Detailed and Specific)

This section defines the structure of Orgo’s communication platform, focusing on its components, their interactions, and the workflow logic. It provides a precise guide for implementing each module, ensuring modularity, offline capability, and role-based routing.

---

1.1 Purpose of System Architecture

Objective: To create a structured, secure, and efficient platform for email-based communication workflows.

Outcome:

A system capable of routing emails based on predefined rules.

Modular design supporting role-based tasks, offline operations, and sensitive workflows.

---

1.2 High-Level Architecture Diagram

Diagram Description: A visual flowchart showing data flow from email reception to task completion:

[Email Server (SMTP/IMAP)] → [Email Parser] → [Rule Engine] → [Database] → [Action/Response System]

↓

[Offline Sync Module (Optional)]

---

1.3 Component Breakdown

1. Email Server:

Role: Handles incoming and outgoing emails using standard protocols.

Protocols:

SMTP for sending emails.

IMAP/POP3 for retrieving emails.

Integration:

Connects with the organization’s existing email infrastructure.

Supports secure transmission via TLS.

2. Email Parser:

Role: Processes incoming emails to extract actionable data.

Functions:

Extract metadata:

Sender (e.g., secretary@organization.com).

Recipient (e.g., emergency@organization.com).

Subject line (e.g., "Water Leak in Office 102").

Identify keywords (e.g., "leak," "urgent").

Detect attachments and store them for later use.

Technology:

Python libraries: imaplib, smtplib, email.

3. Rule Engine:

Role: Applies predefined routing and escalation rules to parsed emails.

Functions:

Matches email content to routing rules defined in YAML/JSON.

Dynamically attaches context-specific documents.

Escalates tasks if unresolved within a defined timeframe.

Example Rule:

- condition: "subject contains 'leak'"

action:

route\_to: "maintenance@organization.com"

attach:

- "location\_map.pdf"

- "leak\_protocol.pdf"

escalate\_after: "2 hours"

Technology:

Python: PyYAML for rule parsing.

4. Database:

Role: Stores all data related to workflows, logs, and configurations.

Types:

PostgreSQL for scalable deployments.

SQLite for offline environments.

Data Stored:

Workflow rules.

Logs (e.g., email routing, task statuses).

Documents (e.g., protocols, templates).

5. Action/Response System:

Role: Executes responses or triggers follow-up actions.

Functions:

Sends automated replies or notifications.

Updates workflow statuses in the database.

Routes replies (e.g., technician ETA) to appropriate modules for further action.

Technology:

Python libraries: smtplib, json.

6. Offline Sync Module:

Role: Ensures functionality without internet connectivity.

Functions:

Processes .pst or .mbox files for local synchronization.

Stores data in SQLite and syncs with PostgreSQL when connectivity is restored.

Technology:

Tools: py-outlook for .pst handling, rclone for manual file transfers.

---

1.4 Communication Flow

A detailed step-by-step process for email handling:

1. Email Reception:

An email is sent to emergency@organization.com via the organization’s email server.

The email server stores the email and forwards it to Orgo.

2. Parsing:

Orgo’s parser extracts:

Subject: "Water Leak in Office 102."

Sender: secretary@organization.com.

Keywords: "leak."

Attachments are identified and stored.

3. Rule Application:

The rule engine applies a predefined rule:

Routes the email to maintenance@organization.com.

Attaches a map and protocol for leak resolution.

4. Action Trigger:

Maintenance staff receives the email and replies with:

"ETA: 30 minutes" (parsed and logged by Orgo).

If unresolved in 2 hours, the task is escalated to supervisor@organization.com.

5. Resolution:

Technician completes the task and replies.

Orgo logs the resolution and sends a summary to emergency@organization.com.

---

1.5 Features and Benefits

1. Modular Design:

Components (parser, rule engine, database) operate independently for easy upgrades.

New modules (e.g., sector-specific workflows) can be added without disrupting core functionality.

2. Offline Capability:

Emails are processed locally during outages.

Data is synchronized when connectivity is restored.

3. Scalability:

PostgreSQL supports millions of emails and complex workflows for large organizations.

Redis or RabbitMQ can handle high-volume task queuing.

4. Security:

End-to-end TLS encryption protects email data.

RBAC ensures access control for sensitive workflows.

---

1.6 Deliverables

1. Architecture Diagram:

A visual map of Orgo’s components and their interactions.

2. Component Details:

In-depth descriptions of the email parser, rule engine, database, and more.

3. Workflow Examples:

Specific communication flows for maintenance, sensitive reporting, and general workflows.

---

Summary

This section defines Orgo’s architecture with precise details about its components, their functions, and their interactions. By outlining the flow of information, modular design, and offline capabilities, it establishes a solid foundation for building a reliable and scalable platform tailored to organizational needs.