### Orgo v2 Blueprint Section 2 - Functional Specifications

#### Section 2: Functional Specifications

This section defines the core functionalities of Orgo, focusing on dynamic workflows, modular components, and metadata-driven task management. The updates replace domain-specific workflows with generalized workflows that adapt dynamically to organizational needs using metadata attributes.

### 2.1 Purpose of Functional Specifications

Objective: To ensure Orgo’s functionalities meet diverse organizational needs through flexibility and adaptability.

Outcome:

* A metadata-driven approach to workflows, reducing domain-specific dependencies.
* Dynamic task handling based on attributes like type, priority, and metadata.
* Streamlined routing and execution processes.

### 2.2 Core Functionalities

1. Email Parsing:
   * Orgo extracts essential information from incoming emails, such as:
     + Subject line, sender, recipient, body content, and attachments.
   * This data is stored in structured formats for easy integration with workflows.

Example:

* + Email Subject: "Urgent: HVAC Repair Needed"
  + Extracted Metadata:
    - Sender: [maintenance@organization.com](mailto:maintenance@organization.com)
    - Recipient: [support@organization.com](mailto:support@organization.com)
    - Keywords: ["urgent", "HVAC", "repair"]

1. Dynamic Task Management:
   * Tasks are created dynamically using metadata attributes:
     + type: Defines the category (e.g., maintenance, HR).
     + metadata: Stores custom details (e.g., "subtype": "plumbing").
     + status: Tracks progress (e.g., pending, in progress, completed).
   * A unified task handler processes all tasks by interpreting their attributes.
2. Role-Based Routing:
   * Emails and tasks are routed dynamically based on their metadata.
   * Keywords, sender roles, or predefined rules determine the destination.

Example:

* + Keywords like "urgent" and "HVAC" route the email to the facilities team.

1. Attachment and Template Management:
   * Automatically attaches relevant documents and templates to tasks.
   * Templates include preformatted responses or workflows.

Example:

* + For an HVAC issue, Orgo attaches:
    - "HVAC Manual.pdf"
    - "Incident Report Template.docx"

1. Feedback Loop Integration:
   * Orgo dynamically updates workflows based on replies or task actions.
   * Example:
     + A technician replies, "ETA: 30 minutes." Orgo logs the update.
2. Workflow Escalation:
   * Escalates unresolved tasks based on time limits or priorities.
   * Example:
     + If no action occurs within 2 hours, escalate to the department head.
3. Sensitive Data Anonymization:
   * Ensures privacy for sensitive workflows by anonymizing data.
   * Example:
     + Replace Sender: employee@company.com with Sender: Anon1.
4. Offline Processing:
   * Handles tasks offline using SQLite and syncs with PostgreSQL when online.
   * Ensures continuous functionality in remote or disconnected environments.
5. Logging and Audit Trails:
   * Tracks all actions for compliance and transparency.
   * Logs include routing decisions, escalations, and task updates.

Example:

* + Log Entry:
    - Email ID: 12345
    - Sender: [maintenance@organization.com](mailto:maintenance@organization.com)
    - Keywords: ["urgent", "leak"]
    - Routed To: [facilities@organization.com](mailto:facilities@organization.com)
    - Status: Completed

### 2.3 Workflow Examples

1. General Maintenance Workflow:
   * Trigger: Email received about a maintenance issue.
   * Steps:
     1. Email is parsed for metadata and keywords.
     2. Orgo creates a task with type=maintenance and metadata={"subtype": "HVAC"}.
     3. Task is routed dynamically to the facilities team.
     4. Templates and relevant documents are attached.
     5. Updates are logged, and unresolved tasks escalate automatically.
2. HR Issue Reporting Workflow:
   * Trigger: Employee emails HR about a conflict.
   * Steps:
     1. Email is anonymized, and identifying metadata is stripped.
     2. Orgo creates a task with type=HR and metadata={"subtype": "conflict\_resolution"}.
     3. Task is routed to HR and assigned priority based on keywords.
     4. Updates are tracked in the system for compliance.
3. Offline Operations Workflow:
   * Trigger: Organization operates without internet connectivity.
   * Steps:
     1. Incoming emails are stored locally.
     2. Tasks are created and tracked using SQLite.
     3. Once connectivity is restored, tasks and updates sync with PostgreSQL.

### 2.4 Modular Functionalities

Orgo’s modular architecture supports flexibility by enabling organizations to implement only the features they need. Each module leverages the generalized task handler to adapt dynamically.

1. Maintenance Module:
   * Handles incident reporting, document attachment, and escalations.
2. HR Module:
   * Manages workflows like harassment reporting and employee grievances.
3. Education Module:
   * Facilitates parent-teacher communication and incident tracking.
4. Government Module:
   * Supports crisis management, resource allocation, and inter-departmental communication.

### 2.5 Key Benefits

1. Dynamic Adaptability:
   * Task handling is based on metadata, making it versatile and scalable.
2. Efficiency:
   * Automates routing, escalations, and updates, reducing manual effort.
3. Reliability:
   * Offline capabilities ensure continuity in disconnected environments.
4. Privacy:
   * Protects sensitive workflows with anonymization protocols.
5. Scalability:
   * Adapts to organizational growth and diverse industries.

### 2.6 Deliverables

1. Generalized workflow templates for task creation, routing, and execution.
2. Metadata-driven task definitions enabling dynamic workflows.
3. Updated routing rules and escalation logic using YAML/JSON.
4. Guidelines for implementing modular functionalities.

### Summary

This section redefines Orgo’s functional specifications to focus on metadata-driven workflows and generalized task management. By consolidating task handling into a unified system, Orgo ensures efficiency, scalability, and adaptability across diverse organizational needs. The updates eliminate domain-specific dependencies while maintaining the modular flexibility needed for specialized use cases.