### **Rewritten Implementation Blueprint for Orgo**

This updated blueprint incorporates corrections to detail cross-module dependencies, enhance integration workflows, and support organization-specific configurations.

### **Implementation Blueprint for Orgo**

#### **Purpose**

The Implementation Blueprint outlines a detailed roadmap for designing, deploying, and maintaining Orgo. This updated version enhances modularity, scalability, and cross-module integration, emphasizing workflows that align Infrastructure, Core Services, and Interfaces.

### **Key Elements of the Implementation Blueprint**

#### **1. System Architecture**

* **High-Level Design**:
  + Modular architecture supporting:
    - Email-based workflows.
    - Offline capabilities.
    - Organization-specific dynamic configurations.
* **Key Components**:
  + **Email Servers**: Manage email-based task submissions.
  + **Parsing Module**: Extracts actionable data from emails.
  + **Rule Engine**: Dynamically loads routing and escalation rules.
  + **Database**:
    - PostgreSQL for scalable operations.
    - SQLite for offline mode.
  + **Workflow Automation**: Manages task escalations and notifications.
  + **Sync Engine**: Handles .pst files for offline operations.
* **Integration Workflow**:

Infrastructure → Core Services → Interfaces

* + **Example**:
    - An email triggers a workflow via the parsing module.
    - The task is routed through the rule engine.
    - Notifications are sent via Interfaces.

#### **2. Functional Specifications**

* **Core Features**:
  + Role-based communication via email (e.g., hr@organization.com).
  + Dynamic task routing and escalation.
  + Offline synchronization using .pst files.
* **Key Workflows**:
  + **Maintenance Requests**: Automatically routes tasks to appropriate departments.
  + **Harassment Reports**: Anonymizes and routes sensitive data.
  + **Emergency Escalations**: Prioritizes critical tasks with dynamic escalation.

#### **3. Technological Stack**

* **Languages**:
  + Python for backend logic.
  + Go for high-performance processing.
* **Databases**:
  + PostgreSQL for online use.
  + SQLite for offline scenarios.
* **Frameworks**:
  + Flask or FastAPI for APIs.
  + Django Admin for lightweight management.
* **Protocols**:
  + SMTP/IMAP for email handling.

#### **4. Deployment Plan**

* **Steps**:
  1. **Install Dependencies**: Python, PostgreSQL, and libraries.
  2. **Configure Email Servers**: Secure with TLS and role-based addresses.
  3. **Setup Offline Mode**:
     + Configure .pst file synchronization.
     + Use SQLite for local operations.
  4. **Containerization**:
     + Deploy using Docker or Kubernetes.
     + Example: Multi-container orchestration via Docker Compose.
* **Integration with Core Services**:
  + Infrastructure scripts (backup.py, sync.py) feed data into Core Services for task routing.

#### **5. Security Configuration**

* **Encryption**:
  + TLS for secure communication.
  + AES-256 for data storage.
* **Role-Based Access Control (RBAC)**:
  + Limits access to sensitive workflows.
* **Anonymization**:
  + Removes identifying data in logs and workflows.

#### **6. Workflow Integration**

* **Dynamic Rule Loading**:
  + Routing and escalation rules dynamically adjust based on /config/rules/.
  + Example Rule:

- condition: "subject contains 'urgent'"

action:

route\_to: "emergency@organization.com"

priority: "high"

* **Cross-Module Dependency**:
  + **Infrastructure → Core Services**:
    - Sync offline data into PostgreSQL.
  + **Core Services → Interfaces**:
    - Route tasks and escalate via APIs.
* **Preformatted Templates**:
  + Example: Incident reports generated automatically.

#### **7. Testing and Validation**

* **Unit Tests**:
  + Validate workflows (e.g., parsing, routing, notifications).
* **Integration Tests**:
  + Simulate cross-module workflows (e.g., Maintenance to HR escalation).
* **Performance Tests**:
  + Test high email volumes (e.g., 100,000 emails/hour).

#### **8. Scalability and Modularity**

* **Scalability Plan**:
  + Redis for high-volume task queues.
  + Dynamic organization types.
* **Modular Design**:
  + Industry-specific modules (e.g., healthcare, education).

#### **9. Logging and Monitoring**

* **Audit Trails**:
  + Logs workflow and routing actions.
* **Monitoring Tools**:
  + Elastic Stack for real-time metrics.
* **Retention Policies**:
  + Configured dynamically for each organization.

#### **10. Maintenance and Support**

* **Self-Monitoring**:
  + Automates health checks and email server uptime.
* **Troubleshooting**:
  + Documentation for resolving issues.

### **Integration Workflow Example**

**Scenario**: Offline Task Routing

1. **Infrastructure**:
   * A .pst file is uploaded via sync.py and stored in SQLite.
2. **Core Services**:
   * The rule engine processes tasks, resolving conflicts between SQLite and PostgreSQL.
3. **Interfaces**:
   * Notifications are sent to users via APIs.

### **Why the Implementation Blueprint Matters**

1. **Clarity**: Provides a step-by-step guide.
2. **Scalability**: Adapts to organizational growth.
3. **Security**: Ensures sensitive data protection.

### **Conclusion**

This updated Implementation Blueprint integrates cross-module workflows and organization-specific configurations, aligning with Orgo’s goals for modularity and scalability.