### Updated Checklist for Domain Modules

#### Purpose

Domain Modules represent workflows and tasks tailored to specific organizational needs, such as HR, maintenance, or education. This updated checklist emphasizes a centralized and generalized task handling system while maintaining modularity and adaptability for domain-specific needs. It ensures domain modules remain focused, reusable, secure, and aligned with Orgo’s core architecture.

### Checklist

#### 1. Modular Logic

* Ensure tasks are handled dynamically through the unified task handler in /core\_services/task\_handler.py.
* Replace individual domain task files (e.g., /tasks/plumbing\_tasks.py) with logic processed using metadata attributes such as type and metadata.subtype.

#### 2. Directory Structure

* Retain the following subdirectories for domain modules:
  + /templates/: Store templates for emails, reports, or notifications specific to the domain.
  + /rules/: Contain YAML files defining domain-specific workflow routing and escalation rules.
  + /logs/: Track domain-specific actions and errors.

#### 3. Task Management

* Define task types (type, metadata) in the database to dynamically process domain-specific logic.
* Ensure all tasks follow a lifecycle (pending, in-progress, completed) and that their statuses are logged.
* Validate task inputs, such as required fields and data types, before processing.

#### 4. Workflow Rules

* Define reusable workflow rules in /config/workflows/workflow\_rules.yaml for global settings.
* Create domain-specific overrides (e.g., /domain\_modules/maintenance/rules/maintenance\_workflow\_rules.yaml).
* Validate rules for correct syntax and required fields before applying them.

#### 5. Templates

* Include placeholders in email templates for dynamic content (e.g., <TASK\_ID>, <USER\_NAME>).
* Standardize report templates with metadata fields for traceability (e.g., date, author, task details).
* Separate notification templates by medium (e.g., email, SMS, push notifications).

#### 6. Logs

* Maintain domain-specific logs to track actions such as task creation, assignment, and escalation.
* Ensure logs comply with global retention policies, such as storing activity logs for six months.
* Use consistent formats for error and activity logs to simplify integration with monitoring systems.

#### 7. Security

* Restrict domain modules to authorized roles or users using access controls defined in /config/security/authentication\_rules.yaml.
* Anonymize sensitive information, such as reporter identities in HR or compliance workflows.
* Sanitize all inputs from external sources to prevent injection attacks.

#### 8. Reusability

* Extract shared logic, such as task validation or notification generation, into /utils/.
* Store reusable templates in /domain\_modules/common/templates/.
* Use shared YAML rules for workflows applicable across multiple domains.

#### 9. Integration

* Ensure seamless interaction between domain-specific workflows and core services like email and logging.
* Verify that tasks created in domain modules comply with global workflow rules.
* Notify relevant users or groups via notifications managed by /core\_services/notifier\_service.py.

#### 10. Testing

* Write unit tests for domain templates and rules to ensure correctness.
* Conduct integration tests to validate interactions with core services.
* Simulate real-world scenarios for end-to-end testing of domain workflows.

#### 11. Scalability

* Ensure task scripts can handle increased volumes without performance degradation.
* Follow the standardized structure for adding new domain modules:
  + Add workflow rules to /rules/.
  + Create domain-specific templates in /templates/.

### Example Application for a Maintenance Domain

Structure:

* /domain\_modules/maintenance/
  + templates/maintenance\_email\_template.html: Template for maintenance-related emails.
  + rules/maintenance\_workflow\_rules.yaml: Workflow rules specific to maintenance tasks.
  + logs/maintenance\_activity.log: Tracks all maintenance-related actions.

Checklist Applied:

* Tasks dynamically routed based on metadata (e.g., metadata.subtype=plumbing).
* Templates include placeholders like <TASK\_ID> and <ASSIGNED\_USER> and validate replacements before use.
* Workflow rules route tasks based on priority and escalate unresolved issues within the specified timeframe.
* Logs track task lifecycle events and escalation activities.

### Optimized Workflow

1. Shared Components:
   * Define reusable templates, rules, and task scripts to minimize redundancy.
2. Validate Configurations:
   * Validate workflow rules and templates for correctness before deployment.
3. Integrate with Core Services:
   * Ensure seamless interaction with email, notification, and logging services.
4. Testing for Domain Modules:
   * Simulate common workflows and edge cases to ensure reliability and scalability.

### Conclusion

This updated checklist replaces domain-specific task files with a centralized and dynamic approach using Orgo’s core services. It ensures domain modules are modular, secure, and seamlessly integrated, while maintaining flexibility for domain-specific needs. By emphasizing reusability and metadata-driven workflows, the checklist aligns domain modules with Orgo’s commitment to scalability and efficiency.