To help AI make no mistakes and ensure smooth development, you should start with the simplest, foundational group of files that other components rely on. This builds a solid base for future components and reduces errors as more complex modules are added.

---

Recommended Order for AI-Assisted Programming

1. Configuration Files

Why First?

These files define the environment and settings for the entire system (e.g., database connections, email servers, logging levels).

Every other component will reference these settings, so they must be consistent and error-free.

Files to Start With:

/config/email/email\_config.yaml

/config/database/db\_config.yaml

/config/logging/logging\_config.yaml

/config/workflows/workflow\_rules.yaml

---

2. Core Services (Basic Modules)

Why Next?

These are the backbone of Orgo, including basic email handling, database operations, and logging.

They are foundational for workflows, tasks, and interfaces.

Files to Start With:

/core\_services/database/connectors/postgres\_connector.py

/core\_services/email/receiver.py

/core\_services/logging/activity\_logger.py

---

3. Basic Workflow Logic

Why Third?

Workflows are essential for task routing, escalation, and execution.

Starting simple allows debugging and validation before adding domain-specific complexity.

Files to Start With:

/core\_services/workflow/rules/rule\_executor.py

/core\_services/workflow/escalations/escalation\_handler.py

/core\_services/workflow/workflow\_manager.py

---

4. Unit Tests for Core Services

Why Now?

Before moving to more complex files, you should validate the core modules to ensure correctness.

Automated testing reduces the risk of downstream issues.

Files to Start With:

/tests/unit/test\_database/test\_postgres\_connector.py

/tests/unit/test\_email/test\_receiver.py

/tests/unit/test\_logging/test\_activity\_logger.py

---

Why This Order?

1. Starts Simple: Configuration and basic modules are less prone to errors and provide clear validation points.

2. Builds a Foundation: Core services and workflow logic must function correctly before domain-specific tasks or interfaces.

3. Reduces Redundancy: Unit tests ensure that foundational components work as expected, reducing debugging effort later.

---

How to Proceed With AI-Assisted Programming

1. Provide Context:

Share Orgo’s purpose and file dependencies (e.g., workflows depend on configuration files).

2. Define Scope:

Start with small, independent tasks (e.g., writing a single connector or parser file).

3. Test Continuously:

After generating each file, validate functionality with small test scripts or unit tests.

---

Would you like me to prepare code examples or detailed prompts for the files in the first group? Let me know!