Recommended Implementation Order (Step-by-Step)

Phase 0 — Project Foundations (Preparation)

(Implied prerequisites before Phase 1)

1. **Define supported technologies (initial scope)**
    - Decide which frameworks are supported first (e.g. Spring Boot + Maven).
    - Decide which scanners/tools are included (Semgrep, Trivy, Gitleaks, OWASP ZAP).
2. **Set resource & isolation constraints**
    - Max concurrent jobs (e.g. 3).
    - Default CPU & memory limits per job.
    - Decide Docker-based isolation strategy.
3. **Define canonical project structures**
    - Example: Spring Boot must have `pom.xml` at root.
    - Store these definitions in code or config files.

Phase 1 — User Input & Safety Validation

Task Group 1: Input Handling

1. **Expose a job submission API**
    - Accept:
        - ZIP uploads
        - Public GitHub repositories
        - Public DockerHub images (DAST only)
2. **Normalize inputs**
    - ZIP → extract to temp directory

- GitHub → shallow clone
- DockerHub → pull image reference only

Task Group 2: Security Validation (Critical)

1. **Implement archive security checks**
    - Zip bomb detection (compressed vs uncompressed size)
    - File count limits
    - Directory depth limits
    - Path traversal protection (`../`)
2. **Block dangerous content**
    - Disallow binaries/executables
    - Enforce allowed file extensions per framework
3. **Classify validation results**
    - `ACCEPTED`
    - `ACCEPTED_WITH_ISSUES`
    - `REFUSED`

⚠️ **Hard failures must stop the process immediately**

⚠️ **Rejected jobs never create a workspace**

Phase 2 — Job Creation & Project Analysis

Task Group 3: Structure Detection

1. **Detect project structure**
    - Identify framework (Spring, Angular, FastAPI, etc.)
    - Validate against canonical structure
    - Detect missing but non-critical elements (e.g. no tests)
2. **Collect project metadata**

- Language
- Framework

- Build tool
- Tool versions
- Selected pipeline stages

Task Group 4: Metadata Generation

1. **Generate `project.json` metadata**

- Store:
  - Project info
  - Pipeline configuration
  - Execution constraints
  - Job ID

1. **Assign a unique job ID**

- Example: `job-2025-00127`

Task Group 5: Job Queueing

1. **Implement an asynchronous job queue**

- Max 3 concurrent jobs
- FIFO ordering
- Rejected jobs never enter the queue

1. **Allocate default resources per job**

- CPU
- Memory
- Network isolation

Phase 3 — Pipeline Selection & Definition

Task Group 6: Pipeline Architecture

1. **Create pipeline script structure**

```
pipelines/
  spring/
    build.sh
    test.sh
    package.sh
    run-smoke.sh
    dast.sh
```

1. **Ensure pipelines are framework-specific**

- No framework logic in backend code
- Backend only executes scripts

Task Group 7: Stage Coverage

1. **Implement standard CI/CD stages**

- Build
- Unit tests
- SAST
- SCA
- Secret scanning
- Container scanning
- Smoke / integration tests
- DAST (optional)

1. **Make pipeline execution metadata-driven**

- `project.json` determines which scripts run
- Missing stages are skipped safely

Phase 4 — Pipeline Execution & Isolation

Task Group 8: Runner Infrastructure

1. **Build “fat runner” Docker images**

- Preinstalled runtimes (Java, Maven, Node, etc.)
- Preinstalled security tools

1. **Mount job workspace into runner**

- `project/` → read-only
- `reports/` → writable
- `pipelines/` → executable
- `metadata/` → read-only

Task Group 9: Secure Execution

1. **Run each job in an isolated container**

- Private Docker network per job
- No access to host or other jobs

1. **Handle DAST safely**

- App container + OWASP ZAP container
- Same isolated network only

Phase 5 — Reporting & Cleanup

Task Group 10: Result Handling

1. **Collect tool outputs**

- Parse raw outputs
- Normalize into a common JSON schema
- Classify findings by severity

1. **Aggregate final report**

- Per-stage results
- Overall job summary
- Execution metadata

Task Group 11: Cleanup & Resource Management

1. **Destroy job environment**

- Stop containers
- Remove Docker networks
- Delete workspace folders

1. **Ensure no leftover artifacts**

- Prevent disk leaks
- Prevent container leaks
