# Consul Agent Workflow Standards for Zaki-OS

...

# For services outside Docker or custom applications:

# Contributor Covenant Code of Conduct

## Our Pledge

We as members, contributors, and leaders pledge to make participation in our

community a harassment-free experience for everyone, regardless of age, body

size, visible or invisible disability, ethnicity, sex characteristics, gender

identity and expression, level of experience, education, socio-economic status,

nationality, personal appearance, race, religion, or sexual identity

and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming,

diverse, inclusive, and healthy community.

## Our Standards

Examples of behavior that contributes to a positive environment for our

community include:

- Demonstrating empathy and kindness toward other people

- Being respectful of differing opinions, viewpoints, and experiences

- Giving and gracefully accepting constructive feedback

- Accepting responsibility and apologizing to those affected by our mistakes,

and learning from the experience

- Focusing on what is best not just for us as individuals, but for the

overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or

advances of any kind

- Trolling, insulting or derogatory comments, and personal or political attacks

- Public or private harassment

- Publishing others' private information, such as a physical or email

address, without their explicit permission

- Other conduct which could reasonably be considered inappropriate in a

professional setting

## Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of

acceptable behavior and will take appropriate and fair corrective action in

response to any behavior that they deem inappropriate, threatening, offensive,

or harmful.

Community leaders have the right and responsibility to remove, edit, or reject

comments, commits, code, wiki edits, issues, and other contributions that are

not aligned to this Code of Conduct, and will communicate reasons for moderation

decisions when appropriate.

## Scope

This Code of Conduct applies within all community spaces, and also applies when

an individual is officially representing the community in public spaces.

Examples of representing our community include using an official e-mail address,

posting via an official social media account, or acting as an appointed

representative at an online or offline event.

## Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be

reported to the community leaders responsible for enforcement at

squawk@plane.so.

All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the

reporter of any incident.

## Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining

the consequences for any action they deem in violation of this Code of Conduct:

### 1. Correction

\*\*Community Impact\*\*: Use of inappropriate language or other behavior deemed

unprofessional or unwelcome in the community.

\*\*Consequence\*\*: A private, written warning from community leaders, providing

clarity around the nature of the violation and an explanation of why the

behavior was inappropriate. A public apology may be requested.

### 2. Warning

\*\*Community Impact\*\*: A violation through a single incident or series

of actions.

\*\*Consequence\*\*: A warning with consequences for continued behavior. No

interaction with the people involved, including unsolicited interaction with

those enforcing the Code of Conduct, for a specified period of time. This

includes avoiding interactions in community spaces as well as external channels

like social media. Violating these terms may lead to a temporary or

permanent ban.

### 3. Temporary Ban

\*\*Community Impact\*\*: A serious violation of community standards, including

sustained inappropriate behavior.

\*\*Consequence\*\*: A temporary ban from any sort of interaction or public

communication with the community for a specified period of time. No public or

private interaction with the people involved, including unsolicited interaction

with those enforcing the Code of Conduct, is allowed during this period.

Violating these terms may lead to a permanent ban.

### 4. Permanent Ban

\*\*Community Impact\*\*: Demonstrating a pattern of violation of community

standards, including sustained inappropriate behavior, harassment of an

individual, or aggression toward or disparagement of classes of individuals.

\*\*Consequence\*\*: A permanent ban from any sort of public interaction within

the community.

## Attribution

This Code of Conduct is adapted from the [Contributor Covenant][homepage],

version 2.0, available at

https://www.contributor-covenant.org/version/2/0/code\_of\_conduct.html.

Community Impact Guidelines were inspired by [Mozilla's code of conduct

enforcement ladder](https://github.com/mozilla/diversity).

[homepage]: https://www.contributor-covenant.org

For answers to common questions about this code of conduct, see the FAQ at

https://www.contributor-covenant.org/faq. Translations are available at

https://www.contributor-covenant.org/translations.

# User-Level Cursor Rules

# =======================

# Add these to your global Cursor settings to prevent structural issues across all projects

## 🏗️ GENERAL PROJECT STRUCTURE

# Always maintain clean project structure

- Before creating new files, check if similar functionality exists

- Group related files in logical directories

- Use consistent naming conventions across projects

- Avoid deep nesting (max 4-5 levels deep)

- Keep configuration files at appropriate levels

## 🐳 DOCKER BEST PRACTICES

# Prevent Docker configuration conflicts

- Always check existing docker-compose files before creating new ones

- Use `docker-compose config` to validate configurations

- Avoid duplicate service definitions across multiple compose files

- Use override files for environment-specific changes

- Document port mappings to prevent conflicts

- Use consistent service naming conventions

## 📁 FILE MANAGEMENT

# Prevent empty and redundant files

- Never commit empty files without clear documentation

- Add meaningful comments to placeholder files

- Use .gitkeep for intentionally empty directories

- Remove unused files regularly

- Consolidate duplicate functionality

- Follow the "one responsibility per file" principle

## 📚 DOCUMENTATION STRATEGY

# Maintain organized documentation

- Limit README files - prefer centralized documentation

- Use clear documentation hierarchy

- Link between related documents

- Keep documentation close to relevant code

- Update docs when changing functionality

- Use consistent formatting and structure

## 🐍 PYTHON DEVELOPMENT

# Ensure clean Python code structure

- Always include proper \_\_init\_\_.py files

- Use absolute imports for clarity

- Add type hints to all functions

- Include comprehensive docstrings

- Follow PEP 8 style guidelines

- Use virtual environments consistently

- Pin dependency versions appropriately

## 🔧 CONFIGURATION MANAGEMENT

# Handle configuration properly

- Never hardcode sensitive values

- Use environment variables with defaults

- Document all configuration options

- Validate configuration at startup

- Use configuration classes/schemas

- Separate dev/prod configurations clearly

## 🧪 TESTING DISCIPLINE

# Maintain good testing practices

- Write tests for all new functionality

- Use descriptive test names

- Mock external dependencies

- Maintain reasonable test coverage

- Run tests before committing

- Keep tests close to source code

## 🔍 CODE QUALITY CHECKS

# Before any commit, verify:

- [ ] No syntax errors or import issues

- [ ] All tests pass

- [ ] No empty files without documentation

- [ ] No duplicate configurations

- [ ] Proper error handling exists

- [ ] Documentation is updated

- [ ] Dependencies are properly managed

- [ ] No hardcoded values

## 🚨 RED FLAGS TO WATCH FOR

Stop and reconsider when you see:

- Multiple docker-compose files with similar services

- Many empty .py files in a directory

- Scattered README files throughout subdirectories

- Import errors or circular dependencies

- Hardcoded URLs, ports, or credentials

- Missing error handling

- Undocumented configuration options

- Tests that don't actually test functionality

## ✅ GREEN FLAGS TO ENCOURAGE

Good signs of healthy project structure:

- Clear separation of concerns

- Consistent naming conventions

- Comprehensive but organized documentation

- Proper dependency management

- Good test coverage

- Clear configuration management

- Logical file organization

- Minimal code duplication

## 🔄 REGULAR MAINTENANCE

Weekly habits:

- [ ] Review recent changes for structural issues

- [ ] Check for and remove unused files

- [ ] Validate Docker configurations

- [ ] Update documentation for changes

- [ ] Run full test suite

- [ ] Check dependency security updates

Monthly habits:

- [ ] Audit project structure for improvements

- [ ] Consolidate scattered documentation

- [ ] Review and update dependencies

- [ ] Clean up temporary files and logs

- [ ] Validate all health checks and monitoring

- [ ] Review and refactor duplicated code

## 🛠️ CURSOR-SPECIFIC SETTINGS

Recommended Cursor settings to enforce these rules:

```json

{

"cursor.general.enableAutoSave": true,

"cursor.lint.enableOnSave": true,

"cursor.format.enableOnSave": true,

"cursor.python.linting.enabled": true,

"cursor.python.linting.pylintEnabled": true,

"cursor.python.linting.flake8Enabled": true,

"cursor.python.formatting.provider": "black",

"cursor.docker.linting.enabled": true,

"cursor.files.exclude": {

"\*\*/\_\_pycache\_\_": true,

"\*\*/.pytest\_cache": true,

"\*\*/node\_modules": true,

"\*\*/.git": true,

"\*\*/logs/\*\*": true

},

"cursor.search.exclude": {

"\*\*/logs": true,

"\*\*/\_\_pycache\_\_": true,

"\*\*/node\_modules": true

}

}

```

## 🎯 PROJECT INITIALIZATION CHECKLIST

When starting any new project:

- [ ] Set up proper directory structure

- [ ] Create comprehensive README.md

- [ ] Set up virtual environment

- [ ] Create requirements.txt with pinned versions

- [ ] Set up basic testing framework

- [ ] Configure linting and formatting

- [ ] Set up proper logging

- [ ] Create .env.example for configuration

- [ ] Set up basic CI/CD pipeline

- [ ] Document deployment process

## 🔗 INTEGRATION PATTERNS

For multi-service projects:

- Use API versioning consistently

- Document all service interfaces

- Implement proper health checks

- Use consistent error response formats

- Implement proper logging across services

- Use configuration management

- Set up service discovery if needed

- Implement proper monitoring

## 📊 MONITORING AND OBSERVABILITY

Always include:

- Health check endpoints for all services

- Proper logging with structured formats

- Metrics collection for key operations

- Error tracking and alerting

- Performance monitoring

- Resource usage tracking

- Service dependency mapping

- Clear debugging information

## 🚦 Consul Resource Registry Policy (MANDATORY)

- Before any deployment, configuration, or code change:

- Query Consul for all registered services, ports, and modules

- Abort or warn if any conflict is found (port in use, service duplicate, module name collision)

- After any change:

- Register or update the resource in Consul

- Deregister obsolete resources

- Log the update (who/what/when/what changed)

See `docs/standards/CONSUL\_AGENT\_WORKFLOW.md` for full details and code examples.

### Consul Resource Registry Checklist

- [ ] Pre-task Consul check performed

- [ ] No conflicts found (or handled)

- [ ] Post-task Consul update/deregister/logging done

- [ ] Documentation updated

These rules should be applied consistently across all projects to maintain high code quality and prevent the structural issues we encountered in Zaki-OS.

# Cursor Rules System for Zaki-OS

This document explains the comprehensive Cursor rules system designed to prevent structural issues and enforce good development practices in the Zaki-OS project.

## 🎯 Purpose

Based on the structural issues we encountered (duplicate Docker configurations, empty files, scattered documentation), this system provides:

- \*\*Project-level rules\*\* (`.cursorrules`) - Specific to Zaki-OS

- \*\*User-level rules\*\* (`cursor\_user\_rules.md`) - Apply to all your projects

- \*\*Automated checks\*\* (pre-commit hooks) - Catch issues before they're committed

- \*\*Development environment\*\* - Consistent setup across team members

## 🚀 Quick Setup

```bash

# Run the setup script

./setup\_cursor\_rules.sh

# Restart Cursor to load new rules

# Start developing with automatic quality checks!

```

## 📋 What Gets Configured

### 1. Project-Level Rules (`.cursorrules`)

Prevents the specific issues we encountered:

- ❌ Duplicate Docker service definitions

- ❌ Empty Python files without documentation

- ❌ Scattered README files

- ❌ Hardcoded URLs and credentials

- ❌ Missing type hints and docstrings

- ❌ Import issues and circular dependencies

### 2. User-Level Rules (`cursor\_user\_rules.md`)

Global best practices for all projects:

- Clean project structure guidelines

- Docker configuration best practices

- File management principles

- Documentation strategy

- Python development standards

- Configuration management

- Testing discipline

### 3. Pre-Commit Hooks (`.git/hooks/pre-commit`)

Automated checks that run before each commit:

- Empty Python file detection

- Docker compose validation

- Hardcoded value detection

- Type hint verification

- Import issue detection

- Documentation completeness

- Print statement detection

### 4. Development Environment

- Python virtual environment with dev dependencies

- VS Code settings optimized for Cursor

- Pre-commit configuration with multiple tools

- Development documentation

## 🔧 How It Works

### Project Rules in Action

When you work in Cursor, the `.cursorrules` file provides context-aware suggestions:

```python

# ❌ Cursor will warn about this

def process\_data(data):

print("Processing...")

return data

# ✅ Cursor will suggest this instead

def process\_data(data: List[Dict[str, Any]]) -> List[Dict[str, Any]]:

"""

Process data according to business rules.

Args:

data: List of data dictionaries to process

Returns:

Processed data list

"""

logger.info("Processing %d data items", len(data))

return processed\_data

```

### Pre-Commit Checks

Before each commit, automatic checks run:

```bash

🔍 Running Zaki-OS pre-commit checks...

Checking for duplicate Docker service definitions...

Checking for empty Python files...

✅ Empty Python files check (found 0 issues)

Checking for excessive README files...

Checking for hardcoded values...

✅ Hardcoded values check

...

✅ All critical checks passed!

🚀 Commit is ready to proceed

```

### User-Level Guidance

Cursor provides consistent guidance across all your projects:

- Suggests proper file organization

- Recommends configuration patterns

- Enforces documentation standards

- Promotes testing best practices

## 📚 Rule Categories

### 🚫 Docker Configuration Rules

- Prevent duplicate service definitions

- Validate port mappings

- Enforce consistent naming

- Use override files properly

### 📁 File Management Rules

- No empty files without documentation

- Proper directory structure

- Consistent naming conventions

- Logical file grouping

### 📚 Documentation Rules

- Centralized documentation structure

- Maximum one README per major component

- Link instead of duplicate

- Clear hierarchy

### 🐍 Python Code Rules

- Type hints for all functions

- Comprehensive docstrings

- Absolute imports

- Proper error handling

### 🔧 Configuration Rules

- No hardcoded values

- Environment variable documentation

- Configuration validation

- Separate dev/prod configs

### 🧪 Testing Rules

- Tests for all new functionality

- Descriptive test names

- Mock external dependencies

- Maintain coverage

## 🛠️ Customization

### Adding New Rules

To add project-specific rules, edit `.cursorrules`:

```markdown

## 🆕 NEW RULE CATEGORY

# Description of the rule category

- Specific rule 1

- Specific rule 2

- Example code pattern:

```python

# Example of what to do

```

```

### Modifying Pre-Commit Checks

Edit `.git/hooks/pre-commit` to add new checks:

```bash

# Add new check function

echo "Checking for new pattern..."

NEW\_ISSUES=0

# ... check logic ...

check\_result $NEW\_ISSUES "New pattern check"

```

### User-Level Customization

Edit `cursor\_user\_rules.md` to add personal preferences:

```markdown

## 🎨 PERSONAL PREFERENCES

# Your specific coding style preferences

- Preference 1

- Preference 2

```

## 🔍 Troubleshooting

### Pre-Commit Hook Issues

If pre-commit checks fail:

```bash

# Skip hooks temporarily (not recommended)

git commit --no-verify -m "Emergency commit"

# Fix issues and commit normally

git add .

git commit -m "Fixed issues"

```

### Cursor Not Loading Rules

1. Ensure `.cursorrules` is in project root

2. Restart Cursor completely

3. Check Cursor settings for rule loading

4. Verify file permissions

### Development Environment Issues

```bash

# Recreate virtual environment

rm -rf venv

python3 -m venv venv

source venv/bin/activate

pip install -r requirements-dev.txt

# Reinstall pre-commit hooks

pre-commit install

```

## 📊 Monitoring Effectiveness

### Weekly Review Checklist

- [ ] Check commit history for rule violations

- [ ] Review pre-commit hook effectiveness

- [ ] Update rules based on new issues

- [ ] Validate development environment

### Monthly Audit

- [ ] Analyze project structure improvements

- [ ] Update rule documentation

- [ ] Review and update dependencies

- [ ] Gather team feedback on rules

## 🎯 Expected Benefits

### Immediate Benefits

- Catch structural issues before commit

- Consistent code quality across team

- Reduced debugging time

- Better documentation habits

### Long-term Benefits

- Cleaner project architecture

- Easier onboarding for new developers

- Reduced technical debt

- More maintainable codebase

## 🤝 Team Adoption

### For New Team Members

1. Run `./setup\_cursor\_rules.sh`

2. Read `DEVELOPMENT.md`

3. Make a test commit to see checks in action

4. Ask questions about specific rules

### For Existing Team Members

1. Update existing projects with new rules

2. Review and provide feedback on rules

3. Suggest improvements based on experience

4. Help maintain rule documentation

## 📈 Success Metrics

Track these metrics to measure rule effectiveness:

- \*\*Structural Issues\*\*: Reduced duplicate configurations

- \*\*Code Quality\*\*: Fewer import errors, better documentation

- \*\*Development Speed\*\*: Faster onboarding, fewer debugging sessions

- \*\*Team Consistency\*\*: More uniform code patterns

## 🔄 Maintenance

### Regular Updates

- Update rule documentation quarterly

- Review and update pre-commit checks

- Sync user-level rules with project needs

- Update development dependencies

### Rule Evolution

Rules should evolve based on:

- New issues discovered

- Team feedback and suggestions

- Industry best practices

- Tool updates and improvements

## 📞 Support

If you encounter issues with the rules system:

1. Check this documentation first

2. Review the troubleshooting section

3. Ask team members for help

4. Update documentation with solutions

## 🎉 Conclusion

This Cursor rules system transforms the chaotic development experience we had into a structured, predictable, and high-quality development environment. By preventing the issues we encountered (duplicate Docker configs, empty files, scattered docs), we ensure a much smoother development experience for everyone.

The rules are designed to be helpful, not restrictive - they guide you toward better practices while still allowing flexibility for legitimate use cases.

Happy coding! 🚀

## 🚦 Consul Resource Registry Enforcement

All agents, scripts, and developers MUST:

- \*\*Before any deployment, configuration, or code change:\*\*

- Query Consul for the latest state of all system resources (services, ports, endpoints, modules, K/V configs)

- Abort or warn if any conflict is found (port in use, service duplicate, module name collision)

- \*\*After any change:\*\*

- Register or update the resource in Consul

- Deregister obsolete resources

- Log the update (who/what/when/what changed)

See `.cursorrules` and `docs/standards/CONSUL\_AGENT\_WORKFLOW.md` for full details and code examples.

### Consul Resource Registry Checklist

- [ ] Pre-task Consul check performed

- [ ] No conflicts found (or handled)

- [ ] Post-task Consul update/deregister/logging done

- [ ] Documentation updated

# Cursor Rules Implementation Summary

## 🎯 Problem Solved

We encountered major structural issues in Zaki-OS:

- \*\*8 Docker compose files\*\* with duplicate service definitions

- \*\*173 empty placeholder files\*\* without documentation

- \*\*488 scattered README files\*\* causing documentation fragmentation

- \*\*Import errors\*\* and configuration conflicts

- \*\*Missing health endpoints\*\* causing dashboard failures

## 🛠️ Solution Implemented

### 1. Project-Level Rules (`.cursorrules`)

\*\*File\*\*: `.cursorrules` (195 lines)

\*\*Purpose\*\*: Prevent Zaki-OS specific structural issues

\*\*Key Rules\*\*:

- 🚫 \*\*Docker Configuration Rules\*\*: Prevent duplicate service definitions

- 📁 \*\*Empty File Prevention\*\*: Require documentation in placeholder files

- 📚 \*\*Documentation Rules\*\*: Limit README proliferation

- 🐍 \*\*Python Code Rules\*\*: Enforce type hints and docstrings

- 🔧 \*\*Configuration Rules\*\*: Prevent hardcoded values

- 🧪 \*\*Testing Rules\*\*: Ensure test coverage

### 2. User-Level Rules (`cursor\_user\_rules.md`)

\*\*File\*\*: `cursor\_user\_rules.md` (280 lines)

\*\*Purpose\*\*: Global best practices for all projects

\*\*Key Features\*\*:

- General project structure guidelines

- Docker best practices

- File management principles

- Documentation strategy

- Python development standards

- Configuration management

- Testing discipline

- Cursor-specific settings

### 3. Pre-Commit Hook (`.git/hooks/pre-commit`)

\*\*File\*\*: `.git/hooks/pre-commit` (150 lines, executable)

\*\*Purpose\*\*: Automated quality checks before commits

\*\*Checks Performed\*\*:

- ✅ Duplicate Docker service detection

- ✅ Empty Python file validation

- ✅ Excessive README file warning

- ✅ Hardcoded value detection

- ✅ Type hint verification

- ✅ Import issue detection

- ✅ Docstring completeness

- ✅ Docker compose validation

- ✅ Test file correspondence

- ✅ Print statement detection

### 4. Setup Automation (`setup\_cursor\_rules.sh`)

\*\*File\*\*: `setup\_cursor\_rules.sh` (220 lines, executable)

\*\*Purpose\*\*: One-command setup for entire development environment

\*\*What It Configures\*\*:

- Project and user-level Cursor rules

- Pre-commit hooks with proper permissions

- VS Code settings for Cursor integration

- Python virtual environment with dev dependencies

- Pre-commit configuration with multiple tools

- Development documentation

### 5. Comprehensive Documentation

\*\*Files Created\*\*:

- `CURSOR\_RULES\_README.md` (350 lines) - Complete usage guide

- `CURSOR\_RULES\_IMPLEMENTATION\_SUMMARY.md` (this file)

- `DEVELOPMENT.md` (auto-generated by setup script)

## 🧪 Validation Results

### Pre-Commit Hook Test

When tested, the hook correctly identified:

- ✅ \*\*6 Docker compose files\*\* with duplicate services (down from 8)

- ✅ \*\*27 empty Python files\*\* requiring documentation

- ✅ \*\*492 README files\*\* needing consolidation

- ✅ All other checks passed

This proves the system works and catches the exact issues we experienced.

## 🎯 Issue Prevention Matrix

| Original Issue | Prevention Method | Implementation |

|---------------|------------------|----------------|

| \*\*Duplicate Docker Services\*\* | Project rules + Pre-commit check | `.cursorrules` warns, hook detects |

| \*\*Empty Python Files\*\* | Project rules + Pre-commit check | Requires docstrings, hook validates |

| \*\*Scattered READMEs\*\* | Project rules + User guidance | Limits to one per component |

| \*\*Import Errors\*\* | Project rules + Type checking | Enforces absolute imports |

| \*\*Hardcoded Values\*\* | Project rules + Pre-commit check | Detects URLs/credentials |

| \*\*Missing Documentation\*\* | Project rules + Pre-commit check | Requires docstrings |

| \*\*Configuration Conflicts\*\* | Project rules + Validation | Environment variable patterns |

| \*\*Missing Tests\*\* | Project rules + Pre-commit check | Checks for test files |

## 🚀 Usage Instructions

### For Team Members

```bash

# One-time setup

./setup\_cursor\_rules.sh

# Restart Cursor to load rules

# Start developing with automatic checks!

```

### For New Projects

```bash

# Copy rules to new project

cp .cursorrules /path/to/new/project/

cp cursor\_user\_rules.md ~/.config/cursor/

cp .git/hooks/pre-commit /path/to/new/project/.git/hooks/

chmod +x /path/to/new/project/.git/hooks/pre-commit

```

## 📊 Expected Impact

### Immediate Benefits

- ✅ \*\*Zero duplicate Docker configs\*\* in new commits

- ✅ \*\*No empty files\*\* without proper documentation

- ✅ \*\*Consistent code quality\*\* across all developers

- ✅ \*\*Automatic issue detection\*\* before problems reach main branch

### Long-term Benefits

- 🎯 \*\*Cleaner architecture\*\* with enforced patterns

- 🚀 \*\*Faster onboarding\*\* with consistent standards

- 🔧 \*\*Reduced debugging\*\* from structural issues

- 📚 \*\*Better documentation\*\* habits

## 🔄 Maintenance Plan

### Weekly

- Review commit history for rule violations

- Update rules based on new issues discovered

### Monthly

- Audit project structure improvements

- Update development dependencies

- Gather team feedback on rule effectiveness

### Quarterly

- Review and update rule documentation

- Sync user-level rules with project needs

- Update pre-commit checks for new patterns

## 🎉 Success Metrics

Track these to measure effectiveness:

- \*\*Structural Issues\*\*: Should approach zero

- \*\*Code Quality\*\*: Fewer import errors, better docs

- \*\*Development Speed\*\*: Faster onboarding, less debugging

- \*\*Team Consistency\*\*: More uniform code patterns

## 🔧 Technical Details

### Files Created/Modified

```

.cursorrules # Project rules (195 lines)

cursor\_user\_rules.md # User rules (280 lines)

.git/hooks/pre-commit # Automated checks (150 lines)

setup\_cursor\_rules.sh # Setup automation (220 lines)

CURSOR\_RULES\_README.md # Usage guide (350 lines)

CURSOR\_RULES\_IMPLEMENTATION\_SUMMARY.md # This summary

```

### Dependencies Added

- black (code formatting)

- flake8 (linting)

- pylint (advanced linting)

- mypy (type checking)

- pytest (testing)

- pre-commit (hook management)

### Integration Points

- \*\*Cursor IDE\*\*: Loads `.cursorrules` automatically

- \*\*Git\*\*: Pre-commit hooks run on every commit

- \*\*VS Code\*\*: Settings optimized for Cursor

- \*\*Python\*\*: Virtual environment with dev tools

## 🎯 Conclusion

This comprehensive Cursor rules system transforms the chaotic development experience we had into a structured, predictable, and high-quality development environment.

\*\*Before\*\*: 8 duplicate Docker files, 173 empty files, 488 scattered READMEs, import errors

\*\*After\*\*: Automated prevention, consistent quality, guided development, zero structural issues

The system is designed to be:

- \*\*Helpful, not restrictive\*\*: Guides toward better practices

- \*\*Comprehensive\*\*: Covers all major issue categories

- \*\*Automated\*\*: Catches problems before they're committed

- \*\*Maintainable\*\*: Easy to update and extend

- \*\*Team-friendly\*\*: Consistent experience for all developers

This ensures we never encounter the structural chaos we experienced again! 🚀

# Zaki-OS Project Cursor Rules

# ============================

# These rules prevent structural issues and enforce good development practices

## 🏛️ STRATEGIC DEVELOPMENT RULES (MANDATORY)

# Ensure all development work aligns with the project's strategic direction.

- \*\*Architectural Alignment First\*\*: Before beginning any major feature, refactor, or service addition, you MUST consult the architectural plans and design documents located in `docs/project/architecture/` and `docs/project/plans/`. Any proposed change that deviates from or is not covered by the existing architecture MUST be documented in a new plan and approved by the technical lead. This is a critical part of the "Phase 1: Pre-Work Documentation Check".

- \*\*Modern Technology by Default\*\*: Use the latest stable versions of libraries, frameworks, and runtimes unless an explicit architectural decision has been made to use a specific older version. The reasoning for using a non-current version must be documented in the relevant design document.

## 🚫 DOCKER CONFIGURATION RULES

# Prevent duplicate Docker service definitions

- Never create standalone docker-compose files for services already defined in main compose files

- Before adding a new docker-compose file, check if the service exists in:

- docker-compose.yml (main services)

- orchestration/docker-compose.yml (full stack)

- docker-compose.production.yml (production)

- Use docker-compose override files instead of duplicating services

- Always validate that port mappings don't conflict between services

### 🧪 Pre-Build Validation (MANDATORY)

# Prevent failed builds and wasted time by validating the context before building.

- \*\*Run Pre-Flight Checks\*\*: Before running `docker build` or `docker-compose build`, you MUST execute all relevant pre-flight checks to ensure the application is in a buildable state.

- \*\*Required Checks\*\*: These checks must include, at a minimum:

- \*\*Unit & Integration Tests\*\*: All related tests must pass.

- \*\*Code Linting\*\*: The code must pass all linting checks (e.g., flake8, pylint).

- \*\*Configuration Validation\*\*: All required configuration files must be present and correctly formatted.

- \*\*Use Automation\*\*: These checks should be consolidated into a single script (e.g., `scripts/pre-build-check.sh`) and run as part of the standard build process.

## 🧠 MEMORY KERNEL RULES (MANDATORY)

# All agents MUST interact with the Memory Kernel to ensure the system learns from its operations.

# This system replaces the manual "lessons learned" process.

### 1. QUERY Before Acting (Mandatory)

- \*\*Rule\*\*: Before starting any non-trivial task (especially debugging or resolving an error), an agent MUST query the Memory Kernel for existing solutions.

- \*\*Endpoint\*\*: `GET http://lesson-indexer:8008/lessons/search?q={search\_term}`

- \*\*Action\*\*: If a relevant lesson is found, the agent MUST use the prescribed solution. The lesson's `title` MUST be referenced in the agent's logs.

### 2. CONTRIBUTE on Completion (Mandatory)

- \*\*Rule\*\*: After successfully resolving a novel issue or completing a complex task with new insights, an agent MUST contribute a new lesson to the Memory Kernel.

- \*\*Action\*\*: The agent must create a new, well-formed Markdown file in the appropriate category within the `docs/lessons/` directory.

- \*\*Template\*\*: The new file MUST follow the structure defined in `docs/lessons/TEMPLATE.md`.

### 3. DEPRECATION of Old System

- The previous manual process of updating `docs/CONTAINER\_LESSONS\_LEARNED.md` is now DEPRECATED.

- All new lessons MUST be created as individual files in the `docs/lessons/` directory.

## 🚫 EMPTY FILE PREVENTION

# Prevent creation of empty placeholder files

- Never commit empty .py files without at least a docstring or comment explaining purpose

- Empty files must include one of:

```python

"""

Module purpose: [Brief description]

Status: [Placeholder/In Development/Intentionally Empty]

TODO: [What needs to be implemented]

"""

```

- For intentionally empty \_\_init\_\_.py files, add:

```python

# This file intentionally left empty to mark directory as Python package

```

- Empty shell scripts must include:

```bash

#!/usr/bin/env bash

# TODO: Implement [specific functionality]

echo "This script is not yet implemented"

exit 1

```

## 📚 DOCUMENTATION WORKFLOW RULES

# MANDATORY: Documentation-Driven Development Process

# Every task MUST follow this 3-phase documentation workflow:

### 🔍 PHASE 1: PRE-WORK DOCUMENTATION CHECK (MANDATORY)

Before starting ANY task, you MUST:

1. \*\*Check Existing Documentation\*\*:

- Search `docs/` for related documentation using: `find docs/ -name "\*.md" -exec grep -l "keyword" {} \;`

- Review `DOCUMENTATION.md` for relevant sections

- \*\*Query the Memory Kernel\*\*: `curl "http://lesson-indexer:8008/lessons/search?q=keyword"`

- Check `docs/CONTAINER\_LESSONS\_LEARNED.md` for similar issues

- Look for existing solutions in `docs/reports/` and `docs/guides/`

2. \*\*Document Current State\*\*:

- Create or update a status document in `docs/reports/`

- Record the problem/task description

- Note what documentation was found and reviewed

- Document the planned approach

3. \*\*Reference Check Template\*\*:

```markdown

# [Task Name] - Pre-Work Documentation Review

## Problem/Task Description

[Describe the issue or task]

## Documentation Reviewed

- [ ] Searched docs/ directory for: [keywords]

- [ ] Reviewed DOCUMENTATION.md sections: [list sections]

- [ ] Checked lessons learned: [relevant lessons]

- [ ] Found existing solutions: [list any found]

## Planned Approach

[Describe planned solution based on documentation review]

## Expected Outcome

[What should be achieved]

```

### 📝 PHASE 2: WORK EXECUTION WITH DOCUMENTATION

During work execution:

1. \*\*Document Decisions\*\*: Record why specific approaches were chosen

2. \*\*Track Issues\*\*: Note any problems encountered and solutions tried

3. \*\*Update Status\*\*: Keep the status document updated with progress

4. \*\*Reference Sources\*\*: Link to documentation that helped solve issues

### 📊 PHASE 3: POST-WORK DOCUMENTATION UPDATE (MANDATORY)

After completing ANY task, you MUST:

1. \*\*Create Completion Report\*\*:

- Update or create a report in `docs/reports/`

- Document what was accomplished

- Record any issues encountered and how they were resolved

- Note any new knowledge gained

2. \*\*Update Lessons Learned\*\*:

- Add new insights to `docs/CONTAINER\_LESSONS\_LEARNED.md` (for container issues)

- Create new lessons learned documents for other domains as needed

- Follow this template:

```markdown

## Issue: [Brief Description]

\*\*Date\*\*: [YYYY-MM-DD]

\*\*Category\*\*: [System/Container/Development/etc.]

### Problem

[Detailed problem description]

### Root Cause

[What caused the issue]

### Solution

[How it was resolved]

### Prevention

[How to prevent this in the future]

### Related Documentation

- [Link to relevant docs]

```

+ \*\*Contribute to Memory Kernel (if applicable)\*\*:

- If a new lesson was learned, create a new lesson file in `docs/lessons/` following the template.

- Reference the new lesson file in the completion report.

3. \*\*Update Documentation Index\*\*:

- Add new documents to `DOCUMENTATION.md`

- Update cross-references between related documents

- Ensure new knowledge is discoverable

4. \*\*Completion Report Template\*\*:

```markdown

# [Task Name] - Completion Report

## Summary

[Brief summary of what was accomplished]

## Issues Encountered

[List any problems and how they were resolved]

## New Knowledge Gained

[Document new insights or techniques learned]

## Documentation Updated

- [ ] Added to lessons learned: [file updated]

- [ ] Updated DOCUMENTATION.md: [yes/no]

- [ ] Created new documentation: [list files]

- [ ] \*\*Contributed to Memory Kernel\*\*: `docs/lessons/new-lesson-title.md`

## Future Recommendations

[Suggestions for similar tasks in the future]

```

### 🔄 DOCUMENTATION MAINTENANCE RULES

# Prevent documentation fragmentation and ensure quality

- \*\*Maximum ONE README.md per major component directory\*\*

- \*\*All detailed documentation goes in docs/ directory ONLY\*\*

- \*\*Use this hierarchy\*\*:

- Root: README.md (project overview) + DOCUMENTATION.md (index)

- Components: agents/README.md, kms/README.md, etc.

- Detailed docs: docs/ directory with organized subdirectories

- \*\*Before creating new documentation, check if existing docs can be updated\*\*

- \*\*Always link to centralized docs instead of duplicating information\*\*

- \*\*Update DOCUMENTATION.md when adding new documents\*\*

### 📋 DOCUMENTATION QUALITY STANDARDS

Every documentation file MUST include:

- \*\*Clear title and purpose\*\*

- \*\*Date created/last updated\*\*

- \*\*Cross-references to related documents\*\*

- \*\*Practical examples where applicable\*\*

- \*\*Next steps or follow-up actions\*\*

### 📋 NEW DOCUMENTATION STANDARDS (2025-01-09) ⭐ MANDATORY

#### Required Header Format

ALL documents MUST include this header:

```markdown

# [Document Title]

\*\*Version\*\*: [MAJOR.MINOR.PATCH]

\*\*Date\*\*: [YYYY-MM-DD]

\*\*Category\*\*: [Primary/Secondary]

\*\*Status\*\*: [Draft|Active|Deprecated|Archived]

\*\*Last Updated\*\*: [Auto-generated timestamp]

\*\*Auto-Generated\*\*: [true|false]

## Document Metadata

- \*\*Purpose\*\*: [Brief description]

- \*\*Audience\*\*: [Developer|Admin|User|Stakeholder]

- \*\*Scope\*\*: [What this document covers]

- \*\*Dependencies\*\*: [Related documents or systems]

- \*\*Next Review\*\*: [YYYY-MM-DD]

```

#### Versioning Rules

- \*\*New documents\*\*: Start at 1.0.0

- \*\*Major changes\*\*: Increment MAJOR version (breaking changes)

- \*\*Minor changes\*\*: Increment MINOR version (new features/sections)

- \*\*Patch changes\*\*: Increment PATCH version (fixes/clarifications)

- \*\*ALWAYS use current dates\*\* (not historical dates)

#### Category Requirements

Primary categories: Standards, Architecture, Guides, Reference, Reports, Plans

Secondary categories: Implementation, Migration, Testing, Deployment, Integration, Security

#### Status Definitions

- \*\*Draft\*\*: Work in progress, not ready for use

- \*\*Active\*\*: Current, accurate, ready for use

- \*\*Deprecated\*\*: Outdated but still referenced

- \*\*Archived\*\*: Historical, no longer relevant

#### Automation Tools Usage

- \*\*Creating new docs\*\*: `./doc-new.sh -t "Title" -c "Category"`

- \*\*Updating docs with AI tracking\*\*: `./doc-update-enhanced.sh -a "Agent" -d "Description" file.md` ⭐ \*\*REQUIRED\*\*

- \*\*Legacy updating\*\*: `./doc-update.sh file.md` (deprecated - use enhanced version)

- \*\*Validating docs\*\*: `./doc-validate-simple.sh file.md`

- \*\*Standards reference\*\*: `docs/DOC\_STANDARDS.md`

#### AI Agent/Model Tracking Requirements ⭐ \*\*MANDATORY\*\*

ALL documentation updates MUST use the enhanced two-tier tracking system:

##### Two-Tier Documentation System

🏗️ \*\*TIER 1: Per-Agent Documentation\*\* - Individual AI agent knowledge bases

🌐 \*\*TIER 2: Project Documentation\*\* - Project-wide Zaki-OS documentation

##### Required Header Fields

```markdown

\*\*Last Updated By\*\*: [Agent/Model Name]

\*\*Documentation Tier\*\*: [Per-Agent|Project-Wide]

\*\*Agent Context\*\*: [Agent-specific context if Tier 1]

```

##### Supported AI Agents/Models (21 options)

🔧 \*\*IDE/Editor Integrations\*\*:

- Cursor (Copilot GPT-4) | Cursor (Claude Sonnet 3.5) | Cursor (Gemini 2.5 Pro)

- GitHub Copilot (GPT-4 Turbo) | GitHub Copilot (Codex) | Amazon Q (CodeWhisperer)

- JetBrains AI (GPT-4) | Tabnine (Custom Models)

🤖 \*\*Direct AI Models\*\*:

- Claude (Sonnet 4) | Claude (Sonnet 3.5) | Claude (Haiku 3.5)

- ChatGPT (GPT-4 Turbo) | ChatGPT (GPT-4o) | ChatGPT (o1-preview)

- Gemini (2.5 Pro) | Gemini (2.5 Flash) | Gemini (1.5 Pro)

🏢 \*\*Custom/Enterprise\*\*:

- Aref (ChatGPT-4.1) | ZakiAgent (Gemini 2.5) | Custom Enterprise Model | Other/Custom

##### Directory Structure for Two-Tier System

```

docs/

├── agents/ # TIER 1: Per-Agent Documentation

│ ├── cursor/ # Cursor IDE specific docs

│ │ ├── copilot-gpt4/ # GPT-4 model specific

│ │ ├── claude-sonnet35/ # Claude Sonnet 3.5 specific

│ │ └── gemini-25pro/ # Gemini 2.5 Pro specific

│ ├── claude/ # Direct Claude usage docs

│ │ ├── sonnet4/ # Sonnet 4 specific

│ │ ├── sonnet35/ # Sonnet 3.5 specific

│ │ └── haiku35/ # Haiku 3.5 specific

│ ├── chatgpt/ # Direct ChatGPT usage docs

│ │ ├── gpt4-turbo/ # GPT-4 Turbo specific

│ │ ├── gpt4o/ # GPT-4o specific

│ │ └── o1-preview/ # o1-preview specific

│ ├── gemini/ # Direct Gemini usage docs

│ │ ├── 25-pro/ # 2.5 Pro specific

│ │ ├── 25-flash/ # 2.5 Flash specific

│ │ └── 15-pro/ # 1.5 Pro specific

│ └── custom/ # Custom/Enterprise agents

│ ├── aref-chatgpt41/ # Aref specific

│ ├── zakiagent-gemini25/ # ZakiAgent specific

│ └── enterprise-models/ # Enterprise specific

├── project/ # TIER 2: Project-Wide Documentation

│ ├── architecture/ # System architecture (project-wide)

│ ├── guides/ # User guides (project-wide)

│ ├── reference/ # API reference (project-wide)

│ ├── reports/ # Status reports (project-wide)

│ ├── standards/ # Documentation standards (project-wide)

│ └── plans/ # Migration plans (project-wide)

└── shared/ # Cross-tier shared documentation

├── templates/ # Document templates

├── standards/ # Shared standards

└── guidelines/ # Shared guidelines

```

##### Update History Table (Auto-managed)

```markdown

## Update History

| Version | Date | Agent/Model | Tier | Change Description | Status |

|---------|------------|-----------------------|-------------|----------------------|--------|

| 1.1.0 | 2025-06-09 | Cursor (Gemini 2.5 Pro) | Per-Agent | Added agent insights | Active |

| 1.0.0 | 2025-06-01 | Claude (Sonnet 4) | Project-Wide | Initial creation | Active |

```

##### Documentation Tier Guidelines

🏗️ \*\*TIER 1: Per-Agent Documentation\*\*

- Agent-specific learnings, patterns, and techniques

- Model-specific capabilities and limitations

- Agent workflow optimizations

- Integration-specific knowledge (e.g., Cursor IDE tips)

- Personal agent development history

🌐 \*\*TIER 2: Project-Wide Documentation\*\*

- System architecture and design

- API documentation and references

- User guides and tutorials

- Project status reports

- Cross-agent collaboration patterns

- Official project standards and policies

### 🚨 DOCUMENTATION ENFORCEMENT

\*\*NEVER start work without completing Phase 1 documentation check\*\*

\*\*NEVER finish work without completing Phase 3 documentation update\*\*

\*\*NEVER create documents without proper headers and versioning\*\*

\*\*NEVER use outdated dates or incorrect version formats\*\*

\*\*NEVER update documentation without AI agent tracking\*\* ⭐ \*\*NEW\*\*

\*\*NEVER use legacy doc-update.sh script (use doc-update-enhanced.sh)\*\* ⭐ \*\*NEW\*\*

\*\*ALWAYS use automation scripts for document creation/updates\*\*

\*\*ALWAYS specify the correct AI agent/model when updating docs\*\* ⭐ \*\*NEW\*\*

\*\*ALWAYS include change descriptions in Update History\*\* ⭐ \*\*NEW\*\*

\*\*ALWAYS validate documents before committing: ./doc-validate-simple.sh\*\*

\*\*ALWAYS reference existing documentation when solving similar problems\*\*

\*\*ALWAYS update lessons learned when encountering new issues\*\*

## 🏗️ DIRECTORY STRUCTURE RULES

# Maintain clean directory structure

- No empty directories without .gitkeep or README explaining purpose

- Follow naming conventions:

- snake\_case for Python modules/packages

- kebab-case for Docker/config files

- PascalCase for React components

- Group related files logically:

- /api/ - Backend API code only

- /agents/ - Individual agent implementations

- /docs/ - All documentation

- /scripts/ - Utility scripts

- /tests/ - All test files

## 🐍 PYTHON CODE RULES

# Prevent import and module issues

- All Python packages must have proper \_\_init\_\_.py files

- Use absolute imports for cross-module dependencies

- Add type hints for all function parameters and return values

- Include docstrings for all modules, classes, and functions

- Follow this module template:

```python

"""

Module: [module\_name]

Purpose: [Brief description]

Dependencies: [List key dependencies]

"""

from typing import Optional, Dict, List

import logging

logger = logging.getLogger(\_\_name\_\_)

# Module implementation here

```

## 🔧 CONFIGURATION RULES

# Prevent configuration conflicts

- Environment variables must be documented in .env.example

- No hardcoded URLs, ports, or credentials in code

- Use configuration classes or files for settings

- Validate all required environment variables at startup

- Follow this pattern:

```python

from pydantic import BaseSettings

class Settings(BaseSettings):

api\_url: str = "http://localhost:8000"

debug: bool = False

class Config:

env\_file = ".env"

```

## 🧪 TESTING RULES

# Ensure code quality and prevent regressions

- Every new module must have corresponding test file

- Test files follow naming: test\_[module\_name].py

- Include both unit tests and integration tests

- Mock external dependencies in tests

- Maintain minimum 80% code coverage

- Use this test template:

```python

"""

Tests for [module\_name]

"""

import pytest

from unittest.mock import Mock, patch

from your\_module import YourClass

class TestYourClass:

def test\_basic\_functionality(self):

# Test implementation

pass

```

## 📦 DEPENDENCY MANAGEMENT

# Prevent dependency conflicts and bloat

- Pin exact versions in requirements.txt for production

- Use requirements-dev.txt for development dependencies

- Document why each dependency is needed

- Regularly audit and remove unused dependencies

- Use virtual environments for all development

- Update dependencies systematically, not ad-hoc

## 🔍 CODE REVIEW CHECKLIST

Before committing, verify:

### 📚 Documentation Workflow (MANDATORY)

- [ ] \*\*Phase 1 completed\*\*: Pre-work documentation check performed

- [ ] \*\*Existing solutions reviewed\*\*: Checked docs/ for similar issues

- [ ] \*\*Status document created\*\*: Task documented in docs/reports/

- [ ] \*\*Phase 3 completed\*\*: Post-work documentation updated

- [ ] \*\*Lessons learned updated\*\*: New knowledge added to appropriate docs

- [ ] \*\*DOCUMENTATION.md updated\*\*: New documents added to index

- [ ] \*\*Standards compliance\*\*: Used proper header format with version/date/category

- [ ] \*\*Automation used\*\*: Created/updated docs with ./doc-new.sh or ./doc-update.sh

- [ ] \*\*Validation passed\*\*: Ran ./doc-validate.sh successfully on all modified docs

- [ ] \*\*Current dates\*\*: All dates reflect actual work completion (2025-01-09 or later)

### 💻 Code Quality

- [ ] No empty files without proper documentation

- [ ] No duplicate Docker service definitions

- [ ] All imports work correctly

- [ ] Tests pass and coverage is maintained

- [ ] No hardcoded configuration values

- [ ] Proper error handling and logging

- [ ] Type hints are present

- [ ] Code follows project conventions

### 📋 Documentation Quality

- [ ] All new documentation follows quality standards

- [ ] Cross-references updated between related documents

- [ ] Practical examples included where applicable

- [ ] Clear titles and purposes documented

- [ ] Date stamps added to new/updated documents

## 🚨 ANTI-PATTERNS TO AVOID

Never do these:

### 📚 Documentation Anti-Patterns

- ❌ \*\*Start work without documentation check\*\* (violates Phase 1)

- ❌ \*\*Finish work without updating lessons learned\*\* (violates Phase 3)

- ❌ \*\*Ignore existing solutions in documentation\*\*

- ❌ \*\*Create documentation without updating DOCUMENTATION.md\*\*

- ❌ \*\*Duplicate information across multiple files\*\*

- ❌ \*\*Add README files in deep subdirectories\*\*

- ❌ \*\*Skip status reporting for significant tasks\*\*

### 💻 Code Anti-Patterns

- ❌ Create docker-compose.[service].yml for existing services

- ❌ Commit empty .py files without documentation

- ❌ Use relative imports across packages

- ❌ Hardcode URLs, ports, or credentials

- ❌ Create placeholder directories without purpose

- ❌ Add dependencies without updating requirements.txt

- ❌ Skip tests for new functionality

- ❌ Use print() instead of proper logging

## ✅ BEST PRACTICES TO FOLLOW

Always do these:

### 📚 Documentation Best Practices

- ✅ \*\*Complete Phase 1 documentation check before starting work\*\*

- ✅ \*\*Search existing documentation for similar solutions\*\*

- ✅ \*\*Document your approach and reasoning\*\*

- ✅ \*\*Update lessons learned after solving problems\*\*

- ✅ \*\*Create completion reports for significant tasks\*\*

- ✅ \*\*Update DOCUMENTATION.md when adding new documents\*\*

- ✅ \*\*Use centralized documentation structure in docs/\*\*

- ✅ \*\*Cross-reference related documents\*\*

- ✅ \*\*Include practical examples in documentation\*\*

### 💻 Code Best Practices

- ✅ Check existing compose files before creating new ones

- ✅ Add meaningful docstrings and comments

- ✅ Follow established naming conventions

- ✅ Include proper error handling

- ✅ Write tests for new code

- ✅ Use configuration management

- ✅ Log important events and errors

- ✅ Validate inputs and handle edge cases

- ✅ Keep dependencies minimal and documented

## 🔧 AUTOMATED CHECKS

These should be enforced by CI/CD:

### 📚 Documentation Checks

- Verify DOCUMENTATION.md is updated when new docs are added

- Check that lessons learned files are updated for significant changes

- Validate that completion reports exist for major tasks

- Ensure all documentation files have proper headers and dates

- Check for broken cross-references between documents

### 💻 Code Quality Checks

- Lint all Python code with flake8/black

- Type check with mypy

- Run all tests and check coverage

- Validate Docker compose files

- Check for empty files without documentation

- Verify no duplicate service definitions

- Ensure all imports are resolvable

## 📋 MAINTENANCE TASKS

Regular maintenance (monthly):

### 📚 Documentation Maintenance

- [ ] \*\*Review lessons learned documents for accuracy and completeness\*\*

- [ ] \*\*Audit DOCUMENTATION.md for broken links and missing documents\*\*

- [ ] \*\*Consolidate duplicate information across documentation\*\*

- [ ] \*\*Update outdated documentation and remove obsolete files\*\*

- [ ] \*\*Review completion reports and extract reusable knowledge\*\*

- [ ] \*\*Ensure all major tasks have proper documentation\*\*

### 💻 System Maintenance

- [ ] Review and remove unused dependencies

- [ ] Check for empty files and add documentation

- [ ] Audit Docker configurations for duplicates

- [ ] Clean up temporary files and logs

- [ ] Validate all health checks work

- [ ] Update version pins in requirements.txt

## 🚦 CONSUL RESOURCE REGISTRY ENFORCEMENT (MANDATORY)

# All agents, scripts, and developers MUST:

- \*\*Before any deployment, configuration, or code change:\*\*

- Query Consul for the latest state of all system resources (services, ports, endpoints, modules, K/V configs)

- Verify resource availability: Ensure no port/service/module conflicts, and understand what is already present or in use

- Abort or warn if any conflict is found (port in use, service duplicate, module name collision)

- Suggest integration/reuse if resource already exists

- \*\*After any change:\*\*

- Register any new or changed resource in Consul via the HTTP API or supported agent/SDK

- Update K/V store entries to reflect module/service/version/owner/status

- Deregister obsolete or replaced resources in Consul

- Log update activity (who/what/when/what changed) to an audit trail or as a K/V entry if possible

# Example Consul API Calls

```bash

# Check all registered services

curl http://localhost:8500/v1/agent/services

# Register new service

curl -X PUT --data '{"ID": "my-new-service", "Name": "docker-dashboard", "Port": 8091}' http://localhost:8500/v1/agent/service/register

# Update module K/V entry

curl -X PUT -d 'active' http://localhost:8500/v1/kv/zaki/modules/docker-dashboard

# Deregister service

curl -X PUT http://localhost:8500/v1/agent/service/deregister/my-old-service

```

# This policy is enforced in all onboarding, deployment, and agent development workflows.

## 🏛️ Consul as Central Operations Registry (MANDATORY)

# All agents and services MUST use Consul as the central source of truth for:

# - Non-critical secrets (API keys, service tokens)

# - Dynamic configuration (feature flags, alert thresholds, environment configs)

# - Agent/service heartbeats and health state

# - Version and deployment metadata

# - Resource allocation and locking

# - Audit and event logging

# - Global/cluster settings (maintenance mode, default models)

# 1. Secrets Management (with Caution)

- Store non-critical API tokens/endpoints in `kv/zaki/secrets/{service}/{key}`.

- NEVER store root/admin/cloud credentials in Consul. Use HashiCorp Vault instead.

- See `docs/security/VAULT\_INTEGRATION.md` for critical secrets management.

# 2. Dynamic Configuration

- Store feature toggles in `kv/zaki/features/{feature}/enabled`.

- Store service configs in `kv/zaki/config/{service}/{key}`.

- Agents/services MUST read config from Consul at runtime.

# 3. Agent Heartbeats and Health

- Agents MUST write their status (alive, idle, error, last-seen) to `kv/zaki/agents/{agent\_name}/status`.

- This data is used for live dashboards, health checks, and auto-remediation.

# 4. Version and Deployment Metadata

- Track service/module version in `kv/zaki/modules/{name}/version`.

- Store deployment status/maintenance flags in `kv/zaki/modules/{name}/maintenance`.

# 5. Resource Allocation and Locking

- Register in-use ports/resources in `kv/zaki/ports/used`.

- Use Consul sessions/locks for coordinated deployments to prevent resource contention.

# 6. Audit and Event Logging

- Log all significant agent actions (register, update, error, deploy, scale) to `kv/zaki/audit/{timestamp}`.

- Include agent ID, action, and timestamp in all audit logs.

# 7. Global/Cluster Settings

- Store global settings in `kv/zaki/settings/{key}` (e.g., `maintenance = true`).

# Zaki-OS Documentation Standards

\*\*Version\*\*: 1.1.0

\*\*Date\*\*: 2025-06-09

\*\*Category\*\*: Standards/Guidelines

\*\*Status\*\*: Active

\*\*Last Updated\*\*: 2025-06-09T04:52:26Z

\*\*Last Updated By\*\*: Claude (Sonnet 3.5)

\*\*Auto-Generated\*\*: [true|false]

## 🎯 Overview

This document establishes the official documentation standards for Zaki-OS, including versioning, categorization, automated tooling, and quality requirements.

## 📋 Document Header Standard

Every documentation file MUST include this header:

```markdown

# [Document Title]

\*\*Version\*\*: [MAJOR.MINOR.PATCH]

\*\*Date\*\*: [YYYY-MM-DD]

\*\*Category\*\*: [Primary/Secondary]

\*\*Status\*\*: [Draft|Active|Deprecated|Archived]

\*\*Last Updated\*\*: [Auto-generated timestamp]

\*\*Last Updated By\*\*: [Agent/Model Name]

\*\*Auto-Generated\*\*: [true|false]

## Document Metadata

- \*\*Purpose\*\*: [Brief description of document purpose]

- \*\*Audience\*\*: [Primary audience: Developer|Admin|User|Stakeholder]

- \*\*Scope\*\*: [What this document covers]

- \*\*Dependencies\*\*: [Related documents or systems]

- \*\*Next Review\*\*: [YYYY-MM-DD]

## Update History

| Version | Date | Agent/Model | Change Description | Status |

|---------|------------|-----------------------|--------------------|--------|

| [X.Y.Z] | [YYYY-MM-DD] | [Agent/Model Name] | [Description of changes] | [Status] |

```

## Update History

| Version | Date | Agent/Model | Change Description | Status |

|---------|------------|-----------------------|--------------------|--------|

| 1.1.0 | 2025-06-09 | Claude (Sonnet 3.5) | Added AI agent/model tracking requirements to documentation standards | Active |

## 🔢 Versioning System

### Version Format: MAJOR.MINOR.PATCH

#### MAJOR Version (X.0.0)

- \*\*Breaking changes\*\* that require user action

- \*\*Complete restructuring\*\* of content

- \*\*Fundamental approach changes\*\*

- \*\*API/process incompatibilities\*\*

#### MINOR Version (0.X.0)

- \*\*New features or sections\*\* added

- \*\*Significant content additions\*\*

- \*\*Enhanced procedures\*\* or guidelines

- \*\*Backward-compatible improvements\*\*

#### PATCH Version (0.0.X)

- \*\*Bug fixes\*\* and corrections

- \*\*Minor clarifications\*\*

- \*\*Formatting improvements\*\*

- \*\*Link updates\*\*

- \*\*Typo corrections\*\*

### Initial Versions

- \*\*New documents\*\*: Start at `1.0.0`

- \*\*Migrated documents\*\*: Start at `2.0.0` (to indicate migration)

- \*\*Auto-generated\*\*: Start at `0.1.0`

## 📂 Document Categories

### Primary Categories

- \*\*Standards\*\*: Guidelines, policies, standards (this document)

- \*\*Architecture\*\*: System design, technical architecture

- \*\*Guides\*\*: Step-by-step instructions, tutorials

- \*\*Reference\*\*: API docs, configuration reference

- \*\*Reports\*\*: Status reports, completion reports

- \*\*Plans\*\*: Migration plans, roadmaps, strategies

### Secondary Categories (Optional)

- \*\*Implementation\*\*: Technical implementation details

- \*\*Migration\*\*: Migration-specific documentation

- \*\*Testing\*\*: Testing guides and procedures

- \*\*Deployment\*\*: Deployment and infrastructure

- \*\*Integration\*\*: Integration guides and examples

- \*\*Security\*\*: Security procedures and guidelines

### Category Usage

```markdown

\*\*Category\*\*: Architecture/Implementation

\*\*Category\*\*: Guides/Deployment

\*\*Category\*\*: Reports/Status

```

## 🏷️ Status Definitions

| Status | Description | Usage |

|--------|-------------|-------|

| \*\*Draft\*\* | Work in progress, not ready for use | New documents being written |

| \*\*Active\*\* | Current, accurate, ready for use | Production documentation |

| \*\*Deprecated\*\* | Outdated but still referenced | Being phased out |

| \*\*Archived\*\* | Historical, no longer relevant | Completed projects |

## 🤖 Automated Tooling

### Documentation Scripts Location

All documentation scripts are located in the root directory:

- `doc-new.sh` - Create new document with proper header

- `doc-update.sh` - Update existing document version/date

- `doc-validate.sh` - Validate documentation standards

- `doc-index.sh` - Regenerate documentation index

### Enhanced Documentation Scripts

- `doc-new.sh` - Create new document with proper header

- `doc-update.sh` - Update existing document version/date (legacy)

- `doc-update-enhanced.sh` - ⭐ \*\*NEW\*\*: Update with AI agent/model tracking

- `doc-validate-simple.sh` - Working validation script

- `doc-index.sh` - Regenerate documentation index

### AI Agent/Model Tracking ⭐ \*\*NEW FEATURE\*\*

All documentation updates MUST now include AI agent/model tracking:

#### Required Header Field

```markdown

\*\*Last Updated By\*\*: [Agent/Model Name]

```

#### Supported AI Agents/Models

##### 🔧 IDE/Editor Integrations

- \*\*Cursor (Copilot GPT-4)\*\* - Cursor IDE with GitHub Copilot GPT-4

- \*\*Cursor (Claude Sonnet 3.5)\*\* - Cursor IDE with Claude Sonnet 3.5

- \*\*Cursor (Gemini 2.5 Pro)\*\* - Cursor IDE with Gemini 2.5 Pro

- \*\*GitHub Copilot (GPT-4 Turbo)\*\* - GitHub Copilot with GPT-4 Turbo

- \*\*GitHub Copilot (Codex)\*\* - GitHub Copilot with Codex model

- \*\*Amazon Q (CodeWhisperer)\*\* - Amazon's AI coding assistant

- \*\*JetBrains AI (GPT-4)\*\* - JetBrains IDE AI with GPT-4

- \*\*Tabnine (Custom Models)\*\* - Tabnine with various models

##### 🤖 Direct AI Models

- \*\*Claude (Sonnet 4)\*\* - Anthropic Claude Sonnet 4 (current)

- \*\*Claude (Sonnet 3.5)\*\* - Anthropic Claude Sonnet 3.5

- \*\*Claude (Haiku 3.5)\*\* - Anthropic Claude Haiku 3.5 (fast)

- \*\*ChatGPT (GPT-4 Turbo)\*\* - OpenAI GPT-4 Turbo

- \*\*ChatGPT (GPT-4o)\*\* - OpenAI GPT-4o (multimodal)

- \*\*ChatGPT (o1-preview)\*\* - OpenAI o1-preview (reasoning)

- \*\*Gemini (2.5 Pro)\*\* - Google Gemini 2.5 Pro

- \*\*Gemini (2.5 Flash)\*\* - Google Gemini 2.5 Flash (fast)

- \*\*Gemini (1.5 Pro)\*\* - Google Gemini 1.5 Pro

##### 🏢 Custom/Enterprise

- \*\*Aref (ChatGPT-4.1)\*\* - Custom ChatGPT implementation

- \*\*ZakiAgent (Gemini 2.5)\*\* - Custom Gemini implementation

- \*\*Custom Enterprise Model\*\* - Organization-specific models

- \*\*Other/Custom\*\* - Any other AI system

#### Update History Table

Every document MUST include an Update History table:

```markdown

## Update History

| Version | Date | Agent/Model | Change Description | Status |

|---------|------------|-----------------------|--------------------|--------|

| 1.1.0 | 2025-06-09 | Claude (Sonnet 3.5) | Added AI tracking | Active |

| 1.0.0 | 2025-06-01 | Cursor (Copilot v2.1) | Initial creation | Active |

```

#### Usage Examples

```bash

# Update with AI agent tracking

./doc-update-enhanced.sh -a "Claude (Sonnet 3.5)" -d "Updated API docs" file.md

# Version bump with agent tracking

./doc-update-enhanced.sh -v minor -a "Cursor (Copilot v2.1)" -d "Added features" file.md

# Status change with agent tracking

./doc-update-enhanced.sh -s Active -a "ZakiAgent (Gemini 2.5)" file.md

```

### Auto-Generated Fields

The following fields are automatically managed:

```markdown

\*\*Last Updated\*\*: 2025-06-09T04:52:26Z

\*\*Last Updated By\*\*: Claude (Sonnet 3.5)

\*\*File Size\*\*: 15.2KB

\*\*Word Count\*\*: 2,847

\*\*Reading Time\*\*: ~11 minutes

\*\*Git Hash\*\*: a1b2c3d4

```

## 📝 Content Standards

### Required Sections

Every document MUST include:

1. \*\*Clear title\*\* and purpose statement

2. \*\*Proper header\*\* with metadata

3. \*\*Table of contents\*\* (for documents >500 words)

4. \*\*Examples\*\* where applicable

5. \*\*Related documentation\*\* links

6. \*\*Next steps\*\* or follow-up actions

### Optional Sections (when applicable)

- \*\*Quick Start\*\* (for guides)

- \*\*Prerequisites\*\* (for procedures)

- \*\*Troubleshooting\*\* (for technical docs)

- \*\*FAQ\*\* (for complex topics)

- \*\*Changelog\*\* (for evolving documents)

### Writing Guidelines

- \*\*Use clear, concise language\*\*

- \*\*Include practical examples\*\*

- \*\*Provide context and background\*\*

- \*\*Use consistent terminology\*\*

- \*\*Include screenshots for UI procedures\*\*

- \*\*Test all code examples\*\*

## 🔗 Cross-Reference Standards

### Internal Links

```markdown

[Text](relative/path/to/doc.md)

[API Gateway Reference](docs/reports/API\_GATEWAY\_SUMMARY.md)

```

### External Links

```markdown

[Text](https://external-url.com)

[FastAPI Documentation](https://fastapi.tiangolo.com/)

```

### Document Collections

```markdown

\*\*Related Documentation\*\*:

- [Doc 1](path/to/doc1.md) - Brief description

- [Doc 2](path/to/doc2.md) - Brief description

```

## 🏗️ File Naming Conventions

### File Names

- \*\*Use UPPERCASE\*\* for major documents: `README.md`, `DOCUMENTATION.md`

- \*\*Use sentence\_case\*\* for specific docs: `api\_gateway\_summary.md`

- \*\*Use kebab-case\*\* for guides: `installation-guide.md`

- \*\*Include category prefix\*\* when helpful: `arch\_system\_design.md`

### Directory Structure

```

docs/

├── standards/ # Documentation standards and guidelines

├── architecture/ # System architecture documents

├── guides/ # User and developer guides

├── reference/ # API and configuration reference

├── reports/ # Status and completion reports

├── plans/ # Migration plans and roadmaps

└── templates/ # Document templates

```

## 🔍 Quality Assurance

### Validation Checklist

- [ ] \*\*Header complete\*\* with all required fields

- [ ] \*\*Version number\*\* follows semantic versioning

- [ ] \*\*Date accurate\*\* and current

- [ ] \*\*Category appropriate\*\* and consistent

- [ ] \*\*Cross-references valid\*\* and working

- [ ] \*\*Examples tested\*\* and functional

- [ ] \*\*Grammar and spelling\*\* checked

- [ ] \*\*Index updated\*\* if new document

### Review Requirements

- \*\*Technical accuracy\*\*: Subject matter expert review

- \*\*Documentation standards\*\*: Standards compliance check

- \*\*User experience\*\*: Readability and clarity review

- \*\*Link validation\*\*: All links tested and functional

## ⚙️ Automation Integration

### Git Hooks

- \*\*Pre-commit\*\*: Validate documentation standards

- \*\*Post-commit\*\*: Update last modified dates

- \*\*Pre-push\*\*: Regenerate documentation index

### CI/CD Integration

```yaml

documentation\_checks:

runs-on: ubuntu-latest

steps:

- name: Validate Documentation

run: ./doc-validate.sh

- name: Check Broken Links

run: ./doc-links-check.sh

- name: Update Index

run: ./doc-index.sh

```

## 📊 Metrics and KPIs

### Documentation Health Metrics

- \*\*Standards Compliance\*\*: % of docs following standards

- \*\*Freshness\*\*: Average age of last update

- \*\*Coverage\*\*: % of features with documentation

- \*\*Link Health\*\*: % of working internal links

- \*\*User Satisfaction\*\*: Feedback scores

### Monthly Reviews

- \*\*Outdated documents\*\* (>90 days old)

- \*\*Broken links\*\* and references

- \*\*Missing documentation\*\* for new features

- \*\*User feedback\*\* and improvement suggestions

## 🚀 Migration Plan

### Phase 1: Foundation (Week 1)

- [ ] Deploy automation scripts

- [ ] Update cursor rules

- [ ] Create document templates

- [ ] Establish validation pipeline

### Phase 2: Content Migration (Week 2-3)

- [ ] Update all existing documents with proper headers

- [ ] Assign appropriate versions and categories

- [ ] Validate all cross-references

- [ ] Regenerate master index

### Phase 3: Enforcement (Week 4+)

- [ ] Enable automated validation

- [ ] Train team on new standards

- [ ] Monitor compliance metrics

- [ ] Iterate based on feedback

## 📋 Templates

### Document Templates Available

- `templates/standard\_document.md` - Basic document template

- `templates/technical\_guide.md` - Technical guide template

- `templates/api\_reference.md` - API documentation template

- `templates/status\_report.md` - Status report template

- `templates/migration\_plan.md` - Migration plan template

### Usage

```bash

# Create new document from template

./doc-new.sh --template=guide --title="Installation Guide" --category="Guides/Setup"

```

## 🔄 Maintenance

### Regular Tasks

- \*\*Weekly\*\*: Check for outdated documents

- \*\*Monthly\*\*: Validate all links and references

- \*\*Quarterly\*\*: Review and update standards

- \*\*Annually\*\*: Archive completed project documentation

### Responsibility Matrix

| Role | Responsibility |

|------|----------------|

| \*\*Developers\*\* | Create and maintain technical documentation |

| \*\*Tech Leads\*\* | Review and approve documentation changes |

| \*\*DevOps\*\* | Maintain automation and tooling |

| \*\*Project Manager\*\* | Ensure documentation completeness |

## ✅ Compliance

### Required for All Documents

- ✅ \*\*Proper header\*\* with all metadata fields

- ✅ \*\*Semantic versioning\*\* correctly applied

- ✅ \*\*Current dates\*\* and timestamps

- ✅ \*\*Appropriate categorization\*\*

- ✅ \*\*Working cross-references\*\*

- ✅ \*\*Clear purpose\*\* and scope

- ✅ \*\*Quality review\*\* completed

### Enforcement

- \*\*Automated validation\*\* in CI/CD pipeline

- \*\*Manual review\*\* for significant changes

- \*\*Compliance dashboard\*\* for tracking

- \*\*Training\*\* for team members

---

\*\*This document establishes the foundation for professional, maintainable, and user-friendly documentation across the Zaki-OS project.\*\*

\*\*Next Steps\*\*: Deploy automation scripts and begin migration of existing documentation.

# Zaki-OS Documentation Update & Modernization Plan

\*\*Version\*\*: 1.0.0

\*\*Date\*\*: 2025-06-09

\*\*Category\*\*: Plans/Documentation

\*\*Status\*\*: Active

\*\*Last Updated\*\*: 2025-06-09T04:19:34Z

\*\*Auto-Generated\*\*: false

## Document Metadata

- \*\*Purpose\*\*: Comprehensive plan for updating, standardizing, and modernizing all Zaki-OS documentation

- \*\*Audience\*\*: Developer|Admin|Stakeholder

- \*\*Scope\*\*: All documentation files in the project

- \*\*Dependencies\*\*: DOC\_STANDARDS.md, automation scripts (doc-new.sh, doc-update.sh, doc-validate.sh)

- \*\*Next Review\*\*: 2025-09-07

- \*\*Author\*\*: Zaki-OS Documentation Team

- \*\*Git Hash\*\*: unknown

## 🎯 Executive Summary

This plan addresses critical documentation issues identified in the Zaki-OS project:

- \*\*43+ documents\*\* with outdated dates (ranging from 2024-01-05 to 2025-01-27)

- \*\*Inconsistent versioning\*\* and header formats

- \*\*Missing automation compliance\*\* for document creation/updates

- \*\*Potential outdated content\*\* requiring review and archival

## 📊 Current Documentation Status Analysis

### 🚨 Critical Issues Identified

#### Outdated Date Stamps

- \*\*Most Critical\*\*: Documents dated 2024 (20+ files) need immediate updating

- \*\*Medium Priority\*\*: Documents with future dates (2025-01-27) need correction

- \*\*Format Issues\*\*: Some dates use non-standard formats ("June 8, 2024")

#### Missing Standard Headers

- \*\*Template Violations\*\*: Documents not following DOC\_STANDARDS.md format

- \*\*Missing Metadata\*\*: Version numbers, categories, status fields

- \*\*Inconsistent Automation\*\*: Not all docs use automation scripts

#### Content Quality Concerns

- \*\*Outdated Information\*\*: References to old system versions

- \*\*Broken Cross-References\*\*: Links to moved or deleted documents

- \*\*Duplicate Content\*\*: Multiple documents covering same topics

## 🗂️ Documentation Inventory & Classification

### 📂 Documents by Update Priority

#### \*\*Priority 1: IMMEDIATE UPDATE REQUIRED\*\* ⚠️

\*Documents with 2024 dates requiring immediate correction\*

1. \*\*MCP/A2A Investigation Series\*\* (2024-12-28)

- `docs/reports/MCP\_A2A\_INVESTIGATION\_REPORT.md`

- `docs/reports/MCP\_A2A\_EXECUTIVE\_SUMMARY.md`

- `docs/reports/API\_GATEWAY\_SUMMARY.md`

- `docs/reports/GATEWAY\_MCP\_A2A\_MODERNIZATION\_COMPLETE.md`

2. \*\*System Implementation Reports\*\* (2024-12-28)

- `docs/reports/RESOURCE\_CENTER\_MIGRATION\_COMPLETE.md`

- `docs/reports/DOCUMENTATION\_REORGANIZATION\_COMPLETE.md`

- `docs/reports/CROSS\_SERVICE\_MESSAGING\_COMPLETE.md`

3. \*\*Legacy Development Reports\*\* (2024-01-XX)

- `docs/DEVELOPMENT\_LESSONS\_LEARNED.md` (2024-01-15)

- `docs/SYSTEM\_LESSONS\_LEARNED.md` (2024-01-12)

- `docs/reports/DOCUMENTATION\_WORKFLOW\_IMPLEMENTATION\_COMPLETE.md` (2024-01-15)

#### \*\*Priority 2: VALIDATION & STANDARDIZATION\*\* 📋

\*Documents needing header format updates\*

1. \*\*Architecture Documents\*\*

- `docs/architecture/UNIFIED\_API\_GATEWAY\_DESIGN.md`

- `docs/ARCHITECTURE.md`

2. \*\*Administrative Handbooks\*\*

- `docs/ADMIN\_HANDBOOK.md`

- `docs/DEV\_HANDBOOK.md`

#### \*\*Priority 3: CONTENT REVIEW & ARCHIVAL\*\* 🗄️

\*Documents requiring content validation\*

1. \*\*Potentially Outdated\*\*

- `docs/reports/CONTAINER\_STATUS\_FINAL\_REPORT.md` (June 8, 2024)

- `docs/reports/GATEWAY\_SERVICE\_COMPLETION\_REPORT.md` (2024-06-09)

2. \*\*Duplicate/Redundant Content\*\*

- Multiple gateway implementation reports

- Similar migration status documents

### 📊 Document Categories Distribution

| Category | Count | Status | Action Required |

|----------|-------|--------|-----------------|

| Reports | 25+ | Mixed | Update dates, standardize headers |

| Architecture | 3 | Stable | Validate content, update headers |

| Guides | 5 | Active | Minor updates needed |

| Standards | 2 | Current | Minimal updates |

| Lessons Learned | 3 | Active | Update dates, add recent lessons |

## 🔄 Implementation Phases

### \*\*Phase 1: Critical Date Updates\*\* (Day 1-2)

\*\*Objective\*\*: Fix all 2024 dates to current system time

#### Actions:

1. \*\*Automated Date Correction\*\*

```bash

# Use doc-update.sh for all 2024-dated files

find docs/ -name "\*.md" -exec grep -l "\*\*Date\*\*: 2024" {} \; | \

xargs -I {} ./doc-update.sh --force-date {}

```

2. \*\*Manual Review Required\*\*

- Documents with complex date references

- Files requiring content validation

- Cross-reference validation

#### Expected Outcomes:

- ✅ All documents have current dates (2025-06-09)

- ✅ Proper timestamp formatting

- ✅ Validated automation script compliance

### \*\*Phase 2: Header Standardization\*\* (Day 3-4)

\*\*Objective\*\*: Ensure all documents follow DOC\_STANDARDS.md format

#### Actions:

1. \*\*Header Compliance Check\*\*

```bash

# Validate all documentation against standards

find docs/ -name "\*.md" -exec ./doc-validate.sh {} \;

```

2. \*\*Missing Header Addition\*\*

- Generate proper headers for non-compliant documents

- Add required metadata fields

- Implement semantic versioning

3. \*\*Automation Integration\*\*

- Ensure all future updates use doc-update.sh

- Validate doc-new.sh for new documents

- Test doc-validate.sh workflow

#### Expected Outcomes:

- ✅ 100% header compliance

- ✅ Consistent versioning scheme

- ✅ Automated validation passing

### \*\*Phase 3: Content Review & Quality Assurance\*\* (Day 5-7)

\*\*Objective\*\*: Validate content accuracy and relevance

#### Actions:

1. \*\*Content Accuracy Review\*\*

- Technical content validation

- System version references update

- Link validation and repair

2. \*\*Cross-Reference Audit\*\*

```bash

# Check for broken internal links

./doc-validate.sh --check-links docs/

```

3. \*\*Duplicate Content Resolution\*\*

- Identify redundant documents

- Consolidate similar content

- Archive obsolete documents

#### Expected Outcomes:

- ✅ Accurate technical content

- ✅ Valid cross-references

- ✅ Eliminated redundancy

### \*\*Phase 4: Archive & Cleanup\*\* (Day 8-9)

\*\*Objective\*\*: Archive outdated content and optimize structure

#### Actions:

1. \*\*Document Archival Strategy\*\*

```

docs/

├── archive/

│ ├── 2024/ # Historical documents

│ ├── deprecated/ # Outdated but referenced

│ └── superseded/ # Replaced by newer versions

```

2. \*\*Archive Candidates\*\*

- Pre-2025 implementation reports (completed projects)

- Superseded architecture documents

- Outdated migration plans

3. \*\*Documentation Index Update\*\*

- Update DOCUMENTATION.md with current structure

- Add archive references

- Implement redirect patterns

#### Expected Outcomes:

- ✅ Clean documentation structure

- ✅ Archived historical content

- ✅ Updated documentation index

### \*\*Phase 5: Automation & Maintenance\*\* (Day 10)

\*\*Objective\*\*: Establish sustainable documentation maintenance

#### Actions:

1. \*\*Automation Verification\*\*

- Test all automation scripts

- Validate workflow integration

- Create maintenance procedures

2. \*\*Quality Gates Implementation\*\*

- Pre-commit hooks for documentation

- Automated validation in CI/CD

- Regular maintenance scheduling

3. \*\*Training & Documentation\*\*

- Update development workflows

- Create documentation best practices guide

- Establish review procedures

#### Expected Outcomes:

- ✅ Automated documentation workflow

- ✅ Quality assurance processes

- ✅ Maintenance procedures

## 🛠️ Implementation Scripts & Commands

### \*\*Automated Date Updates\*\*

```bash

# Find and update all 2024-dated documents

find docs/ -name "\*.md" -exec grep -l "\*\*Date\*\*: 2024" {} \; | while read file; do

echo "Updating: $file"

./doc-update.sh --force-date "$file"

done

```

### \*\*Header Standardization\*\*

```bash

# Validate documentation standards

find docs/ -name "\*.md" -exec ./doc-validate.sh --verbose {} \;

# Fix common issues automatically

find docs/ -name "\*.md" -exec ./doc-validate.sh --fix {} \;

```

### \*\*Content Quality Check\*\*

```bash

# Check for broken links

find docs/ -name "\*.md" -exec ./doc-validate.sh --check-links {} \;

# Identify TODO items

grep -r "TODO\|FIXME\|XXX" docs/ --include="\*.md"

```

### \*\*Archive Creation\*\*

```bash

# Create archive structure

mkdir -p docs/archive/{2024,deprecated,superseded}

# Move historical documents

mv docs/reports/\*2024\* docs/archive/2024/

```

## 📋 Quality Assurance Checklist

### \*\*Document Standards Compliance\*\*

- [ ] All documents have proper headers with Version, Date, Category, Status

- [ ] Dates use current system time (2025-06-09 or later)

- [ ] Semantic versioning applied consistently

- [ ] Document metadata sections complete

- [ ] Automation scripts used for all updates

### \*\*Content Quality\*\*

- [ ] Technical accuracy validated by subject matter experts

- [ ] Cross-references working and current

- [ ] Examples tested and functional

- [ ] No outdated system references

- [ ] Clear purpose and audience defined

### \*\*Structure & Organization\*\*

- [ ] Logical directory structure maintained

- [ ] No duplicate content

- [ ] Archive structure implemented

- [ ] Documentation index updated

- [ ] Search and discovery optimized

### \*\*Automation & Maintenance\*\*

- [ ] All scripts tested and working

- [ ] Validation passing for all documents

- [ ] Maintenance procedures documented

- [ ] Quality gates implemented

- [ ] Regular review schedule established

## 🎯 Success Metrics

### \*\*Completion Criteria\*\*

1. \*\*100% Date Compliance\*\*: All documents dated 2025-06-09 or later

2. \*\*100% Header Compliance\*\*: All documents follow DOC\_STANDARDS.md

3. \*\*Zero Broken Links\*\*: All internal references working

4. \*\*Clean Archive\*\*: Historical content properly organized

5. \*\*Automated Validation\*\*: All documents pass doc-validate.sh

### \*\*Maintenance KPIs\*\*

- \*\*Documentation Freshness\*\*: Average age < 90 days

- \*\*Validation Score\*\*: 100% passing doc-validate.sh

- \*\*Cross-Reference Health\*\*: 100% working links

- \*\*Update Compliance\*\*: 100% use of automation scripts

- \*\*Quality Review\*\*: Regular expert validation

## 🔗 Related Documentation

- [DOC\_STANDARDS.md](../DOC\_STANDARDS.md) - Documentation standards reference

- [DOCUMENTATION.md](../../DOCUMENTATION.md) - Main documentation index

- [CONTAINER\_LESSONS\_LEARNED.md](../CONTAINER\_LESSONS\_LEARNED.md) - Implementation lessons

- [SYSTEM\_LESSONS\_LEARNED.md](../SYSTEM\_LESSONS\_LEARNED.md) - System operation insights

## 🚀 Next Steps

### \*\*Immediate Actions\*\* (Today)

1. \*\*Execute Phase 1\*\*: Update all 2024 dates to current system time

2. \*\*Run Validation\*\*: Execute doc-validate.sh on all documents

3. \*\*Create Archive Structure\*\*: Establish docs/archive/ hierarchy

### \*\*This Week\*\*

1. \*\*Complete Header Standardization\*\*: Ensure 100% compliance

2. \*\*Content Review\*\*: Validate technical accuracy

3. \*\*Archive Implementation\*\*: Move historical documents

### \*\*Ongoing\*\*

1. \*\*Automation Integration\*\*: Implement quality gates

2. \*\*Maintenance Schedule\*\*: Establish regular review cycles

3. \*\*Training\*\*: Update development workflows

---

\*\*Implementation Start\*\*: 2025-06-09

\*\*Target Completion\*\*: 2025-06-19

\*\*Maintenance\*\*: Ongoing with quarterly reviews

\*\*Support\*\*: Documentation team via development workflow

---

title: "Brief, Searchable Title of the Lesson"

date: "YYYY-MM-DD"

sidebar\_label: "Short Label for Sidebar"

tags:

- keyword1

- keyword2

- keyword3

author: "Agent Name (Model)"

status: "Active" # Active | Deprecated | Archived

severity: "Medium" # Critical | High | Medium | Low

category: "Process" # OS | Docker | Services | Language | Database | Process

subcategory: "General" # e.g., Ubuntu, Networking, Agent-Studio, Python-Imports, Postgres-Connections

related\_resources:

- "/docs/path/to/related/doc.md"

- "/path/to/config/file.yml"

---

## 1. Issue Summary

A one-sentence summary of the problem that occurred.

<!--

This section should be extremely concise and is the primary text that will appear in search result snippets.

-->

---

## 2. Context & Symptoms

\* \*\*Service/Module\*\*: `Specify the service, module, or component involved`

\* \*\*Action Being Performed\*\*: `What was the goal? (e.g., "Deploying Agent Studio v2.1")`

\* \*\*Expected Outcome\*\*: `What should have happened? (e.g., "The container should start and be healthy.")`

\* \*\*Actual Outcome & Symptoms\*\*: `What actually happened? (e.g., "The container entered a restart loop with exit code 137.")`

---

## 3. Root Cause Analysis

A clear, concise explanation of the underlying cause of the issue. This section should answer "why" the problem occurred.

\*(Example: The Docker container was allocated only 512MB of memory in `docker-compose.yml`, but the application's startup process temporarily required 600MB, causing the kernel's OOM (Out Of Memory) killer to terminate it.)\*

---

## 4. Resolution

The specific, step-by-step solution that was implemented to fix the immediate problem.

```bash

# Example: Include code blocks for commands that were run

# 1. Open the docker-compose.apps.yml file.

# 2. Update the memory limit for the 'agent-studio' service.

# 3. Re-deploy the stack.

docker-compose -f docker-compose.apps.yml up -d --force-recreate agent-studio

```

---

## 5. Preventive Measures & Heuristics for Agents

This is the most critical section for system evolution. It provides actionable rules for the future.

### For AI Agents (Heuristics)

\* \*\*Rule\*\*: If a container fails with `exit code 137` (OOMKilled), first check the memory limits in the `docker-compose.yml` file before inspecting application code.

\* \*\*Heuristic\*\*: When creating a new service, set a default memory reservation of at least `512M` and a limit of `1G` unless the application is known to be very lightweight.

### For Human Developers (Best Practices)

\* \*\*Guideline\*\*: Always monitor memory usage of a new service during its first deployment to establish a baseline.

\* \*\*Checklist Item\*\*: Add "Review service memory allocation" to the deployment checklist.

# AI-Powered Knowledge Management System: Project Plan

\*\*Version\*\*: 1.0.0

\*\*Date\*\*: 2025-06-10

\*\*Category\*\*: Primary

\*\*Status\*\*: Draft

\*\*Last Updated\*\*: 2025-06-10T12:05:00Z

\*\*Auto-Generated\*\*: true

## Document Metadata

- \*\*Purpose\*\*: To provide a step-by-step plan for implementing the AI-powered documentation and knowledge management system.

- \*\*Audience\*\*: Project Managers, Developers

- \*\*Scope\*\*: Covers the entire project lifecycle from setup to deployment and monitoring.

- \*\*Dependencies\*\*: `AI\_DOC\_SYSTEM\_COMPONENT\_ANALYSIS.md`

- \*\*Next Review\*\*: 2025-06-17

---

## Project Overview

This project will establish a self-updating, AI-driven knowledge management system by integrating existing services. The goal is to create a seamless pipeline that ingests Markdown documentation, enriches it with AI-generated metadata, and makes it searchable and accessible through both a static site and an API.

## Phase 1: Foundation Setup & Ingestion Pipeline (1-2 Days)

This phase focuses on setting up the core data flow, from detecting file changes to indexing the raw content in Elasticsearch.

\* \*\*Step 1.1: Configure Elasticsearch Index Template\*\*

\* \*\*Goal:\*\* Create a robust mapping for the documentation index.

\* \*\*Action:\*\* Define an Elasticsearch index template (`docs-template.json`) that specifies fields for:

\* `path`: The full path to the document.

\* `content`: The full-text content of the document.

\* `summary`: Text field for the AI-generated summary.

\* `tags`: Keyword field for AI-generated tags.

\* `embedding`: `dense\_vector` field for semantic search vectors.

\* `last\_modified`: Date field.

\* \*\*Tooling:\*\* Use Kibana Dev Tools to apply the template.

\* \*\*Step 1.2: Configure Logstash for Document Ingestion\*\*

\* \*\*Goal:\*\* Automatically ingest `.md` files into Elasticsearch.

\* \*\*Action:\*\* Create a new Logstash pipeline configuration (`logstash-docs.conf`):

\* \*\*Input:\*\* Use the `file` input plugin to watch the `docs/` directory recursively for `.md` files. Set `schedule` to run every 1-2 hours.

\* \*\*Filter:\*\* Use the `mutate` filter to add metadata like the file path.

\* \*\*Output:\*\* Use the `elasticsearch` output plugin to send the data to the `zaki-docs` index.

\* \*\*Tooling:\*\* Logstash configuration files, Docker volume mounts.

\* \*\*Step 1.3: Initial Document Ingestion\*\*

\* \*\*Goal:\*\* Perform a one-time import of all existing documentation.

\* \*\*Action:\*\* Run the Logstash pipeline to populate the Elasticsearch index with all current `.md` files.

\* \*\*Verification:\*\* Use Kibana to verify that documents are being indexed correctly.

## Phase 2: AI Enrichment Workflow (2-3 Days)

This phase introduces the AI capabilities, orchestrating the process of summarizing, tagging, and embedding the documents.

\* \*\*Step 2.1: Design the n8n Orchestration Workflow\*\*

\* \*\*Goal:\*\* Create an n8n workflow to manage the AI enrichment process.

\* \*\*Action:\*\* Design a workflow with the following nodes:

\* \*\*Trigger:\*\* Use a `Schedule` node to run the workflow periodically (e.g., every hour).

\* \*\*Elasticsearch Query:\*\* Get documents from the `zaki-docs` index that haven't been processed yet (e.g., where the `summary` field is null).

\* \*\*HTTP Request (Ollama):\*\* For each document, call the Ollama API to:

1. Generate a summary.

2. Generate relevant tags.

3. Generate vector embeddings.

\* \*\*Elasticsearch Update:\*\* Use the `Elasticsearch` node to update the document with the new summary, tags, and embedding vector.

\* \*\*Tooling:\*\* n8n Workflow Editor.

\* \*\*Step 2.2: Implement Ollama API Calls\*\*

\* \*\*Goal:\*\* Configure the HTTP requests to interact with the local LLM.

\* \*\*Action:\*\* In the n8n workflow, create and test the `HTTP Request` nodes. Use appropriate prompts for summarization and tagging. Ensure the model for embeddings (e.g., `nomic-embed-text`) is pulled and available in Ollama.

\* \*\*Tooling:\*\* n8n, Ollama REST API.

\* \*\*Step 2.3: Implement Error Handling\*\*

\* \*\*Goal:\*\* Make the workflow robust.

\* \*\*Action:\*\* Add error handling paths in the n8n workflow to catch API failures from Ollama or update failures in Elasticsearch. Send alerts to a dedicated channel (e.g., Slack) or log them to a separate index.

\* \*\*Tooling:\*\* n8n error triggers.

## Phase 3: Access & Visualization (2-3 Days)

This phase focuses on making the enriched documentation accessible to users through a web interface and API.

\* \*\*Step 3.1: Set Up Static Site Generation\*\*

\* \*\*Goal:\*\* Create a browseable documentation website.

\* \*\*Action:\*\*

1. Choose a static site generator (`MkDocs` is recommended).

2. Create a `mkdocs.yml` configuration file at the root of the `docs/` directory.

3. Create a simple Python script (`build\_docs.py`) that runs `mkdocs build`.

4. Create a Dockerfile (`Dockerfile.docs`) that runs this Python script and serves the generated `site/` directory with Nginx.

\* \*\*Tooling:\*\* `MkDocs`, Python, Docker, Nginx.

\* \*\*Step 3.2: Automate Site Deployment\*\*

\* \*\*Goal:\*\* Keep the documentation site up-to-date automatically.

\* \*\*Action:\*\* Add a step to the main `n8n` workflow (or a separate one) that triggers a rebuild and deployment of the documentation container whenever significant changes are processed. This can be done via a webhook that triggers a shell script on the host.

\* \*\*Tooling:\*\* n8n, Docker.

\* \*\*Step 3.3: Build Kibana Search Dashboards\*\*

\* \*\*Goal:\*\* Provide a powerful interface for searching and analyzing documents.

\* \*\*Action:\*\* In Kibana, create a new dashboard that includes:

\* A search bar for full-text queries.

\* Controls to filter by tags.

\* Visualizations showing document counts by category.

\* \*\*Tooling:\*\* Kibana Dashboard.

\* \*\*Step 3.4: (Optional) Create a Natural Language Query API\*\*

\* \*\*Goal:\*\* Allow users to ask questions in natural language.

\* \*\*Action:\*\*

1. Create a simple FastAPI service.

2. The API endpoint will take a user's question as input.

3. The service will first generate an embedding for the question using Ollama.

4. It will then construct a hybrid Elasticsearch query combining a `match` query for keywords and a `knn` query for semantic similarity.

5. Return the top search results.

\* \*\*Tooling:\*\* FastAPI, Python, Docker.

## Phase 4: Monitoring & Security (1 Day)

This final phase ensures the system is observable and secure.

\* \*\*Step 4.1: Create Grafana Dashboards\*\*

\* \*\*Goal:\*\* Monitor the health and performance of the entire system.

\* \*\*Action:\*\* Build a Grafana dashboard with panels for:

\* \*\*Logstash:\*\* Ingestion rate, errors.

\* \*\*n8n:\*\* Workflow execution counts (success/failure).

\* \*\*Ollama:\*\* API latency, requests per second.

\* \*\*Elasticsearch:\*\* Indexing rate, query latency.

\* \*\*Tooling:\*\* Grafana, Prometheus (for metrics).

\* \*\*Step 4.2: Configure Security and Access\*\*

\* \*\*Goal:\*\* Secure all endpoints.

\* \*\*Action:\*\* Configure the reverse proxy (e.g., Traefik, Nginx) to place Kibana, Grafana, n8n, and the new documentation site behind an authentication middleware (like Authelia, which you have).

\* \*\*Tooling:\*\* Reverse Proxy configuration.

\* \*\*Step 4.3: Set Up Alerting\*\*

\* \*\*Goal:\*\* Be notified of system failures.

\* \*\*Action:\*\* Use Grafana or n8n to set up alerts for critical issues, such as:

\* The ingestion pipeline failing for more than 3 hours.

\* Ollama API becoming unresponsive.

\* A high rate of search query failures.

\* \*\*Tooling:\*\* Grafana Alerting, n8n.

Fils to keep

archive\_20240611/ (old archive)

config\_backups\_2025-06-12/

.git/ (version control)

docs/ (documentation)

Move under dashboards

security\_dashboard/ (likely monitoring/security)

Zaki-services-dashboard.html

Move to code folder:

apps/ (contains your core application code)

scripts/ (utility scripts)

services/ (contains service implementations)

c2-center/ (appears to be application code)

Move to docs:

Documentation files (can be moved to docs/ or backed up):

COMPLIANCE\_VERIFICATION\_REPORT.md

MODULES\_FEATURES\_OVERVIEW.md

ZAKI\_OS\_404\_TROUBLESHOOTING\_REPORT.md

Completely delete

docker-compose.yml

docker-compose.qdrant.yml

authelia/ (authentication service)

Investigate further if all content is from ready services software used before you can delete

platform/