Execute Task: 3. Create configuration management system 3. Create configuration management system

**Kiro**

I'll implement task 3 to create the configuration management system. Let me start by updating the task status and then implementing the required components.

Task: *3. Create configuration management system* from tasks.md

Status: In Progress

Now I'll implement the ConfigManager class in core/config\_manager.py:

Accepted edits to

config\_manager.py

Now I'll create the default config.yaml file with all required sections and example values:

Accepted edits to

config.yaml

Now I'll create comprehensive unit tests for the configuration loading and validation:

Created

test\_config\_manager.py

Now let me run the tests to verify the implementation works correctly:

Command

python -m pytest tests/test\_config\_manager.py -v

(.venv) PS E:\Forschung\KIRO\PM Assistant> python -m pytest tests/test\_conf

ig\_manager.py -v ========================== test session starts ===========================

platform win32 -- Python 3.10.11, pytest-8.4.1, pluggy-1.6.0 -- E:\Forschun

g\KIRO\PM Assistant\.venv\Scripts\python.exe cachedir: .pytest\_cache

rootdir: E:\Forschung\KIRO\PM Assistant

configfile: pytest.ini

plugins: cov-6.2.1

collected 22 items

tests/test\_config\_manager.py::TestConfigManager::test\_init\_default\_config\_p

ath PASSED [ 4%] tests/test\_config\_manager.py::TestConfigManager::test\_init\_custom\_config\_pa

th PASSED [ 9%] tests/test\_config\_manager.py::TestConfigManager::test\_load\_config\_success P

ASSED [ 13%] tests/test\_config\_manager.py::TestConfigManager::test\_load\_config\_file\_not\_

found\_creates\_default PASSED [ 18%] tests/test\_config\_manager.py::TestConfigManager::test\_load\_config\_invalid\_y

aml PASSED [ 22%] tests/test\_config\_manager.py::TestConfigManager::test\_validate\_config\_missi

ng\_sections PASSED [ 27%] tests/test\_config\_manager.py::TestConfigManager::test\_validate\_config\_missi

ng\_project\_fields PASSED [ 31%] tests/test\_config\_manager.py::TestConfigManager::test\_validate\_config\_missi

ng\_mode\_fields PASSED [ 36%] tests/test\_config\_manager.py::TestConfigManager::test\_validate\_required\_doc

uments\_invalid\_structure PASSED [ 40%] tests/test\_config\_manager.py::TestConfigManager::test\_validate\_required\_doc

uments\_invalid\_types PASSED [ 45%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_project\_config PA

SSED [ 50%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_required\_document

s PASSED [ 54%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_modes\_config PASS

ED [ 59%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_output\_config PAS

SED [ 63%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_logging\_config PA

SSED [ 68%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_project\_path PASS

ED [ 72%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_project\_path\_defa

ult PASSED [ 77%] tests/test\_config\_manager.py::TestConfigManager::test\_is\_mode\_enabled\_true

PASSED [ 81%] tests/test\_config\_manager.py::TestConfigManager::test\_is\_mode\_enabled\_false

PASSED [ 86%]

tests/test\_config\_manager.py::TestConfigManager::test\_is\_mode\_enabled\_missi

ng\_mode PASSED [ 90%] tests/test\_config\_manager.py::TestConfigManager::test\_create\_default\_config

\_io\_error PASSED [ 95%] tests/test\_config\_manager.py::TestConfigManager::test\_get\_default\_config\_da

ta\_structure PASSED [100%]

=========================== 22 passed in 0.34s ===========================

Excellent! All tests are passing. Let me also test that the configuration loading works with the actual config.yaml file:

Command

python -c "from core.config\_manager import ConfigManager; cm = ConfigManager(); config = cm.load\_config(); print('Config loaded successfully!'); print(f'Project name: {config[\"project\"][\"name\"]}'); print(f'Required docs count: {len(config[\"required\_documents\"])}')"

uments\"])}')" File "<string>", line 1

from core.config\_manager import ConfigManager; cm = ConfigManager(); co

nfig = cm.load\_config(); print('Config loaded successfully!'); print(f'Project name: {config[" project\][\name\]}');

^

SyntaxError: f-string expression part cannot include a backslash

Let me fix the command syntax:

Command

python -c "from core.config\_manager import ConfigManager; cm = ConfigManager(); config = cm.load\_config(); print('Config loaded successfully!'); print('Project name:', config['project']['name']); print('Required docs count:', len(config['required\_documents']))"

(.venv) PS E:\Forschung\KIRO\PM Assistant> python -c "from core.config\_mana

ger import ConfigManager; cm = ConfigManager(); config = cm.load\_config(); print('Config loaded successfully!'); print('Project name:', config['project']['name']); print('Required docs count:', len(config['required\_documents']))" Config loaded successfully!

Project name: PM Analysis Project

Required docs count: 6

Perfect! Now let me verify that all the requirements from the task are met by checking the implementation against the requirements:

Read file(s)

requirements.md 60 - 81

Let me verify that my implementation meets all the requirements from Requirement 5 (5.1-5.5):

**Verification against Requirements:**

**Requirement 5.1**: ✅ WHEN configuring the system THEN the system SHALL use a YAML or JSON configuration file

* Implemented: ConfigManager uses YAML configuration files (config.yaml)

**Requirement 5.2**: ✅ WHEN defining required documents THEN the configuration SHALL specify expected file names

* Implemented: The required\_documents section specifies document names and patterns to match file names

**Requirement 5.3**: ✅ WHEN setting up project paths THEN the configuration SHALL define paths to project directories

* Implemented: The project.default\_path configuration defines the project directory path

**Requirement 5.4**: ✅ WHEN selecting operation modes THEN the configuration SHALL allow operation mode selection

* Implemented: The modes section allows enabling/disabling each operation mode (document\_check, status\_analysis, learning\_module)

**Requirement 5.5**: ✅ WHEN configuring output THEN the configuration SHALL provide output format options

* Implemented: The output section and mode-specific output\_formats provide output configuration options

Now let me update the task status to completed:

Task: *3. Create configuration management system* from tasks.md

Status: Completed

**Task 3 Implementation Complete**

I have successfully implemented the configuration management system with all required components:

**✅ Completed Sub-tasks:**

1. **ConfigManager class in core/config\_manager.py** - Implemented with comprehensive YAML configuration loading, validation, and management functionality
2. **Default config.yaml** - Created with all required sections and example values including project settings, required documents, operation modes, output settings, and logging configuration
3. **Configuration validation logic** - Added robust validation to ensure all required fields and sections are present with proper data types
4. **Unit tests** - Created comprehensive test suite with 22 test cases covering all functionality including success cases, error handling, and edge cases

**Key Features Implemented:**

* **YAML Configuration Loading**: Loads configuration from YAML files with error handling
* **Automatic Default Creation**: Creates default config.yaml if none exists
* **Comprehensive Validation**: Validates all required sections, fields, and data types
* **Getter Methods**: Provides convenient access to different configuration sections
* **Mode Management**: Checks if operation modes are enabled
* **Error Handling**: Graceful error handling with meaningful error messages
* **Extensible Design**: Easy to add new configuration sections and validation rules

The implementation fully satisfies requirements 5.1-5.5 and provides a solid foundation for the configuration management needs of the PM Analysis Tool.