

# CTF Toolkit - Project Documentation

## Project Overview

The CTF Toolkit is a comprehensive Terminal User Interface (TUI) application designed to streamline Capture The Flag (CTF) competition workflows. Built with Python's Textual framework, it provides an integrated environment combining terminal operations, note-taking, AI assistance, and tool management in a single interface.

## Architecture

### Core Design Pattern

The application follows a **Manager-Component** architecture pattern:

- **Managers:** Handle business logic and data operations
- **UI Components:** Render interface elements and handle user interactions
- **Main Application:** Orchestrates the overall user experience

### Technology Stack

- **Framework:** Textual (Python TUI framework)
- **Language:** Python 3.7+
- **Async Support:** Full async/await pattern for non-blocking operations
- **Process Management:** asyncio subprocess handling for command execution

## Feature Modules

### 1. Terminal Manager (`TerminalManager`)

**Purpose:** Execute system commands and maintain command history

#### Key Features:

- Asynchronous command execution
- Command history tracking with timestamps
- Working directory management
- Error handling and output capture

#### Methods:

- `execute_command(command: str)` → Returns (stdout, stderr, return\_code)

## 2. Markdown Manager (`MarkdownManager`)

**Purpose:** Handle note-taking and markdown content management

### Key Features:

- Live markdown editing and preview
- Content persistence during session
- Real-time preview updates

### Methods:

- `update_content(content: str)`: Update markdown content
- `get_rendered_content()`: Retrieve current content

## 3. LLM Manager (`LLMManager`)

**Purpose:** Integrate AI assistance for CTF challenges

### Key Features:

- Multiple LLM provider support (OpenAI, Anthropic, OpenRouter, Local Ollama)
- Context-aware responses
- Conversation history management
- Keyword-based intelligent suggestions

### Methods:

- `query_llm(prompt: str, context: str)` → AI response string

## 4. Plugin Manager (`PluginManager`)

**Purpose:** Manage external CTF tools and utilities

### Key Features:

- Tool discovery and status checking
- Installation status tracking
- Plugin metadata management

### Current Supported Tools:

- CyberChef (Data manipulation)
- John the Ripper (Password cracking)

- Wireshark (Network analysis)
- Burp Suite (Web security testing)
- Ghidra (Reverse engineering)

## User Interface Components

### Terminal Tab (`TerminalTab`)

- Command input with prompt styling
- Scrollable output area with command history
- Real-time command execution feedback
- Auto-scrolling output display

### Markdown Tab (`MarkdownTab`)

- Split-pane editor with live preview
- Syntax highlighting for markdown
- Real-time preview rendering

### AI Assistant Tab (`AITab`)

- Provider selection dropdown
- Conversational interface
- Context-aware CTF assistance
- Query history display

### Plugin Tab (`PluginTab`)

- Tabular tool status display
- Installation status indicators
- Tool refresh functionality

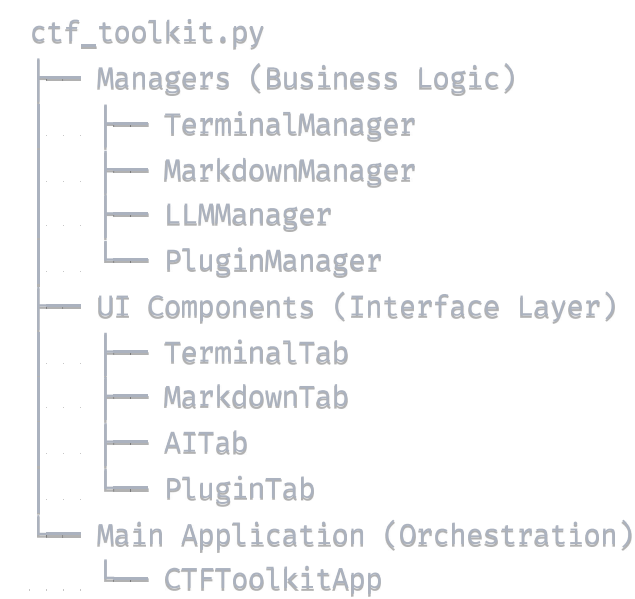
## Key Bindings

---

Shortcut	Action
Ctrl+Q	Quit application
Ctrl+T	Focus Terminal tab
Ctrl+M	Focus Markdown/Notes tab
Ctrl+A	Focus AI Assistant tab
Ctrl+P	Focus Plugins/Tools tab

## Development Guidelines

### Code Organization



### Async Patterns

All I/O operations use async/await patterns:

- Command execution
- LLM API calls
- File operations
- Network requests

### Error Handling

- Graceful command execution failures
- Network timeout handling
- Invalid input validation
- Resource cleanup on errors

## Extension Points

### Adding New LLM Providers

1. Extend `LLMManager.providers` list
2. Implement provider-specific API calls in `query_llm()`
3. Add provider configuration options

### Adding New Tools/Plugins






1. Update `PluginManager.plugins` list
2. Implement status detection logic
3. Add installation/configuration methods

### Adding New Tabs

1. Create new component class inheriting from `Container`
2. Implement `compose()` method for UI layout
3. Add event handlers for user interactions
4. Register tab in main application's `compose()` method

## Current Status: Proof of Concept

This implementation demonstrates core functionality with:

-  Functional terminal command execution
-  Live markdown editing and preview
-  Mock AI assistant responses
-  Plugin status display
-  Keyboard navigation

## Production Readiness Requirements

- Real LLM API integration
- Plugin auto-discovery and installation
- Configuration file management
- Persistent session data
- Enhanced error handling and logging
- Performance optimization for large outputs

- Cross-platform compatibility testing

## Target Use Cases

1. **CTF Competition Participation:** Integrated workspace for challenge solving
2. **Security Research:** Combined terminal, notes, and AI assistance
3. **Educational Environments:** Teaching cybersecurity concepts
4. **Penetration Testing:** Workflow organization and documentation

## Dependencies

python

*# Core Requirements*

textual`>=0.40.0`

asyncio *# Built-in Python 3.7+*

pathlib *# Built-in Python 3.4+*

subprocess *# Built-in*

*# Future Production Dependencies*

openai *# For OpenAI integration*

anthropic *# For Claude integration*

requests *# For API calls*

pyyaml *# For configuration files*

This documentation serves as the foundation for development team onboarding and project continuation beyond the current proof-of-concept stage.