**WhisperJAV Project - Complete Development Guide for version 1.01**

**For Future Code Assistants: This document contains everything needed to understand and continue development of the WhisperJAV project.  
  
PROJECT STRUCTURE:  
  
  
(WJ) PS C:\bin\git\WhisperJav\_V1\_Minami\_Edition> tree /f .**

**Folder PATH listing**

**Volume serial number is 3A73-32BD**

**C:\BIN\GIT\WHISPERJAV\_V1\_MINAMI\_EDITION**

**│ .gitignore**

**│ create\_project\_structure.py**

**│ README.md**

**│ requirements.txt**

**│ setup.py**

**│**

**├───docs**

**├───examples**

**│ config\_example.json**

**│**

**├───tests**

**│ test\_balanced\_pipeline.py**

**│ test\_basic.py**

**│**

**├───whisperjav**

**│ │ cli.py**

**│ │ main.py**

**│ │ \_\_init\_\_.py**

**│ │**

**│ ├───config**

**│ │ \_\_init\_\_.py**

**│ │**

**│ ├───modules**

**│ │ audio\_extraction.py**

**│ │ audio\_preparation.py**

**│ │ audio\_preprocessing.py**

**│ │ media\_discovery.py**

**│ │ scene\_detection.py**

**│ │ segment\_classification.py**

**│ │ srt\_postprocessing.py**

**│ │ srt\_postproduction.py**

**│ │ srt\_stitching.py**

**│ │ stable\_ts\_asr.py**

**│ │ translation.py**

**│ │ whisper\_pro\_asr.py**

**│ │ \_\_init\_\_.py**

**│ │**

**│ ├───pipelines**

**│ │ balanced\_pipeline.py**

**│ │ base\_pipeline.py**

**│ │ faster\_pipeline.py**

**│ │ fast\_pipeline.py**

**│ │ \_\_init\_\_.py**

**│ │**

**│ └───utils**

**│ logger.py**

**│ metadata\_manager.py**

**│ user\_interface.py**

**│ \_\_init\_\_.py**

**│**

**└───whisperjav.egg-info**

**dependency\_links.txt**

**entry\_points.txt**

**PKG-INFO**

**requires.txt**

**SOURCES.txt**

**top\_level.txt**

**🎯 PROJECT OVERVIEW**

WhisperJAV is a **production-ready, fully functional** Japanese subtitle generation system. The core system works perfectly - the development task is **enhancing and extending** existing functionality, not fixing broken code.

**Project Status:**

* ✅ **Core system fully functional** - all 3 pipelines work without errors
* ✅ **No import failures or execution errors**
* ✅ **JSON schema working as designed**
* 🔧 **Optional enhancement modules** need implementation
* 🎨 **User interface** needs enhancement for optional features

**🏗️ SYSTEM ARCHITECTURE (Working)**

**Three Processing Pipelines:**

1. **Faster Pipeline**: Direct transcription (no scene detection)
2. **Fast Pipeline**: Scene detection + standard Whisper
3. **Balanced Pipeline**: Scene detection + VAD-enhanced WhisperPro

**Critical File Naming Convention:**

* **Scenes**: {media\_basename}\_scene\_{scene\_idx:04d}.wav
* **SRTs**: {media\_basename}\_scene\_{scene\_idx:04d}.srt
* **Final Output**: {basename}.ja.whisperjav.srt

**⚠️ This naming convention is hardcoded throughout the system and must be maintained.**

**📊 JSON METADATA SCHEMA**

The schema was designed to be **forward-thinking with placeholders** for optional modules:

json

{

"metadata\_master": {...},

"input\_info": {...},

"config": {

"mode": "faster|fast|balanced",

"enhancement\_options": {

"adaptive\_classification": false,

"adaptive\_audio\_enhancement": false,

"smart\_postprocessing": false

}

},

"scenes\_detected": [...],

"processing\_stages": {

"scene\_classification": {"status": "skipped", "enabled": false},

"audio\_preprocessing": {"status": "skipped", "enabled": false}

},

"quality\_metrics": {

"classification\_confidence": null,

"audio\_enhancement\_applied": false

}

}

**Important:** Optional modules should use **existing schema placeholders**, not add new attributes.

**🎯 CURRENT DEVELOPMENT TASK**

**Phase 1: User Interface Enhancement (Current Priority)**

Design user-centric interface for optional modules:

**Technical Module → User-Facing Name:**

* segment\_classification.py → **"Adaptive Classification"**
* audio\_preprocessing.py → **"Adaptive Audio Enhancement"**
* srt\_postproduction.py → **"Smart Post-Processing"**

**Phase 2: Optional Module Implementation**

Implement actual functionality for stub modules.

**⚙️ CONFIGURATION PHILOSOPHY**

**Current Approach:** No centralized configuration files

* Module defaults in \_\_init\_\_ methods
* Parameter passing through constructors
* Keep simple and straightforward
* Avoid configuration complexity

**🔧 DEVELOPMENT PRINCIPLES**

**Core Rules:**

1. **Don't fix what isn't broken** - system works perfectly
2. **Enhance, don't replace** existing functionality
3. **Maintain backward compatibility** at all costs
4. **Follow established naming conventions** exactly
5. **Use single master JSON** for all metadata
6. **Scene/segment boundaries must remain identical**

**Optional Module Integration:**

* **Read from:** Existing JSON metadata and scene files
* **Write to:** Existing JSON schema placeholders
* **Default state:** All optional modules OFF
* **User control:** CLI flags with user-friendly names

**🚀 IMMEDIATE NEXT STEPS**

**Step 1: CLI Interface Enhancement**

bash

*# Extend main.py with user-friendly flags*

python main.py input --mode balanced \

--adaptive-classification \

--adaptive-audio-enhancement \

--smart-postprocessing

**Step 2: Pipeline Contract Updates**

python

*# Add optional parameters to pipeline constructors*

class BalancedPipeline:

def \_\_init\_\_(self,

adaptive\_classification: bool = False,

adaptive\_audio\_enhancement: bool = False):

**Step 3: Module Implementation**

Implement actual functionality in stub modules, using existing JSON schema fields.

**📚 KEY FILES FOR UNDERSTANDING**

**Entry Points:**

* main.py - CLI interface and file processing coordination
* whisperjav/\_\_init\_\_.py - Public API exports

**Core Infrastructure:**

* utils/logger.py - Logging implementation (provided, working)
* utils/metadata\_manager.py - JSON schema management
* modules/media\_discovery.py - File discovery with ffprobe

**Processing Modules (All Working):**

* modules/audio\_extraction.py - FFmpeg audio extraction
* modules/scene\_detection.py - Two-pass auditok scene detection
* modules/stable\_ts\_asr.py - Stable-whisper with Japanese processing
* modules/whisper\_pro\_asr.py - Standard whisper with VAD
* modules/srt\_stitching.py - Scene SRT combination
* modules/srt\_postprocessing.py - Hallucination removal

**Pipeline Implementations (All Working):**

* pipelines/base\_pipeline.py - Abstract base
* pipelines/faster\_pipeline.py - Direct transcription
* pipelines/fast\_pipeline.py - Scene-based standard
* pipelines/balanced\_pipeline.py - Scene-based with VAD

**Optional Modules (Stubs, Need Implementation):**

* modules/segment\_classification.py - Scene classification
* modules/audio\_preprocessing.py - Audio enhancement
* modules/srt\_postproduction.py - Final SRT refinements
* modules/audio\_preparation.py - Audio optimization
* modules/user\_interface.py - Progress reporting

**🎯 DEVELOPMENT STRATEGY**

**User-Centric Design Priority:**

1. **Start with user interface** - how should optional features be presented?
2. **Design user-friendly naming** - "adaptive classification" vs "segment\_classification"
3. **Create intuitive CLI flags** - easy to understand and use
4. **Propagate UI design to module contracts** - maintain consistency

**Implementation Priority:**

1. CLI interface enhancement for optional modules
2. Pipeline contract updates for optional parameters
3. segment\_classification.py (adaptive classification)
4. audio\_preprocessing.py (adaptive audio enhancement)
5. srt\_postproduction.py (smart post-processing)

**⚠️ CRITICAL WARNINGS**

**What NOT to Change:**

* Scene naming convention: {basename}\_scene\_{idx:04d}.wav
* Working import paths (no import failures currently)
* Existing JSON schema structure
* Working pipeline processing flows
* Any functionality that currently works

**What TO Change:**

* Add user-friendly CLI flags for optional modules
* Implement stub optional modules
* Enhance existing JSON schema placeholders
* Improve user experience with optional features

**🔍 DEBUGGING APPROACH**

If any issues arise:

1. **Verify current system still works** before making changes
2. **Test with user's existing media files** and testing approach
3. **Maintain all existing functionality** while adding new features
4. **Ask questions** before making assumptions about what needs fixing

**📝 COMMUNICATION NOTES**

* User has limited tokens - be efficient and focused
* User tests thoroughly with own media files
* User prefers simple, straightforward approaches
* User values backward compatibility and working systems
* Always ask for clarification before major changes

**SUMMARY: This is a working, production-ready system that needs enhancement, not repair. Focus on user-centric design for optional features and careful implementation that maintains all existing functionality.**

RECOMMENDATIONS FOR THE NEXT STEPS BASED ON A USER CENTRIC APPROACH for VERSION 1.02  
  
  
  
**User-Centric Interface Design**

**🔍 Reviewing Forward-Thinking Schema Design**

Looking at the existing JSON schema, I can see the placeholders you mentioned:

json

{

"processing\_stages": {

"scene\_classification": {"status": "skipped", "enabled": false},

"audio\_preprocessing": {"status": "skipped", "enabled": false}

},

"quality\_metrics": {

"classification\_confidence": null,

"audio\_enhancement\_applied": false

}

}

**🎯 User-Centric Module Naming Strategy**

| **Technical Module** | **User-Facing Name** | **User Benefit Description** |
| --- | --- | --- |
| segment\_classification.py | **"Adaptive Classification"** | Automatically detects scene types (dialogue, music, background) to optimize ASR settings per scene |
| audio\_preprocessing.py | **"Adaptive Audio Enhancement"** | Intelligently cleans and enhances audio quality before transcription |
| srt\_postproduction.py | **"Smart Post-Processing"** | Advanced subtitle refinement and quality improvement |
| audio\_preparation.py | **"Audio Optimization"** | Prepares audio for optimal transcription quality |

**🖥️ User Interface Design Proposal**

**Current CLI (main.py) Extension:**

bash

*# Current working interface*

python main.py input\_path --mode balanced --output\_dir ./output

*# Proposed enhanced interface*

python main.py input\_path --mode balanced --output\_dir ./output \

--adaptive-classification \

--adaptive-audio-enhancement \

--smart-postprocessing

**User-Friendly Help Text:**

Optional Enhancement Features:

--adaptive-classification Automatically optimize ASR settings per scene type

--adaptive-audio-enhancement Intelligently enhance audio before transcription

--smart-postprocessing Advanced subtitle quality improvement

--audio-optimization Prepare audio for optimal transcription

**🔗 Contract Design Between UI and Modules**

**Option 1: Pipeline Constructor Flags (Recommended)**

python

*# In pipeline constructors*

class BalancedPipeline:

def \_\_init\_\_(self,

adaptive\_classification: bool = False,

adaptive\_audio\_enhancement: bool = False,

smart\_postprocessing: bool = False,

audio\_optimization: bool = False):

**Option 2: Config Dictionary Approach**

python

*# Passed to pipeline*

enhancement\_config = {

"adaptive\_classification": {"enabled": True, "params": {}},

"adaptive\_audio\_enhancement": {"enabled": True, "params": {}},

"smart\_postprocessing": {"enabled": False, "params": {}}

}

**Option 3: Enhancement Profile System**

python

*# Predefined enhancement profiles*

--enhancement-profile basic *# No optional modules*

--enhancement-profile standard *# Classification + audio enhancement*

--enhancement-profile premium *# All modules enabled*

--enhancement-profile custom *# Individual flags*

**🚀 Recommended Implementation Flow**

1. **Extend main.py argument parser** with user-friendly optional flags
2. **Pass flags to pipeline constructors** as boolean parameters
3. **Pipeline orchestrates module calls** based on flags
4. **Modules update existing JSON schema fields** (not add new ones)
5. **Maintain backward compatibility** - all flags default to False