Mock Generation System - Project Workflow Documentation

*Generated on August 18, 2025 at 09:02 PM*

# Table of Contents

* 1. Executive Summary
* 2. Project Overview
* 3. System Architecture
* 4. Installation & Setup
* 5. User Workflow
* 6. Configuration Guide
* 7. Usage Examples
* 8. Output Examples
* 9. Troubleshooting
* 10. Advanced Features
* 11. Project Benefits
* 12. Conclusion

# 1. Executive Summary

The Mock Generation System is a powerful, enterprise-grade solution designed to generate high-quality mock JSON data for software testing, development, and data validation workflows. This system addresses the critical need for realistic, consistent, and configurable test data across various development scenarios.

## Key Benefits

* 🚀 Rapid mock data generation with probability-based scenarios
* 🔄 Master template integration for consistent data structure
* 📊 Multiple output formats (single, multiple, split files)
* 🛡️ Comprehensive error handling and validation
* ⚡ CLI interface for automation and CI/CD integration
* 📁 Organized output management with timestamps

# 2. Project Overview

The Mock Generation System was developed to solve common challenges in software development and testing:

* Need for realistic test data that matches production scenarios
* Time-consuming manual data creation for different test cases
* Inconsistent data structures across different testing phases
* Lack of probability-based scenarios for comprehensive testing
* Difficulty in maintaining data consistency across team members

## Our Solution

A comprehensive system that provides:

* Automated mock data generation with configurable templates
* Probability-based scenarios (positive, negative, exclusion)
* Master template integration for consistent data structure
* Flexible output formats and file management
* Easy-to-use CLI interface for developers and testers

# 3. System Architecture

The system follows a modular, layered architecture that ensures maintainability, extensibility, and robust operation.

## System Architecture Diagram

|  |  |  |
| --- | --- | --- |
| Layer | Components | Responsibilities |
| User Interface | • CLI Interface • Configuration Files • Help System | • User interaction • Command processing • Input validation |
| Core Processing | • Data Generator • Probability Engine • Template Manager | • Data generation • Scenario processing • Template integration |
| Data Management | • Input Parser • Output Formatter • File Manager | • Configuration loading • Output formatting • File operations |

## Data Flow Process

1. 1. User provides configuration (user\_input.json)
2. 2. System loads master template (master.json)
3. 3. Configuration is validated and parsed
4. 4. Data generation engine processes models
5. 5. Probability scenarios are applied
6. 6. Output is formatted according to specifications
7. 7. Files are saved with timestamps and organization

# 4. Installation & Setup

## Prerequisites

Before installing the Mock Generation System, ensure you have:

* Python 3.6 or higher installed
* Git or download access to the project files
* Basic understanding of JSON configuration
* Command line interface access

## Installation Steps

1. Clone or download the project files to your local machine
2. Navigate to the project directory: cd Mockup\_up\_data-main
3. Install required dependencies: pip install python-docx
4. Verify installation by running: python -m src.mockgen.cli --help
5. Ensure user\_input.json and master.json files are present

## Quick Verification

Test that everything is working correctly:

python -m src.mockgen.cli --enhanced - This should generate output files in the generated\_outputs/ directory

# 5. User Workflow

The typical user workflow involves three main phases: Configuration, Generation, and Output Management.

## Phase 1: Configuration

1. Review and understand your data requirements
2. Modify user\_input.json with your specific data models
3. Customize master.json template if needed
4. Define probability scenarios (positive, negative, exclusion)
5. Test configuration with small data sets

## Phase 2: Generation

1. Choose the appropriate generation method
2. Run CLI commands with desired parameters
3. Monitor generation progress and output
4. Verify generated data meets requirements
5. Generate additional scenarios if needed

## Phase 3: Output Management

1. Review generated files in output directory
2. Organize outputs by type and timestamp
3. Integrate data into your testing workflow
4. Archive or clean up old output files
5. Document any custom configurations for team use

# 6. Configuration Guide

The system uses two main configuration files: user\_input.json for your data models and master.json for the base template structure.

## user\_input.json Configuration

This file defines your data models and probability scenarios:

|  |  |
| --- | --- |
| Model Type | Purpose |
| Model\_1 | Base model with standard data |
| Model\_1\_Positive | Valid data for positive testing |
| Model\_1\_Negative | Invalid data for error testing |
| Model\_1\_Exclusion | Data that should be filtered out |

## master.json Configuration

The master template provides the base structure for all generated outputs:

* Defines the overall data structure
* Provides default values for common fields
* Ensures consistency across all models
* Can be customized for your specific needs

# 7. Usage Examples

## Basic Usage Examples

### Example 1: Generate Enhanced Output for All Models

python -m src.mockgen.cli --enhanced - This generates output for all configured models using the master template

### Example 2: Generate Output for Specific Model

python -m src.mockgen.cli --enhanced --model Model\_1 - This generates output only for Model\_1

### Example 3: Generate Multiple Records

python -m src.mockgen.cli --model Model\_1 --count 10 - This generates 10 records for Model\_1

## Advanced Usage Examples

### Probability Generator Usage

* Generate all probability types: python generate\_probability\_outputs.py --all
* Generate only positive scenarios: python generate\_probability\_outputs.py --positive
* Generate with multiple records: python generate\_probability\_outputs.py --all --count 5
* Generate split files: python generate\_probability\_outputs.py --all --count 3 --split

# 8. Output Examples

## Output Structure Overview

Generated outputs follow a consistent structure based on your configuration and the master template.

## Enhanced Output Example

When using --enhanced mode, output includes the complete master template structure:

|  |  |
| --- | --- |
| Field | Value |
| first\_name | John |
| last\_name | Smith |
| proc\_cd | Vishnu |
| city | Hyderabad |
| country | United States |

## File Naming Convention

Generated files follow a consistent naming pattern:

* enhanced\_output\_20241201\_143022\_123456Z.json
* Model\_1\_Positive\_20241201\_143022\_123456Z.json
* Model\_1\_Negative\_20241201\_143022\_123457Z.json
* Model\_1\_Exclusion\_20241201\_143022\_123458Z.json

# 9. Troubleshooting

## Common Issues and Solutions

### Issue 1: "Module not found" Error

**Problem:** Python cannot find the required modules

**Solution:** Ensure you are in the project root directory (Mockup\_up\_data-main) when running commands

### Issue 2: "File not found" Error

**Problem:** System cannot find configuration files

**Solution:** Verify that user\_input.json and master.json exist in the current directory

### Issue 3: No Output Generated

**Problem:** Commands run successfully but no output files are created

**Solution:** Check that the generated\_outputs/ folder exists and is writable

## Debug Commands

* python -m src.mockgen.cli --help - Shows all available options
* python generate\_probability\_outputs.py --list - Lists available models
* python -m src.mockgen.cli --init - Creates template configuration files

# 10. Advanced Features

## Split File Generation

Generate separate files for each record or model:

python -m src.mockgen.cli --enhanced --output-format split --count 5 - Creates 5 separate files, one for each record

## Master Template Integration

The system automatically merges your configuration with the master template:

* Consistent data structure across all outputs
* Automatic field population from template
* Flexible override of template values
* Professional, standardized output format

## Probability Scenarios

Generate different types of test scenarios:

* Positive: Valid data for normal testing
* Negative: Invalid data for error handling testing
* Exclusion: Data that should be filtered out

# 11. Project Benefits

## Benefits for Developers

* 🚀 Time savings through automated data generation
* 🔄 Consistent test data across development phases
* 🛡️ Better testing coverage with probability scenarios
* ⚡ Faster development and testing cycles
* 📊 Realistic data that matches production scenarios

## Benefits for Teams

* 👥 Standardized data generation across team members
* 🤝 Easy sharing of data configurations
* 📋 Consistent testing standards and procedures
* 🔄 Simplified onboarding for new team members
* 📊 Unified approach to test data management

## Benefits for Organizations

* 💰 Cost reduction through faster development cycles
* 🎯 Improved software quality and reliability
* 🛡️ Reduced risk through comprehensive testing
* 📈 Scalable testing capabilities as projects grow
* 🏆 Professional, enterprise-grade testing tools

# 12. Conclusion

The Mock Generation System provides a comprehensive solution for generating high-quality mock data for software development and testing. With its advanced features, flexible configuration, and professional output formats, it significantly improves the efficiency and quality of development workflows.

## Key Takeaways

* The system is designed to be easy to use while providing powerful capabilities
* Master template integration ensures consistent, professional outputs
* Probability-based scenarios enable comprehensive testing coverage
* Flexible output formats support various development and testing needs
* The CLI interface enables automation and integration with existing workflows

## Getting Started

To begin using the Mock Generation System:

1. 1. Review the configuration files and understand the structure
2. 2. Start with simple models and gradually expand complexity
3. 3. Experiment with different output formats and scenarios
4. 4. Integrate the system into your existing development workflow
5. 5. Customize configurations to match your specific needs

The system is designed to grow with your needs. Start simple and expand as you become more familiar with its capabilities. For additional support, refer to the built-in help system and comprehensive documentation.

# Appendix A: Command Reference

## Enhanced System CLI Commands

|  |  |
| --- | --- |
| Command Option | Description |
| --enhanced | Use enhanced mode with master template integration |
| --model MODEL | Generate output for specific model |
| --models MODEL1 MODEL2 | Generate output for multiple specific models |
| --count N | Generate N number of records |
| --output-format FORMAT | Set output format (single/multiple/split) |
| --split | Write each output to separate file |
| --config FILE | Path to custom configuration file |
| --master FILE | Path to custom master template file |
| --init | Create template configuration files |
| --legacy | Force legacy mode for backward compatibility |

## Probability Generator Commands

|  |  |
| --- | --- |
| Command Option | Description |
| --all | Generate all probability types (positive, negative, exclusion) |
| --positive | Generate only positive probability outputs |
| --negative | Generate only negative probability outputs |
| --exclusion | Generate only exclusion probability outputs |
| --model MODEL | Generate outputs for specific model only |
| --count COUNT | Number of records to generate per probability type |
| --split | Generate separate files for each record |
| --config FILE | Path to custom configuration file |
| --output-dir DIR | Custom output directory for generated files |
| --list | List available models and their probability types |