AutoProjectManagement Configuration Guide

Complete Setup and Configuration Instructions

Configuration Overview

The **AutoProjectManagement Configuration Guide** provides comprehensive instructions for setting up and customizing all aspects of your project management environment.

- GitHub Integration
- Security Configuration
- Logging Configuration
- JSON Storage
- Project Configuration
- Environment Settings
- Validation
- Management

Purpose of Proper Configuration

- Ensure system security and data protection
- Optimize performance for your specific environment
- Enable seamless integration with external services
- Provide reliable logging for troubleshooting
- Support different deployment environments

GitHub Integration

Environment Variables

GITHUB_TOKEN	`ghp_xxxxxxxxxxxxxxxxxxxx`
GITHUB_USERNAME	`yourusername`
GITHUB_DEFAULT_REPO	`username/repository`

Key Settings

auto_sync	Enable automatic synchronization with GitHub
sync_interval	Time in seconds between sync operations
default_branch	Default branch for repository operations

Best Practices

- ✓ Use fine-grained tokens with minimal required permissions
- ✓ Store tokens securely, never commit to version control
- ✓ Set reasonable sync intervals to avoid API rate limits

Configuration Example

```
{ "github": { "token": "ghp_xxxxxxxxxxxxxxxxxxxx",
"username": "yourusername", "default_repo":
"username/repository", "default_branch": "main",
"auto_sync": true, "sync_interval": 300 } }
```

Security Configuration

Security Architecture

Authentication
Verify user identity

Authorization
Control access permissions

Encryption
Protect data in transit

Rate Limiting
Prevent abuse

Security Components

JWT Tokens
JSON Web Tokens for secure authentication

API Keys
Unique identifiers for API access control

SSL/TLS
Encrypted communication channels

CORS Policies
Cross-origin resource sharing controls

Configuration Options

Setting	Type	Description
jwt_secret_key	`string`	JWT signing secret key
jwt_algorithm	`string`	JWT algorithm (HS256/RS256)
api_key_required	`boolean`	Require API key
cors_allowed_origins	`list`	CORS allowed origins

Configuration Examples

```
{ "security": { "jwt_secret_key": "your-super-secret-
key-change-this", "jwt_algorithm": "HS256",
"jwt_expiration_minutes": 60, "api_key_required":
true, "api_key_header": "X-API-Key",
"cors_allowed_origins": ["*"] } }
```

```
{ "security": { "ssl_cert_path": "/certs/server.crt",
"ssl_key_path": "/certs/server.key" } }
```

Best Practices

Strong JWT Secrets
Use long, randomly generated secrets for JWT signing

Token Expiration
Set reasonable expiration times for access tokens

Logging Configuration

Logging Architecture

Application
Source of all log events

Logging Config
Controls log behavior & format

Log Handlers
Console, File, Rotating File

Log Format
Standard or JSON format

Configuration Options

Setting	Type	Description
level	`string`	Logging level (DEBUG, INFO, WARNING, ERROR)
format	`string`	Log format string
file_path	`string`	Log file path
max_file_size	`string`	Maximum log file size
json_format	`boolean`	Use JSON format

Best Practices

- ✓ Use appropriate log levels
- ✓ Rotate log files regularly
- ✓ Use JSON in production
- ✓ Include timestamps

Configuration Examples

Basic Console Logging

Simple configuration for development environment

```
{ "logging": { "level": "INFO", "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s" } }
```

File Logging with Rotation

Production-ready file logging with rotation

```
{ "logging": { "level": "DEBUG", "file_path": "logs/autoproject.log", "max_file_size": "10MB", "backup_count": 5 } }
```

JSON Structured Logging

Machine-readable logs for monitoring systems

```
{ "logging": { "level": "INFO", "json_format": true, "include_extra": true } }
```

JSON Storage Configuration

Storage Architecture

Application
Interacts with JSON storage layer

JSON Storage
Manages all JSON data operations

JSonDataBase Directory
Contains Inputs, Outputs, Backups

Configuration Options

Setting	Type	Description
type	`string`	Storage type (always 'json')
json_path	`string`	Main JSON configuration file
data_directory	`string`	Directory for JSON data files
backup_enabled	`boolean`	Enable automatic backups
max_file_size	`integer`	Maximum JSON file size

Best Practices

- ✓ Enable backups for data safety
- ✓ Set reasonable file size limits
- ✓ Organize files logically
- ✓ Use UTF-8 encoding

Configuration Examples

Basic JSON Storage

Simple configuration with default paths

```
{ "database": { "type": "json", "json_path":
"autoproject.json", "data_directory":
"JSonDataBase", "backup_enabled": true } }
```

Custom Directory Structure

Custom paths for organized storage

```
{ "database": { "type": "json", "json_path":
"config/project_data.json", "data_directory":
"project_data", "inputs_path":
"project_data/inputs", "outputs_path":
"project_data/outputs", "backup_count": 10 } }
```

Production JSON Storage

Optimized for production environment

```
{ "database": { "type": "json", "json_path":
"/opt/autoproject/data/main.json",
"data_directory": "/opt/autoproject/data",
"backup_enabled": true, "backup_count": 20,
"max_file_size": 52428800 } }
```

Project Configuration

Configuration Structure

ProjectConfig
Core configuration settings and parameters

ProjectManager
Handles directory setup and validation

FileManager
Manages file operations and backups

Configuration Example

```
{ "project": { "base_path": "/path/to/project",
"data_path": "JSonDataBase", "output_path":
"JSonDataBase/OutPuts", "backup_path":
"project_management/PM_Backups", "max_file_size":
10485760, "allowed_extensions": [ ".py", ".js",
".json", ".md" ], "max_workers": 8, "timeout_seconds":
300, "enable_auto_commit": true,
"enable_risk_analysis": true,
"enable_progress_tracking": true,
"enable_github_integration": true } }
```

Key Settings & Options

base_path
Project base directory

data_path
Data directory path

max_file_size
Maximum file size (10MB)

allowed_extensions
Permitted file types

max_workers
Maximum worker threads

timeout_seconds
Operation timeout

enable_auto_commit
Automatic commits

enable_risk_analysis
Risk analysis features

Best Practices

- ✓ Set reasonable file size limits
- ✓ Restrict allowed file extensions
- ✓ Enable auto-commit for safety
- ✓ Adjust workers based on system resources

Feature Toggles

Auto Commit
Automatically save changes to repository

Risk Analysis
Evaluate potential project risks

Progress Tracking
Monitor project advancement

GitHub Integration
Connect with GitHub repositories

Environment-Specific Configurations

Configuration Matrix

Feature	Development	Production	Testing
API Debug	true	false	true
Storage Type	JSON files	JSON files	JSON files
Logging Level	DEBUG	INFO	DEBUG
GitHub Integration	true	true	false
SSL Required	false	true	false
Rate Limiting	relaxed	strict	disabled

Environment Detection

Check ENVIRONMENT
Read environment variable

Match Value
Identify environment type

Load Settings
Apply appropriate config

Key Differences

- Debug mode in dev/test
- SSL required in production
- Verbose logging in dev/test
- Strict rate limiting in prod

Environment Examples

Development

Debug: true | Logging: DEBUG | GitHub: enabled

```
{ "api": { "host": "127.0.0.1", "port": 8000, "debug":
true, "reload": true }, "logging": { "level": "DEBUG" } }
```

Production

SSL: enabled | Logging: INFO | Rate limiting: strict

```
{ "api": { "host": "0.0.0.0", "port": 8080, "debug": false
}, "security": { "ssl_cert_path": "/certs/server.crt" },
"logging": { "level": "INFO" } }
```

Testing

Debug: true | Storage: in-memory | GitHub: disabled

```
{ "api": { "host": "127.0.0.1", "port": 8001, "debug": true
}, "database": { "json_path": ":memory:" }, "project": {
"enable_github_integration": false } }
```

Configuration Validation & Management

Validation Process

1. **Load Configuration**
   Read and parse configuration files
2. **Validate Required Fields**
   Check for presence of mandatory settings
3. **Validate Data Types**
   Ensure correct data types for all values
4. **Validate Security Settings**
   Check security parameters for risks
5. **Validate Dependencies**
   Verify external service connections

Hot Reload Process

Update Config
Administrator modifies configuration file

Detect Change
File system change notification

Validate
Configuration validation process

Apply
Update running application

Management Commands

```
# Python API config.validate() # Validate current configuration print_config()
# Display current configuration reload_config() # Reload configuration from
files # CLI Arguments --config config.json # Specify configuration file --
validate # Validate configuration --generate # Generate sample configuration --
env development # Specify environment
```

Backup & Restore

Backup Commands

```
cp config.json config.backup.json
```

```
cp config.json config.backup.$(date
+%Y%m%d_%H%M%S).json
```

```
python -m
autoprojectmanagement.configuration
backup
```

Restore Commands

```
cp config.backup.json config.json
```

```
python -m
autoprojectmanagement.configuration
validate
```

```
python -m
autoprojectmanagement.configuration
restore
```

Management Tools

Python API
Programmatic configuration management

CLI Tools
Command-line interface utilities

Hot Reload
Live configuration updates

Backup System
Automatic configuration backups

Validation Rules

- ✓ Valid JSON syntax
- ✓ Required fields present
- ✓ Correct data types
- ✓ Valid file paths
- ✓ Secure settings
- ✓ Available ports

Troubleshooting & Quick Reference

Common Issues & Solutions

Invalid JSON
Use JSON validator to check syntax

Missing Required Field
Add missing configuration parameter

Invalid Port
Change to available port (1-65535)

JSON File Not Found
Check file paths and permissions

GitHub Token Invalid
Regenerate GitHub token with proper permissions

SSL Certificate Error
Verify certificate paths and validity

Configuration Checklist

- JSON syntax is valid
- All required fields are present
- JSON file paths are correct
- GitHub token has correct permissions
- Security settings are properly configured

Getting Help

Documentation
- Configuration Guide
- API Reference
- Installation Guide

Community Support
- GitHub Issues
- Discussions
- Wiki Pages

Professional Support
- support@autoprojectmanagement.com
- docs@autoprojectmanagement.com
- consulting@autoprojectmanagement.com

Diagnostic Commands

Quick Diagnostics

```
python -m json.tool config.json
```

```
python -c "from autoproject_configuration import config;
config.validate()"
```

```
python -c "from autoproject_configuration import config;
print(f'Storage: {config.database.type}')"
```

```
python -c "from
autoprojectmanagement.services.github_integration import
GitHubIntegration; print('GitHub ready')"
```