

# Development Environment Guide

## AutoProjectManagement System

Comprehensive guidance for setting up and maintaining the development environment, covering system requirements, tools, configuration, testing, and best practices.



Setup



Development



Tools



Testing

# System Requirements




## Hardware Requirements

Component	Minimum	Recommended
CPU	Dual-core 2.0 GHz	Quad-core 3.0 GHz
RAM	4 GB	8 GB
Storage	10 GB free space	50 GB SSD
Network	Broadband	High-speed

## Software Requirements

Software	Version	Purpose
Python	3.8+	Core runtime
Node.js	14.x+	Frontend tooling
Git	2.20+	Version control
Docker	20.10+	Containerization
VS Code	1.60+	IDE

## Operating System Support

-  **Linux:** Ubuntu 20.04+, CentOS 8+, Debian 10+
-  **macOS:** 10.15+ (Catalina or later)
-  **Windows:** Windows 10 (Build 19041+) with WSL2

# Environment Setup

## Prerequisites Installation

### 1 Python Environment

Install Python 3.8+ and pip

```
python3 --version
```

### 2 Node.js Installation

Use Node Version Manager (recommended)

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
```

### 3 Verify Installation

Check versions of installed tools

```
node --version && npm --version
```

## Project Setup Process

### 1 Clone Repository

Get project source code

```
git clone https://github.com/autoprojectmanagement/autoprojectmanagement
```

### 2 Create Virtual Environment

Isolate project dependencies

```
python3 -m venv venv && source venv/bin/activate
```

### 3 Install Dependencies

Install required packages

```
pip install -r requirements.txt -r requirements-dev.txt
```

### 4 Setup Auto Environment

Run automated setup

```
python -m autoprojectmanagement.setup
```

# Development Tools

## <> IDE Configuration



### Python

Language support and debugging

```
ext install ms-python.python
```



### Pylance

Type checking and IntelliSense

```
ext install ms-python.vscode-pylance
```



### Black Formatter

Code formatting and style

```
ext install ms-python.black-formatter
```



### GitLens

Git integration and insights

```
ext install eamodio.gitlens
```



### Docker

Container development support

```
ext install ms-azuretools.vscode-docker
```

## ⚙️ VS Code Settings

```
{"python.defaultInterpreterPath": "./venv/bin/python", "python.formatting.provider": "black", "editor.formatOnSave": true}
```

# Project Structure

## Directory Architecture

### AutoProjectManagement/

Root directory containing all project files

autoprojectmanagement/

tests/

Docs/

JSonDataBase/

venv/

.vscode/

### autoprojectmanagement/

Main package containing core application code

main\_modules/

services/

api/

templates/

### tests/

Test suite organized by category

unit/

integration/

conftest.py

## Module Structure

### main\_modules

Core business logic

- ProjectManagementSystem
  - TaskManager
  - ConfigurationHandler

### services

External integrations

- GitHubService
- StatusService
- NotificationService

### api

REST API endpoints

- FastAPI application
- Routers
- Request models

### templates

Code generation

- HeaderUpdater
- DocumentationGenerator
- TemplateEngine

## Service Dependencies

ProjectManagementSystem

→ GitHubService, StatusService,  
ConfigurationService

API Server

→ FastAPI, Uvicorn,  
ProjectManagementSystem

# Configuration Management

## ⚙️ Environment Variables

Variable	Description	Default	Required
PYTHONPATH	Module search path	.	Yes
ENV	Environment mode	development	No
LOG_LEVEL	Logging level	INFO	No
GITHUB_TOKEN	GitHub API token	-	Yes*
DATABASE_URL	Database connection	sqlite:///app.db	No

\* Required for GitHub integration features

## 📄 Configuration Files

### ⚙️ pyproject.toml

Project configuration and build settings

```
[build-system]
requires = ["setuptools>=61.0", "wheel"]
build-backend = "setuptools.build_meta"
[project]
name = "autoprojectmanagement"
version = "1.0.0"
requires-python = ">=3.8"
```

### 📄 pytest.ini

Testing framework configuration

```
[tool.pytest.ini_options]
testpaths = ["tests"]
python_files = ["test_*.py"]
python_classes = ["Test*"]
python_functions = ["test_*"]
addopts = "--cov=autoprojectmanagement"
```

### 🔧 .pre-commit-config.yaml

Code quality and linting hooks

```
repos:
- repo: https://github.com/psf/black
  rev: 22.3.0
  hooks:
  - id: black
- repo: https://github.com/pycqa/flake8
  rev: 4.0.1
  hooks:
  - id: flake8
```

# Testing Framework

## Test Architecture

### Unit Tests

Test individual components in isolation

📁 Location: tests/unit/  
% Coverage: 90%+

### Integration Tests

Test component interactions

📁 Location: tests/integration/  
% Coverage: 80%+

### System Tests

Test complete workflows

📁 Location: tests/system/  
% Coverage: 70%+

### Performance Tests

Test performance metrics and memory usage

📁 Location: tests/performance/  
% Coverage: N/A

## Test Configuration

```
# conftest.py
@pytest.fixture
def test_project(tmp_path):
    """Create a test project directory."""
    project_dir = tmp_path / "test_project"
    project_dir.mkdir()
    return project_dir
```

## Running Tests



### Run All Tests

Execute the entire test suite

```
pytest
```



### Run Specific Category

Execute tests from a specific category

```
pytest tests/unit/
```



### Run With Coverage

Generate coverage reports

```
pytest --cov=autoprojectmanagement --cov-report=html
```



### Run With Markers

Execute tests with specific markers

```
pytest -m "not slow"
```



### Run In Parallel

Execute tests in parallel for faster execution

```
pytest -n auto
```

# Development Workflow

## Git Workflow

1 **Branch Strategy**  
main ← develop ← feature/bugfix branches

2 **Commit Convention**  
Format: **type(scope): description**

Type	Format	Example
feat	feat(scope): description	feat(api): add project creation
fix	fix(scope): description	fix(git): resolve merge conflicts
docs	docs(scope): description	docs(readme): update installation

## Development Process


1 **Feature Development**  
Create branch → Develop → Write tests → Run tests

2 **Code Review**  
Automated checks → Peer review → Integration tests

3 **Merge & Deploy**  
Merge to develop → Deploy to staging → Release

```
git checkout -b feature/new-feature develop
# Develop feature
git commit -m "feat(api): add project creation endpoint"
git push origin feature/new-feature
# Create PR to develop
```

## Continuous Integration

 **Triggers:** Push to any branch, Pull request creation

 **Matrix:** Test across multiple Python versions (3.8, 3.9, 3.10, 3.11)

 **Checks:** Linting, Type checking, Testing, Coverage reporting

```
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.8, 3.9, 3.10, 3.11]
```



# Troubleshooting

1	1. Check the power supply and connections.
2	2. Verify the network configuration and IP address.
3	3. Restart the service or application.
4	4. Check the logs for error messages.
5	5. Consult the documentation or support resources.
6	6. Contact the vendor or IT support if the issue persists.
7	7. Document the troubleshooting steps and the resolution.
8	8. Perform a final test to ensure the issue is resolved.
9	9. Communicate the status and resolution to the relevant stakeholders.
10	10. Review the incident to identify any preventive measures.




## ⚙️ Common Issues and Solutions

Issue	Symptom	Solution
Import Errors	ModuleNotFoundError	Check PYTHONPATH, reinstall dependencies
Version Conflicts	Dependency warnings	Use virtual environment, update requirements
Authentication	401 Unauthorized	Configure GitHub token
Test Failures	Assertion errors	Check test data, update assertions

## ⚙️ Debug Configuration


```
{ "version": "0.2.0", "configurations": [ { "name": "Debug Auto Runner", "type": "python", "request": "launch", "module": "autoprojectmanagement.auto_runner", "args": ["--path", "${workspaceFolder}", "--verbose"], "console": "integratedTerminal", "cwd": "${workspaceFolder}", "env": {"PYTHONPATH": "${workspaceFolder}" } }, { "name": "Debug API Server", "type": "python", "request": "launch", "module": "autoprojectmanagement.api.main", "args": ["--reload"], "console": "integratedTerminal", "jinja": true } ] }
```

## 📊 Log Analysis


-  **INFO**  
.auto\_project/logs/info.log - General operation
-  **WARNING**  
.auto\_project/logs/warning.log - Potential issues
-  **ERROR**  
.auto\_project/logs/error.log - Critical problems

# Best Practices


## <> Code Quality Standards

 **Python Style**


Line length: 88 chars  
Type hints required  
Google style docstrings

 **Testing**

Test isolation  
Mock external services  
90%+ coverage target


 **Security**

Regular dependency updates  
Code scanning  
Environment variables


 **Performance**

Parallel test execution  
Lazy loading  
Async operations


## 📄 Documentation Standards

 **Module**


Purpose at top  
Usage examples  
Dependencies

 **Class**

Google style  
Attributes  
Method signatures

 **Function**

Args, returns, raises  
Type hints included  
Usage examples

 **API**

FastAPI auto-docs  
Request/response examples  
Schema documentation


## ⚡ Essential Commands

Task	Command
Setup Environment	./setup_env.sh
Start Development	python -m autoprojectmanagement.auto_runner
Run Tests	pytest
Code Format	black .
Type Check	mypy autoprojectmanagement


## 📁 File Locations

Resource	Path
Source Code	autoprojectmanagement/
Tests	tests/
Documentation	Docs/
Configuration	pyproject.toml
Requirements	requirements*.txt


## 🌐 Support Resources

 **Issues**


GitHub Issues

 **Discussions**

GitHub Discussions

 **Documentation**

Project Wiki

 **Examples**

examples/ directory