🚀 AI Configuration

🔑 OpenAI API Key

••

✅ API Key configured

AI Model

gpt-4o

Selected Model: gpt-4o

🌐 Custom API Base URL (Optional)

http://localhost:11434/v1 (optional)

Advanced Settings

📚 About Multi-Agent Framework

Multi-Agentic System powered by AutoGen with GPT-4o

This framework orchestrates 7 specialized AI agents that collaborate to transform natural language requirements into production-ready code with full documentation, tests, and deployment configuration.

🔄 Agent Pipeline

1. 📋 Requirement Analyst - Structure requirements
2. 💻 Senior Developer - Generate code
3. 🔍 Code Reviewer - Review & iterate (AutoGen loop)
4. 📖 Tech Writer - Create documentation
5. 🧪 QA Engineer - Generate tests
6. 🚀 DevOps - Deployment config
7. 🎨 UI Designer - Streamlit interface

🚀 AutoGen Multi-Agent Code Generator

Transform Ideas into Production-Ready Code with AI Agent Collaboration

AI Agents	Framework	Model	Version
7	Multi-Agent	GPT-4o	2026

📝 Enter Your Requirements

Describe what you want to build:

Create a Fast API REST API for a Todo List Manager with the following features:

1. CRUD Operations:
 - Create new Todo items with title, description, priority (low/medium/high), and due date

💡 Quick Start Examples

🚀 Generate Code with AI Agents

📊 AutoGen Pipeline Results

Generated Artifacts from Multi-Agent Collaboration

📈 Execution Metrics

	Review Iterations	Iteration Limit	Run ID
✅ SUCCESS	2	Within Limit	37e66994

📋 Requirements Analysis | 💻 Python Code | 🔍 Code Review | 📖 Documentation | 🧪 Test Suite | 🚀 Deployment

🧪 Test Suite (pytest)

Generated by QA Engineer Agent

```python
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from unittest.mock import patch, MagicMock
from datetime import datetime, timedelta

from main import app, get_db, Base, TodoModel, StatusEnum, PriorityEnum

# Setup test database
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Create test client
client = TestClient(app)

# Override get_db dependency
def override_get_db():
    try:
        db = TestingSessionLocal()
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db

# Create tables
Base.metadata.create_all(bind=engine)

@pytest.fixture
def sample_todo():
    """Fixture providing a sample todo item."""
    return {
        "title": "Sample Todo",
        "description": "This is a sample todo item.",
        "priority": "medium",
        "due_date": (datetime.utcnow() + timedelta(days=1)).isoformat()
    }

# Happy path tests
def test_create_todo_success(sample_todo):
    """Test creating a new todo item with valid input."""
    response = client.post("/todos", json=sample_todo)
    assert response.status_code == 200
    data = response.json()
    assert data["title"] == sample_todo["title"]
    assert data["priority"] == sample_todo["priority"]

def test_read_todos_success():
    """Test retrieving all todos."""
    response = client.get("/todos")
    assert response.status_code == 200
    assert isinstance(response.json(), list)

def test_read_todo_success(sample_todo):
    """Test retrieving a specific todo item by ID."""
    # Create a todo to retrieve
    create_response = client.post("/todos", json=sample_todo)
    todo_id = create_response.json()["id"]

    response = client.get(f"/todos/{todo_id}")
    assert response.status_code == 200
    data = response.json()
    assert data["id"] == todo_id

# Edge case tests
def test_create_todo_past_due_date(sample_todo):
    """Test creating a todo with a past due date."""
    sample_todo["due_date"] = (datetime.utcnow() - timedelta(days=1)).isoformat()
    response = client.post("/todos", json=sample_todo)
    assert response.status_code == 422
    assert "Due date cannot be in the past" in response.text

def test_read_todo_not_found():
    """Test retrieving a non-existent todo item."""
    response = client.get("/todos/9999")
    assert response.status_code == 404
    assert "Todo not found" in response.text

def test_delete_todo_not_completed(sample_todo):
    """Test deleting a non-completed todo item."""
    # Create a todo to delete
    create_response = client.post("/todos", json=sample_todo)
    todo_id = create_response.json()["id"]

    response = client.delete(f"/todos/{todo_id}")
    assert response.status_code == 400
    assert "Only completed todos can be deleted" in response.text

# Error handling tests
def test_update_todo_invalid_id(sample_todo):
    """Test updating a non-existent todo item."""
    response = client.put("/todos/9999", json=sample_todo)
    assert response.status_code == 404
    assert "Todo not found" in response.text
```

```python

def test_delete_todo_invalid_id():
    """Test deleting a non-existent todo item."""
    response = client.delete("/todos/9999")
    assert response.status_code == 404
    assert "Todo not found" in response.text

# Integration tests
def test_todo_workflow(sample_todo):
    """Test the complete workflow of creating, updating, and deleting a todo."""
    # Create a todo
    create_response = client.post("/todos", json=sample_todo)
    assert create_response.status_code == 200
    todo_id = create_response.json()["id"]

    # Update the todo
    update_data = {"status": "completed"}
    update_response = client.put(f"/todos/{todo_id}", json=update_data)
    assert update_response.status_code == 200
    assert update_response.json()["status"] == "completed"

    # Delete the todo
    delete_response = client.delete(f"/todos/{todo_id}")
    assert delete_response.status_code == 204

# Mock external dependencies
@patch('main.FastAPILimiter.init')
def test_rate_limiter_initialization(mock_init):
    """Test rate limiter initialization."""
    mock_init.return_value = None
    response = client.get("/todos")
    assert response.status_code == 200
    mock_init.assert_called_once()

@patch('main.SessionLocal')
def test_database_error_handling(mock_session):
    """Test handling of database errors."""
    mock_session.side_effect = Exception("Database error")
    response = client.get("/todos")
    assert response.status_code == 500
    assert "Internal Server Error" in response.text
```

📥 Download Tests (Python)

📦 Download All Artifacts (ZIP)