🚀 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

✅ SUCCESS

Review Iterations	Iteration Limit	Run ID
2	Within Limit	37e66994

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

💻 Generated Python Code

Generated by Senior Developer Agent (AutoGen)

```python
import os
import logging
from datetime import datetime
from enum import Enum
from typing import List, Optional

from fastapi import FastAPI, HTTPException, status, Depends, Query
from fastapi.middleware.cors import CORSMiddleware
from sqlalchemy import create_engine, Column, Integer, String, Enum as SQLAEnum, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
from pydantic import BaseModel, validator
from fastapi_limiter import FastAPILimiter
from fastapi_limiter.depends import RateLimiter
from sqlalchemy.exc import SQLAlchemyError
from decouple import config

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Database setup
DATABASE_URL = config('DATABASE_URL', default='sqlite:///./test.db')

# Validate and sanitize the DATABASE_URL
if not DATABASE_URL.startswith(('sqlite://', 'postgresql://', 'mysql://')):
    logger.error("Invalid DATABASE_URL provided.")
    raise ValueError("Invalid DATABASE_URL provided.")

# Conditional configuration for SQLite
if 'sqlite' in DATABASE_URL:
    engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
else:
    engine = create_engine(DATABASE_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

# Enum definitions
class StatusEnum(str, Enum):
    pending = "pending"
    in_progress = "in_progress"
    completed = "completed"

class PriorityEnum(str, Enum):
    low = "low"
    medium = "medium"
    high = "high"

# Database model
class TodoModel(Base):
    __tablename__ = "todos"

    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
    description = Column(String)
    status = Column(SQLAEnum(StatusEnum), default=StatusEnum.pending)
    priority = Column(SQLAEnum(PriorityEnum), default=PriorityEnum.medium)
    due_date = Column(DateTime)
    created_at = Column(DateTime, default=datetime.utcnow)
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)

# Pydantic models
class TodoBase(BaseModel):
    title: str
    description: str
    priority: PriorityEnum
    due_date: datetime

    @validator('due_date')
    def validate_due_date(cls, v: datetime) -> datetime:
        if v < datetime.utcnow():
            raise ValueError('Due date cannot be in the past')
        return v

class TodoCreate(TodoBase):
    pass

class TodoUpdate(TodoBase):
    status: StatusEnum

class TodoResponse(TodoBase):
    id: int
    status: StatusEnum
    created_at: datetime
    updated_at: datetime

    class Config:
        orm_mode = True

# FastAPI app setup
app = FastAPI()

# CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Dependency to get DB session
def get_db() -> Session:
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# Initialize rate limiter
try:
    FastAPILimiter.init()
except Exception as e:
    logger.error(f"Error initializing rate limiter: {e}")
    raise

# CRUD operations
@app.post("/todos", response_model=TodoResponse, dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def create_todo(todo: TodoCreate, db: Session = Depends(get_db)) -> TodoResponse:
    """Create a new todo item."""
    db_todo = TodoModel(**todo.dict())
    db.add(db_todo)
    try:
        db.commit()
        db.refresh(db_todo)
    except SQLAlchemyError as e:
        db.rollback()
        logger.error(f"Error creating todo: {e}")
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Internal Server Error")
    return db_todo

@app.get("/todos", response_model=List[TodoResponse], dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def read_todos(status: Optional[StatusEnum] = None, priority: Optional[PriorityEnum] = None, db: Session = Depends(get_db)) -> List[TodoResponse]:
    """Retrieve all todos with optional filtering by status and priority."""
    query = db.query(TodoModel)
    if status:
        query = query.filter(TodoModel.status == status)
    if priority:
        query = query.filter(TodoModel.priority == priority)
    try:
        todos = query.all()
    except SQLAlchemyError as e:
        logger.error(f"Error reading todos: {e}")
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Internal Server Error")
    return todos

@app.get("/todos/{id}", response_model=TodoResponse, dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def read_todo(id: int, db: Session = Depends(get_db)) -> TodoResponse:
    """Retrieve a specific todo item by ID."""
    db_todo = db.query(TodoModel).filter(TodoModel.id == id).first()
    if db_todo is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Todo not found")
    return db_todo

@app.put("/todos/{id}", response_model=TodoResponse, dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def update_todo(id: int, todo: TodoUpdate, db: Session = Depends(get_db)) -> TodoResponse:
    """Update a specific todo item by ID."""
    db_todo = db.query(TodoModel).filter(TodoModel.id == id).first()
    if db_todo is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Todo not found")
    for key, value in todo.dict(exclude_unset=True).items():
        setattr(db_todo, key, value)
    try:
        db.commit()
        db.refresh(db_todo)
    except SQLAlchemyError as e:
        db.rollback()
        logger.error(f"Error updating todo: {e}")
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Internal Server Error")
    return db_todo

@app.delete("/todos/{id}", status_code=status.HTTP_204_NO_CONTENT, dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def delete_todo(id: int, db: Session = Depends(get_db)) -> None:
    """Delete a specific todo item by ID if it is completed."""
    db_todo = db.query(TodoModel).filter(TodoModel.id == id).first()
    if db_todo is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Todo not found")
    if db_todo.status != StatusEnum.completed:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Only completed todos can be deleted")
    try:
        db.delete(db_todo)
        db.commit()
    except SQLAlchemyError as e:
        db.rollback()
        logger.error(f"Error deleting todo: {e}")
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Internal Server Error")

@app.get("/todos/stats", dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def get_todo_stats(db: Session = Depends(get_db)) -> dict:
    """Retrieve statistics including total, completed, and pending todos."""
    try:
        total = db.query(TodoModel).count()
        completed = db.query(TodoModel).filter(TodoModel.status == StatusEnum.completed).count()
        pending = db.query(TodoModel).filter(TodoModel.status == StatusEnum.pending).count()
    except SQLAlchemyError as e:
        logger.error(f"Error retrieving stats: {e}")
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Internal Server Error")
    return {"total": total, "completed": completed, "pending": pending}

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

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

📥 Download Code

📦 Download All Artifacts (ZIP)

🚀 **Multi-Agentic Framework** © 2026

Powered by AutoGen with GPT-4o • Version 2026.1.0