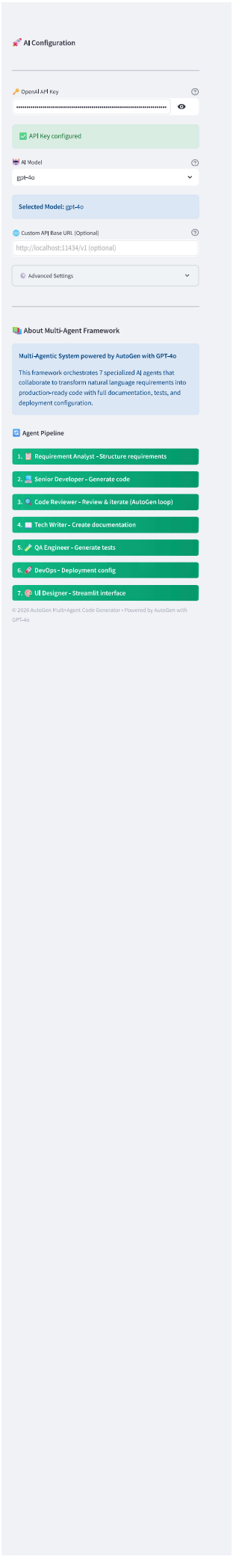

AutoGen Multi-Agent Code Generator

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

Code Review Feedback

Code Needs Revision

Review Status: NEEDS_REVISION

Overall Assessment

The provided code is well-structured and implements most of the functional requirements for a Todo List Manager API using FastAPI and SQLAlchemy. It includes CRUD operations, input validation, error handling, and rate limiting. However, there are areas that need improvement, particularly in security, performance, and adherence to best practices.

Issues Found

- [Severity: HIGH] [Security]: The `DATABASE_URL` validation is insufficient. It only checks the prefix, which could be bypassed with a malicious input. Consider using a more robust validation mechanism (line 15).
- [Severity: MEDIUM] [Performance]: The `read_todos` endpoint fetches all todos without pagination, which could lead to performance issues with large datasets (line 132).
- [Severity: MEDIUM] [Best Practices]: The `FastAPILimiter.init()` is called without specifying a backend, which may lead to unexpected behavior (line 97).
- [Severity: LOW] [Readability]: The `validate_due_date` validator could be more descriptive in its error message (line 88).
- [Severity: LOW] [Testing]: There is no indication of test cases or a testing framework setup, which is critical for maintaining code quality and reliability.

Recommendations

- **Security Improvement**: Enhance the `DATABASE_URL` validation to prevent potential injection attacks. Consider using a library that validates URLs more comprehensively.

```
from urllib.parse import urlparse

def validate_database_url(url: str) -> bool:
    parsed_url = urlparse(url)
    return parsed_url.scheme in ['sqlite', 'postgresql', 'mysql']

if not validate_database_url(DATABASE_URL):
    logger.error("Invalid DATABASE_URL provided.")
    raise ValueError("Invalid DATABASE_URL provided.")
```

- **Performance Enhancement**: Implement pagination for the `read_todos` endpoint to handle large datasets efficiently.

```
@app.get("/todos", response_model=List[TodoResponse], dependencies=[Depends(RateLimiter(times=100, seconds=60))])
def read_todos(skip: int = 0, limit: int = 10, status: Optional[StatusEnum] = None, priority: Optional[PriorityEnum] = None, db: Session = Depends(get_db)) -> List[TodoResponse]:
    query = db.query(TodoModel)
    if status:
        query = query.filter(TodoModel.status == status)
    if priority:
        query = query.filter(TodoModel.priority == priority)
    try:
        todos = query.offset(skip).limit(limit).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
```

- **Best Practices**: Specify a backend for `FastAPILimiter` to ensure consistent rate limiting behavior.

```
from fastapi_limiter.backends.memory import MemoryBackend

try:
    FastAPILimiter.init(backend=MemoryBackend())
except Exception as e:
    logger.error(f"Error initializing rate limiter: {e}")
    raise
```

- **Readability Improvement**: Enhance the error message in the `validate_due_date` validator for clarity.

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

- **Testing**: Set up a testing framework using pytest and create test cases to cover the CRUD operations and edge cases.

Security Concerns

- Insufficient validation of `DATABASE_URL` could lead to security vulnerabilities.

Performance Considerations

- Lack of pagination in the `read_todos` endpoint could degrade performance with large datasets.

The code requires revisions to address the identified issues, particularly in security and performance, to meet the quality standards fully.

Download Review

Download All Artifacts (ZIP)

Multi-Agentic Framework © 2026
Powered by AutoGen with GPT-4o • Version 2026.1.0