🛡️ Project 1 Updated: Secured Simple Python LLM App

💡 *From secured code → to secured container → secured working AI API.*

🎓 **Part of Remoder Inc.'s - AI Engineer Upskilling Program**

Remoder Inc. - remoder.com

WHY?

By implementing these best practices and security layers, we can significantly improve:

- the overall quality
- security of our AI Agent application

It is really essential to build RESPONSIBLE AI-AGENTS. As we go, we will learn from our mistakes and findings on where else we can make our agents better.

If you still have questions on the overall Project-1, please refer to V1 of this document in previous Posts.

If you still have questions on the overall Project-1, please refer to V1 of this document in previous Posts.

There are few things that changed from original version of this document. The changes are mostly made for security enhancements. So, in a nutshell, we changed the following:

- Dockerfile Enhancements 
- FastAPI Security and Configuration 🛡️ => app.py
- Dependency Management 📦 => requirements.txt

Dockerfile Enhancements

- Current Dockerfile is a good start, but it can be improved for better security and efficiency. To do that we basically now:

Use a Specific Base Image

Instead of using a general **ubuntu:20.04** image, we now use **python:3.9-slim**. This will significantly **reduce the size of our image** and **minimize the attack surface by excluding unnecessary packages**.

Run as a Non-Root User

Running containers with root privileges is a major security risk. We can create a dedicated user with limited permissions to mitigate this.

Benefits of a Multi-Stage Build and pip-audit

remoder-lab1 > docker > pr1-secure-simple-llm > Dockerfile > ...

```
1   # Stage 1: The "builder" stage for auditing dependencies
2   # FIX: Changed 'as' to 'AS' for consistent casing
3   FROM python:3.9-slim AS builder (last pushed 1 week ago)
4
5   WORKDIR /app
6
7   # Copy requirements first to leverage Docker's layer caching
8   COPY requirements.txt .
9
10  # Install dependencies and the audit tool
11  RUN pip install --no-cache-dir -r requirements.txt
12
13  # 🛡️ Run the audit. If vulnerabilities are found, the build wil
14  RUN pip-audit
15
16
17  # Stage 2: The final, secure production image
18  FROM python:3.9-slim (last pushed 1 week ago)
```

1. **Security Through Separation** 🛡️
2. **Dependency Auditing** ✅
3. **Reduced Attack Surface** 🔒
4. **Smaller Image Size** 📦

(screenshot does not show full Dockerfile!)

FastAPI Security and Configuration

Input Validation and Sanitization

FastAPI has several built-in features that can secure our application. While **Pydantic** handles basic data validation, we can add more specific constraints to our **Prompt** model to prevent malicious inputs. For example, we can limit the length of the input text to avoid resource exhaustion.

Python

```python
from pydantic import BaseModel, constr

class Prompt(BaseModel):
    text: constr(min_length=1, max_length=500)
```

```python
from fastapi import FastAPI, Request
from slowapi import Limiter, _rate_limit_exceeded_handler
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app = FastAPI()
app.state.limiter = limiter
app.add_exception_handler(_rate_limit_exceeded_handler)

@app.post("/generate/")
@limiter.limit("5/minute")
def generate_text(request: Request, prompt: Prompt):
    # Your text generation logic here
    pass
```

Implement Rate Limiting

To protect our API from denial-of-service (DoS) attacks, we can implement rate limiting. This will restrict the number of requests a user can make in a given time frame. We can use a library like slowapi to easily add this functionality.

1. We add **slowapi** to our requirements.txt
2. We integrate the **slowapi** to app.py

Dependency Management 📦

Managing our dependencies effectively is crucial for maintaining a secure and stable application.

Pin Dependencies

Our requirements.txt file is a good start, but we can make it more robust by pinning the exact versions of all our dependencies, including transitive ones. This ensures that our application will always build with the same set of packages, preventing unexpected breaking changes.

We can generate a detailed requirements.txt file using **pip freeze**:

```
pip freeze > requirements.txt
```

Regularly Scan for Vulnerabilities

It's important to regularly scan our dependencies for known vulnerabilities. We can use a tool like **pip-audit** to automate this process. In this demo, I have just added the **pip-audit** step into the Dockerfile and I am running it as a separate **STAGE** before building the final **PROD** version of the image. Refer back to **SLIDE-4.**

Questions?

Contact us & come learn with us ٩(◕‿◕。)۶

- https://www.linkedin.com/company/remoder
- https://www.linkedin.com/in/sanjars/
- https://www.youtube.com/@remoder-inc
- remoder.com
- **Full walkthrough of this project is available on our “Master AI Deployment “ course**