

FastAPI Gemini AI Integration

peedaluk

June 30, 2025

1 Introduction

The **FastAPI Gemini AI Integration** project demonstrates how to build a secure, scalable REST API for generative AI using Google's Gemini model and Python's FastAPI framework. The system provides AI-powered text generation services via HTTP endpoints, enabling seamless integration of large language models into modern applications.

2 Objectives

- Develop a robust backend API for generative AI using FastAPI.
- Securely integrate Google Gemini AI for advanced text generation.
- Provide a modular, extensible architecture for future AI enhancements.
- Document and demonstrate API usage with interactive documentation and example requests.

3 System Architecture

- **API Layer:** Built with FastAPI, exposes RESTful endpoints for client interaction.
- **AI Integration:** Communicates with Google Gemini via the official SDK and API key.
- **Security:** Uses environment variables for API key management to prevent credential leaks.
- **Documentation:** Auto-generates OpenAPI/Swagger UI at `/docs` for interactive exploration.

4 Project Structure

- `src/`: Main application source code
- `screenshots/`: Screenshots of API usage and docs
- `requirements.txt`: Python dependencies
- `README.md`: Project documentation

5 Implementation Details

- **API Endpoint /chat:** Accepts a POST request with a text prompt and returns the Gemini AI-generated response.
- **Environment Setup:** Requires a `.env` file with `GOOGLE_API_KEY` for secure authentication.
- **Error Handling:** Returns appropriate error messages for missing keys or API failures.
- **Extensibility:** The modular codebase allows for easy addition of new endpoints or AI models.

6 Usage and Results

Setup

1. Clone the repository and install dependencies:

```
git clone https://github.com/peedaluk/fastapi-gemini-ai.git
cd fastapi-gemini-ai
pip install -r requirements.txt
```

2. Add your Google Gemini API key to a `.env` file:

```
GOOGLE_API_KEY=your_google_gemini_api_key
```

3. Start the server:

```
uvicorn src.main:app --reload
```

4. Access the interactive documentation at <http://localhost:8000/docs>

Example Request

- **Endpoint:** POST `/chat`
- **Request Body:**

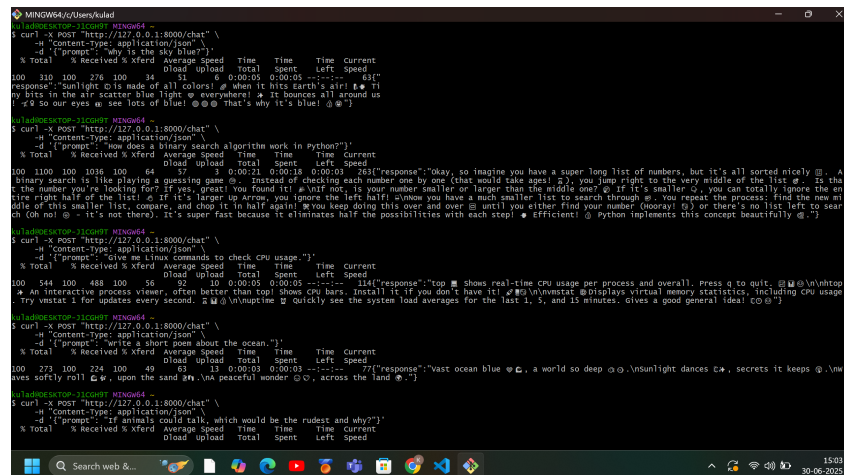
```
{ "text": "Explain the concept of generative AI." }
```

- **Sample Response:**

```
{ "response": "Generative AI refers to..." }
```

7 Screenshots

Sample Chat Request



```
MINGW64/C/Users/ruled
$ curl -X POST "http://127.0.0.1:8000/chat" \
  -H "Content-Type: application/json" \
  -d '{"prompt": "why is the sky blue?"}'
{"Total": {"Received": 100, "Xferd": 100, "Average Speed": 100, "Time": 100, "Time Current": 100, "Time Spent": 100, "Time Left": 100}, "response": "Sunlight is made of all colors! When it hits earth's air, the tiny bits in the air scatter blue light everywhere! It bounces all around us! So our eyes see lots of blue! That's why it's blue!"}

$ curl -X POST "http://127.0.0.1:8000/chat" \
  -H "Content-Type: application/json" \
  -d '{"prompt": "How does a binary search algorithm work in Python?"}'
{"Total": {"Received": 100, "Xferd": 100, "Average Speed": 100, "Time": 100, "Time Current": 100, "Time Spent": 100, "Time Left": 100}, "response": "Okay, so imagine you have a super long list of numbers, but it's all sorted nicely. A binary search is like playing a guessing game. Instead of checking each number one by one (that would take ages!), you jump right to the very middle of the list. Is the number you're looking for? If yes, great! You found it! If not, is your number smaller or larger than the middle one? If it's smaller, you can totally ignore the entire right half of the list. If it's larger, you ignore the left half. Now you have a much smaller list to search through! You repeat the process: find the new middle of this smaller list, compare, and chop it in half again! You keep doing this over and over until you either find your number (hooray!) or there's no list left to search (Oh no! - it's not there). It's super fast because it eliminates half the possibilities with each step! Efficient! Python implements this concept beautifully!"}

$ curl -X POST "http://127.0.0.1:8000/chat" \
  -H "Content-Type: application/json" \
  -d '{"prompt": "Give me Linux commands to check CPU usage."}'
{"Total": {"Received": 100, "Xferd": 100, "Average Speed": 100, "Time": 100, "Time Current": 100, "Time Spent": 100, "Time Left": 100}, "response": "top shows real-time CPU usage per process and overall. Press q to quit. htop is an interactive process viewer, often better than top! Shows CPU bars. Install it if you don't have it! vmstat displays virtual memory statistics, including CPU usage. Try vmstat 1 for updates every second. iostat Quickly see the system load averages for the last 1, 5, and 15 minutes. Gives a good general idea!"}

$ curl -X POST "http://127.0.0.1:8000/chat" \
  -H "Content-Type: application/json" \
  -d '{"prompt": "Write a short poem about the ocean."}'
{"Total": {"Received": 100, "Xferd": 100, "Average Speed": 100, "Time": 100, "Time Current": 100, "Time Spent": 100, "Time Left": 100}, "response": "Vast ocean blue, a world so deep, \nSunlight dances there, secrets it keeps. \nWaves safely trail & surge upon the sand, \nA peaceful wonder across the land."}

$ curl -X POST "http://127.0.0.1:8000/chat" \
  -H "Content-Type: application/json" \
  -d '{"prompt": "If animals could talk, which would be the rudest and why?"}'
{"Total": {"Received": 100, "Xferd": 100, "Average Speed": 100, "Time": 100, "Time Current": 100, "Time Spent": 100, "Time Left": 100}, "response": "If animals could talk, which would be the rudest and why?"}
```

8 Evaluation

- **Performance:** FastAPI ensures low-latency responses and efficient handling of concurrent requests.
- **Security:** API keys are never hardcoded; environment variables are used throughout.
- **Usability:** The /docs endpoint allows for rapid testing and onboarding of new users or developers.
- **Extensibility:** The architecture supports the addition of new AI models, endpoints, or business logic with minimal changes.

9 Conclusion and Future Work

This project successfully demonstrates how to integrate state-of-the-art generative AI into a modern web API using FastAPI. The design prioritizes security, modularity, and ease of use. Future enhancements could include:

- Support for Gemini's multimodal (image/text) endpoints.
- User authentication and request rate limiting.
- Persistent conversation history and advanced context handling.
- Deployment to cloud platforms with CI/CD pipelines.

10 References

- Project GitHub Repository: <https://github.com/peedaluk/fastapi-gemini-ai>
- Google Gemini API Documentation: <https://ai.google.dev/models/gemini>
- FastAPI Documentation: <https://fastapi.tiangolo.com/>
- Implementing Gemini AI with Python: A Simple Guide: <https://blog.stackademic.com/implementing-gemini-ai-with-python-a-simple-guide-71f8c148d24a>