:

**📖 Story Generation Application with FastAPI and Ollama**

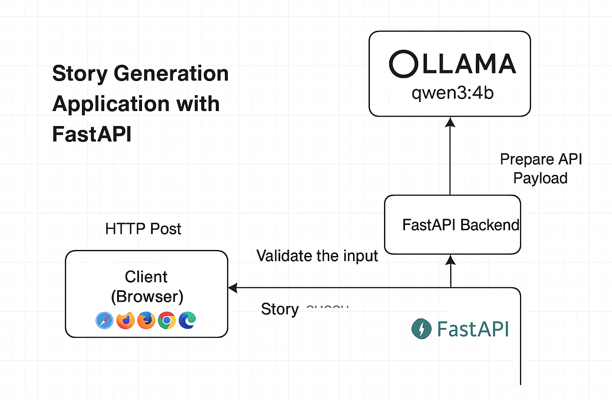
**📌 Overview**

This project is a **FastAPI-based backend service** that integrates with **Ollama’s LLM models** (e.g., qwen3:4b) to generate **creative stories** from user-provided prompts.  
It includes robust **error handling** to manage invalid inputs, timeouts, and Ollama API failures gracefully.

**🏗️ Architecture**

**Flow:**

1. **Client (Browser)** sends HTTP POST request with a story prompt.
2. **FastAPI App** validates the input → prepares payload → calls **Ollama API**.
3. **Ollama Model (qwen3:4b)** processes the request and returns a generated story.
4. **FastAPI App** extracts the response and sends it back.
5. **Client** displays the story.



**⚙️ Tech Stack**

* **Backend**: FastAPI (Python 3.9+)
* **LLM Engine**: Ollama (qwen3:4b)
* **HTTP Client**: requests
* **Web Server**: Uvicorn

**📂 Project Structure**

story-generator-fastapi/

│

├── main.py # FastAPI app with error handling

├── requirements.txt # Python dependencies

├── README.md # Project overview

└── docs.md # Detailed documentation

**📦 Installation**

(Same steps as before, with virtualenv + pip + Ollama installation.)

**▶️ Running the Application**

1. Start Ollama server:

ollama serve

1. Run FastAPI app:

uvicorn main:app --reload --host 0.0.0.0 --port 8000

**📡 API Endpoints**

**Generate Story**

**POST** /generate-story

**Request Body**

{

"prompt": "A brave knight enters a mysterious cave",

"max\_tokens": 300

}

**Example cURL**

curl -X POST "http://127.0.0.1:8000/generate-story" \

-H "Content-Type: application/json" \

-d '{"prompt":"A brave knight enters a mysterious cave","max\_tokens":200}'

**Response (Success)**

{

"story": "Once upon a time, a brave knight ventured into the cave..."

}

**Response (Error Example)**

{

"detail": "Ollama API request failed: Connection timed out"

}

**🔒 Error Handling**

The app handles errors with proper HTTP status codes:

* **400** → Invalid input (empty prompt, negative tokens, etc.)
* **500** → Ollama API errors (timeout, invalid response, server crash)
* **422** → Validation errors (handled automatically by FastAPI)

**📝 Updated OLLAMA3.py with Error Handling**

from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel, Field

import requests

app = FastAPI()

# Enable CORS if frontend will call this API

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

OLLAMA\_API\_URL = "http://localhost:11434/api/generate"

OLLAMA\_MODEL = "qwen3:4b"

class StoryRequest(BaseModel):

prompt: str = Field(..., min\_length=5, description="Prompt must be at least 5 characters long")

max\_tokens: int = Field(500, ge=50, le=2000, description="Number of tokens (50–2000)")

@app.post("/generate-story")

def generate\_story(request: StoryRequest):

# Validate input explicitly

if not request.prompt.strip():

raise HTTPException(status\_code=400, detail="Prompt cannot be empty")

payload = {

"model": OLLAMA\_MODEL,

"prompt": f"Write a creative story based on: {request.prompt}",

"options": {"num\_predict": request.max\_tokens},

"stream": False

}

try:

response = requests.post(OLLAMA\_API\_URL, json=payload, timeout=60)

except requests.exceptions.Timeout:

raise HTTPException(status\_code=500, detail="Ollama API request timed out")

except requests.exceptions.RequestException as e:

raise HTTPException(status\_code=500, detail=f"Ollama API request failed: {str(e)}")

if response.status\_code != 200:

raise HTTPException(status\_code=response.status\_code, detail=response.text)

try:

data = response.json()

story\_text = data.get("response", "")

except Exception:

raise HTTPException(status\_code=500, detail="Invalid JSON response from Ollama")

if not story\_text:

raise HTTPException(status\_code=500, detail="Ollama returned an empty story")

return {"story": story\_text}

**📜 Requirements**

fastapi

uvicorn

requests

**🚀 Future Enhancements**

* Support **streaming responses** for real-time story generation.
* Add **logging middleware** for request/response tracking.
* Integrate with a frontend app (React / Vue).
* Multi-model support with Ollama.