# Complete Project Guide: AI Medical Prescription Verification System

### Project Overview

This document provides a step-by-step guide to building a full-stack AI Medical Prescription Verification System. The system features a **FastAPI backend** for business logic and a **Streamlit frontend** for a user-friendly interface.

**Key features include:**

* **NLP-Based Drug Information Extraction:** Parsing unstructured prescription text.
* **Drug Interaction Detection:** Identifying potential harmful drug combinations.
* **Age-Specific Dosage Recommendation:** Providing safe dosage guidelines.
* **Alternative Medication Suggestions:** Offering safer or equivalent drug options.

### Prerequisites

Before you begin, make sure you have the following installed on your system:

* **Python 3.8+**: The programming language for this project.
* **VS Code (or any code editor)**: To write and manage your code.
* **A Terminal or Command Prompt**: To run the applications.

### Step 1: Set Up Your Project Folder

1. Create a new, empty folder on your computer for this project (e.g., medical-verifier).
2. Open this folder in VS Code.

### Step 2: Create the FastAPI Backend

This code will act as the "brain" of your application. It uses FastAPI to create a robust API that will handle all the data processing and logical checks.

File Name: main.py

Copy the following code and save it as main.py inside your project folder.

# main.py  
# This file contains the FastAPI backend for the medical prescription verification system.  
  
from fastapi import FastAPI, HTTPException  
from pydantic import BaseModel  
from typing import List, Dict, Any, Optional  
  
# Mockup of a Hugging Face model pipeline for NLP  
# In a real application, you would load a specialized model (e.g., ClinicalBERT)  
# For this example, we'll simulate the output of a Named Entity Recognition (NER) model.  
def mock\_nlp\_prescription\_parser(text: str) -> List[Dict[str, Any]]:  
 """  
 Simulates an NLP model to extract drug details.  
   
 Args:  
 text (str): The unstructured medical prescription text.  
  
 Returns:  
 List[Dict[str, Any]]: A list of dictionaries with extracted entities.  
 """  
 mock\_entities = []  
 # Simple keyword-based extraction for demonstration  
 if "amoxicillin" in text.lower():  
 mock\_entities.append({"drug\_name": "Amoxicillin", "dosage\_mg": 500, "frequency": "twice daily"})  
 if "ibuprofen" in text.lower():  
 mock\_entities.append({"drug\_name": "Ibuprofen", "dosage\_mg": 400, "frequency": "as needed"})  
 if "aspirin" in text.lower():  
 mock\_entities.append({"drug\_name": "Aspirin", "dosage\_mg": 81, "frequency": "once daily"})  
 return mock\_entities  
  
# Mockup of a drug database  
# In a real-world scenario, this would be a large, comprehensive database  
# or a service like IBM Micromedex.  
MOCK\_DRUG\_DATABASE = {  
 "amoxicillin": {  
 "therapeutic\_class": "Antibiotic",  
 "contraindications": {"penicillin allergy"},  
 "age\_dosages": {  
 "infant": "25-50 mg/kg/day",  
 "child": "20-40 mg/kg/day",  
 "adult": "250-875 mg per dose",  
 },  
 "alternative\_drugs": ["Azithromycin", "Cefalexin"]  
 },  
 "ibuprofen": {  
 "therapeutic\_class": "NSAID",  
 "contraindications": {"stomach ulcers", "kidney disease"},  
 "age\_dosages": {  
 "child": "5-10 mg/kg per dose",  
 "adult": "200-800 mg per dose",  
 },  
 "alternative\_drugs": ["Acetaminophen", "Naproxen"]  
 },  
 "aspirin": {  
 "therapeutic\_class": "Antiplatelet",  
 "contraindications": {"stomach ulcers", "bleeding disorders", "children with viral infections"},  
 "age\_dosages": {  
 "child": "Not recommended under 16", # Reye's syndrome risk  
 "adult": "81-325 mg daily",  
 },  
 "alternative\_drugs": ["Clopidogrel", "Warfarin"]  
 }  
}  
  
app = FastAPI()  
  
# Pydantic models for request and response validation  
class PrescriptionText(BaseModel):  
 text: str  
 patient\_age: int  
  
class DrugDetails(BaseModel):  
 drug\_name: str  
 dosage\_mg: Optional[int]  
 frequency: Optional[str]  
  
class PrescriptionAnalysis(BaseModel):  
 parsed\_drugs: List[DrugDetails]  
 interactions: List[str]  
 dosage\_recommendations: Dict[str, str]  
 alternative\_suggestions: Dict[str, List[str]]  
 warnings: List[str]  
  
# --- FastAPI Endpoints ---  
  
@app.post("/analyze\_prescription/", response\_model=PrescriptionAnalysis)  
async def analyze\_prescription(prescription: PrescriptionText):  
 """  
 Main endpoint to analyze a full prescription text.  
 It orchestrates all verification steps.  
 """  
 # 1. NLP-Based Drug Information Extraction  
 parsed\_drugs = mock\_nlp\_prescription\_parser(prescription.text)  
   
 # Extract drug names for further analysis  
 drug\_names = [d['drug\_name'].lower() for d in parsed\_drugs]  
   
 # Prepare the response object  
 response\_data = {  
 "parsed\_drugs": parsed\_drugs,  
 "interactions": [],  
 "dosage\_recommendations": {},  
 "alternative\_suggestions": {},  
 "warnings": [],  
 }  
  
 # 2. Drug Interaction Detection System  
 if len(drug\_names) > 1:  
 # Simple interaction check  
 for i in range(len(drug\_names)):  
 for j in range(i + 1, len(drug\_names)):  
 drug1 = drug\_names[i]  
 drug2 = drug\_names[j]  
   
 # Mock interaction logic  
 if (drug1 == "amoxicillin" and drug2 == "aspirin") or \  
 (drug1 == "aspirin" and drug2 == "amoxicillin"):  
 response\_data["interactions"].append(  
 "Potential interaction between Amoxicillin and Aspirin. Consult a doctor."  
 )  
   
 # 3. Age-Specific Dosage Recommendation & 4. Alternative Medication Suggestions  
 for drug\_detail in parsed\_drugs:  
 drug\_name = drug\_detail['drug\_name'].lower()  
 if drug\_name in MOCK\_DRUG\_DATABASE:  
 drug\_info = MOCK\_DRUG\_DATABASE[drug\_name]  
   
 # Age-specific dosage  
 age = prescription.patient\_age  
 dosage\_key = "adult"  
 if 0 < age < 16:  
 dosage\_key = "child"  
 elif age <= 0:  
 dosage\_key = "infant"  
   
 if dosage\_key in drug\_info["age\_dosages"]:  
 recommended\_dosage = drug\_info["age\_dosages"][dosage\_key]  
 response\_data["dosage\_recommendations"][drug\_detail['drug\_name']] = recommended\_dosage  
   
 # Alternative medication suggestions  
 response\_data["alternative\_suggestions"][drug\_detail['drug\_name']] = drug\_info["alternative\_drugs"]  
   
 # Check for contraindications  
 if "aspirin" == drug\_name and age < 16:  
 response\_data["warnings"].append(  
 "WARNING: Aspirin is not recommended for patients under 16 due to the risk of Reye's syndrome."  
 )  
   
 return response\_data

### Step 3: Create the Streamlit Frontend

This code provides the visual interface. It collects user input and sends it to the FastAPI backend for analysis.

File Name: app.py

Copy the following code and save it as app.py in the same project folder.

# app.py  
# This file contains the Streamlit frontend.  
  
import streamlit as st  
import requests  
import json  
import time  
  
# --- Streamlit UI Components ---  
  
st.set\_page\_config(page\_title="Prescription Verification", layout="centered")  
  
st.title("AI Medical Prescription Verification")  
st.markdown("Enter a prescription text and patient age to get an immediate AI-driven analysis.")  
  
# Input fields  
prescription\_text = st.text\_area(  
 "Prescription Text",  
 placeholder="e.g., 'Take 1 tablet of Amoxicillin 500mg, twice daily for 7 days. Also, take Ibuprofen as needed.'"  
)  
patient\_age = st.number\_input(  
 "Patient Age",  
 min\_value=0,  
 max\_value=120,  
 value=30  
)  
  
# Button to trigger analysis  
if st.button("Analyze Prescription"):  
 if not prescription\_text:  
 st.warning("Please enter some prescription text to analyze.")  
 else:  
 # Create a loading spinner while the request is being processed  
 with st.spinner("Analyzing prescription..."):  
 try:  
 # Prepare the data to be sent to the FastAPI backend  
 payload = {  
 "text": prescription\_text,  
 "patient\_age": patient\_age  
 }  
   
 # Make the POST request to the FastAPI endpoint  
 response = requests.post("http://localhost:8000/analyze\_prescription/", json=payload)  
 response.raise\_for\_status() # Raises an HTTPError for bad responses  
   
 analysis\_results = response.json()  
   
 st.success("Analysis complete!")  
   
 st.markdown("---")  
 st.header("Results")  
   
 # Display parsed drugs (from NLP extraction)  
 st.subheader("Extracted Medications")  
 if analysis\_results["parsed\_drugs"]:  
 for drug in analysis\_results["parsed\_drugs"]:  
 st.info(f"\*\*Drug:\*\* {drug.get('drug\_name')} | \*\*Dosage:\*\* {drug.get('dosage\_mg')} mg | \*\*Frequency:\*\* {drug.get('frequency')}")  
 else:  
 st.warning("No drugs were detected in the prescription text.")  
   
 # Display warnings and contraindications  
 st.markdown("---")  
 st.subheader("Warnings & Contraindications")  
 if analysis\_results["warnings"]:  
 for warning in analysis\_results["warnings"]:  
 st.error(warning, icon="🚨")  
 else:  
 st.success("No critical warnings or contraindications found.")  
   
 # Display drug interactions  
 st.markdown("---")  
 st.subheader("Drug Interactions")  
 if analysis\_results["interactions"]:  
 for interaction in analysis\_results["interactions"]:  
 st.warning(interaction, icon="⚠️")  
 else:  
 st.success("No significant drug interactions detected.")  
   
 # Display dosage recommendations  
 st.markdown("---")  
 st.subheader("Age-Specific Dosage Recommendations")  
 if analysis\_results["dosage\_recommendations"]:  
 for drug, dosage in analysis\_results["dosage\_recommendations"].items():  
 st.write(f"- \*\*{drug}:\*\* {dosage}")  
 else:  
 st.info("No dosage recommendations available for the detected medications.")  
   
 # Display alternative suggestions  
 st.markdown("---")  
 st.subheader("Alternative Medication Suggestions")  
 if analysis\_results["alternative\_suggestions"]:  
 for drug, alternatives in analysis\_results["alternative\_suggestions"].items():  
 st.write(f"- For \*\*{drug}\*\*, consider: {', '.join(alternatives)}")  
 else:  
 st.info("No alternative medication suggestions available.")  
  
 except requests.exceptions.ConnectionError:  
 st.error("Could not connect to the FastAPI backend. Please make sure the backend server is running.")  
 except requests.exceptions.HTTPError as e:  
 st.error(f"Error from the server: {e}")  
 except Exception as e:  
 st.error(f"An unexpected error occurred: {e}")

### Step 4: Run the Project

You need to run both the backend and frontend in separate terminal windows.

1. **Open two terminal windows** in VS Code (Terminal > New Terminal).
2. In the **first terminal**, install the required libraries if you haven't already:  
   pip install "fastapi[all]" streamlit requests
3. In the **same first terminal**, run the FastAPI backend:  
   uvicorn main:app --reload
4. In the **second terminal**, run the Streamlit frontend:  
   streamlit run app.py
5. Your browser will automatically open a new tab with the Streamlit application. You can now use the interface to interact with your system.

### Step 5: Next Steps for Enhancements

* **NLP Model Integration:** Replace the mock\_nlp\_prescription\_parser function with a real Hugging Face pipeline for medical NER.
* **Real Drug Databases:** Integrate with actual drug APIs or comprehensive databases to get real-time, up-to-date drug interaction, dosage, and alternative medication data.
* **User Authentication:** Add a login system to manage user profiles and save prescription history.
* **Scalability:** Consider deploying your FastAPI backend on a cloud platform like Google Cloud, AWS, or Azure for public access.