AI Medical Prescription Verification Leveraging IBM Watson and Hugging Face Models

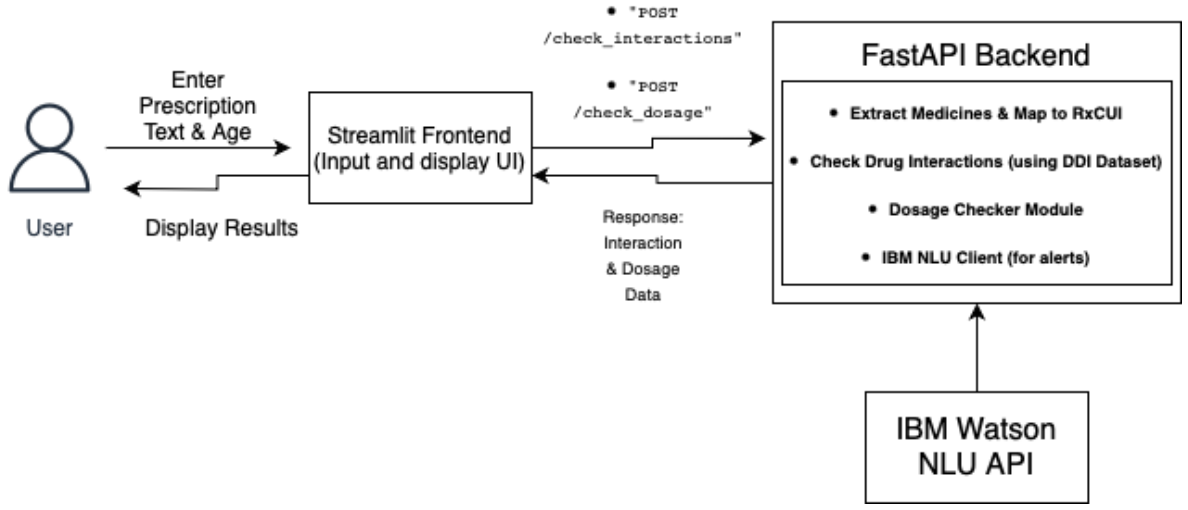
Project Description:

This project aims to analyze drug interactions, identify correct drug dosages, and provide safe alternative medication options based on age and drug details. It integrates multiple datasets and leverages advanced NLP models and APIs for accurate drug information extraction and interaction understanding. The system is built with a FastAPI backend and a Streamlit frontend for easy user interaction.

Scenarios:

Scenario 1: A healthcare provider inputs a list of drugs prescribed to a patient and gets potential interactions along with scientific drug names and RxCUIs.
Scenario 2: A pharmacist wants to verify the correct dosage of a prescribed drug and check for safe alternatives based on patient age.
Scenario 3: A patient uses the system to understand the interaction details and get safe dosage recommendations for their medications.

Architectural Diagram



Pre-requisites

1. **Python 3.8+**
   - Installation guide: Python Official Site
   - Verify installation: python --version
2. **FastAPI Framework**
   - Official docs: FastAPI
   - Install with pip install fastapi
3. **Streamlit for Frontend**
   - Official docs: Streamlit
   - Install with pip install streamlit
4. **Hugging Face API Key** (for samant/medical-ner model)
   - Sign up at Hugging Face
   - Obtain API key to extract drug names and dosages
5. **IBM Watson NLP API**
   - Sign up at IBM Cloud
   - Get API key and URL for NLP-based interaction understanding
6. **RxNorm API Keys**
   - Sign up for RxNav API
   - Used to map RxCUI to drug dosages and alternatives
7. **Additional Python Libraries**
   - requests, pandas, numpy, fastapi, streamlit
   - Install via pip install -r requirements.txt

Project Flow:

Milestone 1: Data Acquisition and Integration
Activity 1.1: Dataset Download
Activity 1.2: Dataset mapping and Preparation

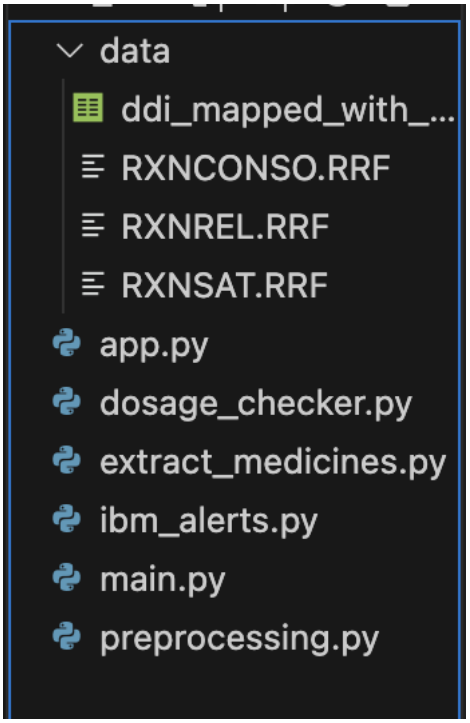
Milestone 2: NLP Model Integration for Drug Extraction and Interaction Understanding
Activity 2.1: Named Entity Recognition
Activity 2.2: IBM Watson NLP for Interaction Context
Activity 3.3: Integration of both the models

Milestone 3: Dosage Verification and Alternative Recommendations
Activity 3.1: RxNorm API Usage
Activity 3.2: Alternative Safe Drug Suggestions

Milestone 4: Backend and Frontend Development
Activity 4.1: FastAPI Backend
Activity 4.2: Streamlit Frontend

Folder Architecture:

Under the folder name, AI_Prescription_Verifier



Milestone 1: Data Acquisition and Integration

This milestone focused on collecting and preparing high-quality datasets essential for accurate drug interaction analysis. We sourced raw interaction data from Kaggle and standardized drug details using the RXNORM dataset from the U.S. National Library of Medicine. By mapping and merging these datasets through RxCUIs, we created a comprehensive and scientifically consistent foundation to power subsequent analysis and model development.

Activity 1.1: Dataset Download

The first step involved acquiring reliable datasets related to drug interactions and drug information.

- Drug Interaction Dataset from Kaggle
  We downloaded the **Drug-Drug Interactions** dataset from Kaggle, which contains pairs of drugs and their known interactions. This dataset provides raw data about which drugs interact with each other, but it mainly uses commercial or common drug names without standardized scientific identifiers.
  Dataset Link: [Drug-Drug Interactions on Kaggle]

Table data was imported. Adjust Se

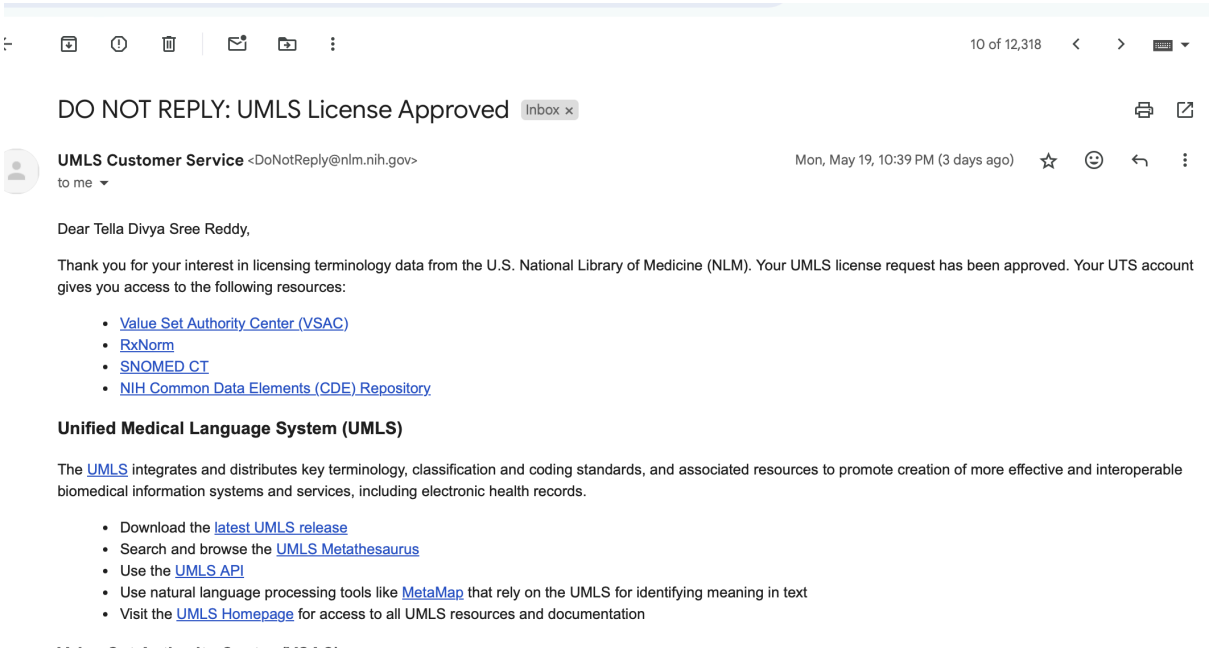
db_drug_interactions

Drug 1	Drug 2	Interaction Description
Trioxsalen	Verteporfin	Trioxsalen may increase the photosensitizing activities of Verteporfin.
Aminolevulinic acid	Verteporfin	Aminolevulinic acid may increase the photosensitizing activities of Verteporfin.
Titanium dioxide	Verteporfin	Titanium dioxide may increase the photosensitizing activities of Verteporfin.
Tiaprofenic acid	Verteporfin	Tiaprofenic acid may increase the photosensitizing activities of Verteporfin.
Cyamemazine	Verteporfin	Cyamemazine may increase the photosensitizing activities of Verteporfin.
Temoporfin	Verteporfin	Temoporfin may increase the photosensitizing activities of Verteporfin.
Methoxsalen	Verteporfin	Methoxsalen may increase the photosensitizing activities of Verteporfin.
Hexaminolevulinate	Verteporfin	Hexaminolevulinate may increase the photosensitizing activities of Verteporfin.
Benzophenone	Verteporfin	Benzophenone may increase the photosensitizing activities of Verteporfin.
Riboflavin	Verteporfin	Riboflavin may increase the photosensitizing activities of Verteporfin.
Carprofen	Verteporfin	Carprofen may increase the photosensitizing activities of Verteporfin.
Cyclophosphamide	Verteporfin	Cyclophosphamide may increase the cardiotoxic activities of Verteporfin.
Paclitaxel	Verteporfin	The risk or severity of adverse effects can be increased when Paclitaxel is combined with Verteporfin.
Docetaxel	Verteporfin	The risk or severity of adverse effects can be increased when Docetaxel is combined with Verteporfin.
Cabazitaxel	Verteporfin	The risk or severity of adverse effects can be increased when Cabazitaxel is combined with Verteporfin.
Aminolevulinic acid	Digoxin	Aminolevulinic acid may decrease the cardiotoxic activities of Digoxin.
Temoporfin	Digoxin	Temoporfin may decrease the cardiotoxic activities of Digoxin.
Cyclophosphamide	Digoxin	Cyclophosphamide may decrease the cardiotoxic activities of Digoxin.
Paclitaxel	Digoxin	Paclitaxel may decrease the cardiotoxic activities of Digoxin.
Docetaxel	Digoxin	Docetaxel may decrease the cardiotoxic activities of Digoxin.
Cabazitaxel	Digoxin	Cabazitaxel may decrease the cardiotoxic activities of Digoxin.
Sulpiride	Digoxin	The risk or severity of adverse effects can be increased when Sulpiride is combined with Digoxin.

Fig. Drug-Drug Interactions Dataset

- RXNORM ONE Dataset from U.S. National Library of Medicine
  To obtain detailed and standardized drug information, we registered for access to

the **RXNORM ONE** dataset through the Unified Terminology Services (UTS) of the U.S. National Library of Medicine. After submitting a request for access on the UTS website (https://uts.nlm.nih.gov/), our application was approved, allowing us to download the latest RXNORM dataset (2025 release).



10 of 12,318

DO NOT REPLY: UMLS License Approved Inbox ×

UMLS Customer Service <DoNotReply@nlm.nih.gov> Mon, May 19, 10:39 PM (3 days ago)
to me

Dear Tella Divya Sree Reddy,

Thank you for your interest in licensing terminology data from the U.S. National Library of Medicine (NLM). Your UMLS license request has been approved. Your UTS account gives you access to the following resources:

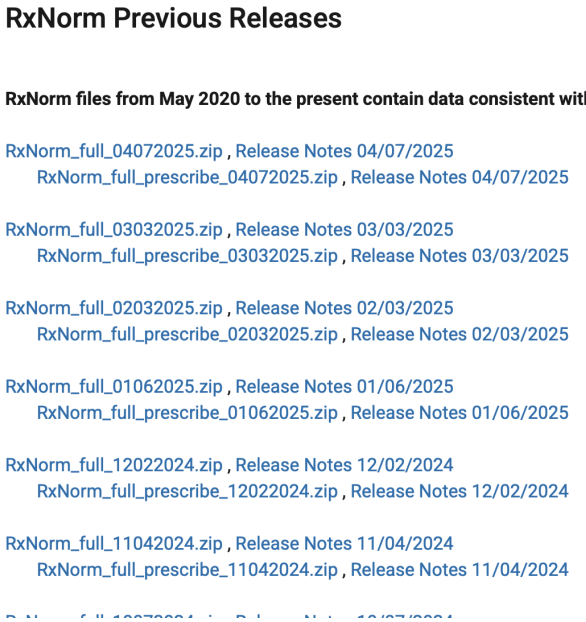
- Value Set Authority Center (VSAC)
- RxNorm
- SNOMED CT
- NIH Common Data Elements (CDE) Repository

Unified Medical Language System (UMLS)

The UMLS integrates and distributes key terminology, classification and coding standards, and associated resources to promote creation of more effective and interoperable biomedical information systems and services, including electronic health records.

- Download the latest UMLS release
- Search and browse the UMLS Metathesaurus
- Use the UMLS API
- Use natural language processing tools like MetaMap that rely on the UMLS for identifying meaning in text
- Visit the UMLS Homepage for access to all UMLS resources and documentation

- This dataset contains drug concept identifiers (RxCUIs), scientific drug names, dosage forms, and other metadata critical for precise drug identification and mapping.
  UTS Access Portal: https://uts.nlm.nih.gov/



RxNorm Previous Releases

RxNorm files from May 2020 to the present contain data consistent with the 2020AA UMLS Metathesaurus

RxNorm_full_04072025.zip , Release Notes 04/07/2025
RxNorm_full_prescribe_04072025.zip , Release Notes 04/07/2025

RxNorm_full_03032025.zip , Release Notes 03/03/2025
RxNorm_full_prescribe_03032025.zip , Release Notes 03/03/2025

RxNorm_full_02032025.zip , Release Notes 02/03/2025
RxNorm_full_prescribe_02032025.zip , Release Notes 02/03/2025

RxNorm_full_01062025.zip , Release Notes 01/06/2025
RxNorm_full_prescribe_01062025.zip , Release Notes 01/06/2025

RxNorm_full_12022024.zip , Release Notes 12/02/2024
RxNorm_full_prescribe_12022024.zip , Release Notes 12/02/2024

RxNorm_full_11042024.zip , Release Notes 11/04/2024
RxNorm_full_prescribe_11042024.zip , Release Notes 11/04/2024

Fig. Various RxNORM datasets

Make sure to download the latest(2025) dataset from the available zip folders.

The datasets from these two sources are complementary — the Kaggle dataset gives interaction pairs, while the RXNORM dataset provides standardized drug details.

Activity 1.2: Dataset Mapping and Preparation

After acquiring both datasets, the next crucial step was to map and merge them to build a scientifically accurate and comprehensive dataset.

- **Mapping Drug Names to RxCUIs**
  Since the Kaggle dataset contained drug names mostly in common or commercial terms, we utilized the RXNORM dataset to standardize the drug information. Specifically, we extracted data from the **RXCONSO.RRF** file within the RXNORM 2025 release zip, which provides detailed mappings between drug names and their corresponding **RxCUI**(RxNorm Concept Unique Identifier). RxCUIs serve as universal, standardized identifiers that enable unambiguous identification of drugs across different databases and applications. This mapping ensured consistency and scientific accuracy in representing drug entities throughout our project.

reprocessing.py

```
# ----------------------------
# Map drug names to RxCUIs
# ----------------------------

# Filter RxNorm to preferred terms only (TTY == 'PT', column 12)
rxnorm_filtered = rxnorm_df[rxnorm_df[12] == 'PT']

# Build lookup dictionary: drug name (col 14) -> RxCUI (col 0)
rxnorm_lookup = {}
for _, row in rxnorm_filtered.iterrows():
    drug_name = str(row[14]).strip().lower()
    rxcui = str(row[0]).strip()
    if drug_name not in rxnorm_lookup:
        rxnorm_lookup[drug_name] = rxcui
```

- **Merging Interaction Data with Standardized Drug Information**
  We merged the interaction pairs from the Kaggle dataset with the mapped RxCUIs and scientific drug names from the RXNORM dataset. This process involved:
  - Matching drug names from both datasets, accounting for synonyms and spelling variations
  - Associating each drug in the interaction pair with its RxCUI
  - Combining interaction details with RxCUI-based drug records to generate a unified dataset

reprocessing.py

```
for _, row in rxnorm_filtered.iterrows():

# Function to map drug name to RxCUI
def map_to_rxcui(drug_name):
    if pd.isna(drug_name) or drug_name == '':
        return np.nan
    return rxnorm_lookup.get(drug_name, np.nan)

# Apply mapping
ddi_df['Drug 1 RxCUI'] = ddi_df['Drug 1'].apply(map_to_rxcui)
ddi_df['Drug 2 RxCUI'] = ddi_df['Drug 2'].apply(map_to_rxcui)

# Inspect mapped data
print("\nMapped DDI Sample:")
print(ddi_df[['Drug 1', 'Drug 1 RxCUI', 'Drug 2', 'Drug 2 RxCUI']].head(10))

# Save the new mapped dataset
#output_path = '/Users/telladivyareddy/Desktop/ddi_mapped_with_rxcui.csv'  # Change as needed
#ddi_df.to_csv(output_path, index=False)
#print(f"\nSaved mapped dataset to: {output_path}")
```

Map and merge and save the new custom dataset.

This combined dataset contains:

- Pairs of interacting drugs
- Standardized scientific drug names for accuracy
- Corresponding RxCUIs for both drugs in each interaction
- Interaction details to inform further analysis

This integration step ensured data consistency and scientific validity, enabling downstream processes like dosage verification and safe alternative recommendations.

The new merged dataset name ddi_mapped_with_rxcui.csv looks like the following:

Sheet 1

	A	B	C	D	E
			ddi_mapped_with_rxcui		
1	**Drug 1**	**Drug 2**	**Interaction Description**	**Drug 1 RxCUI**	**Drug 2 RxCUI**
2	**trioxsalen**	verteporfin	Trioxsalen may increase the photosensitizing activities of Verteporfin.		118886
3	**aminolevulinic acid**	verteporfin	Aminolevulinic acid may increase the photosensitizing activities of Verteporfin.		118886
4	**titanium dioxide**	verteporfin	Titanium dioxide may increase the photosensitizing activities of Verteporfin.	38323	118886
5	**tiaprofenic acid**	verteporfin	Tiaprofenic acid may increase the photosensitizing activities of Verteporfin.	38253	118886
6	**cyamemazine**	verteporfin	Cyamemazine may increase the photosensitizing activities of Verteporfin.		118886
7	**temoporfin**	verteporfin	Temoporfin may increase the photosensitizing activities of Verteporfin.	115243	118886
8	**methoxsalen**	verteporfin	Methoxsalen may increase the photosensitizing activities of Verteporfin.	6854	118886
9	**hexaminolevulinate**	verteporfin	Hexaminolevulinate may increase the photosensitizing activities of Verteporfin.		118886
10	**benzophenone**	verteporfin	Benzophenone may increase the photosensitizing activities of Verteporfin.		118886
11	**riboflavin**	verteporfin	Riboflavin may increase the photosensitizing activities of Verteporfin.	9346	118886
12	**carprofen**	verteporfin	Carprofen may increase the photosensitizing activities of Verteporfin.	20343	118886
13	**cyclophosphamide**	verteporfin	Cyclophosphamide may increase the cardiotoxic activities of Verteporfin.	3002	118886
14	**paclitaxel**	verteporfin	The risk or severity of adverse effects can be increased when Paclitaxel is combined with Verteporfin.	56946	118886
15	**docetaxel**	verteporfin	The risk or severity of adverse effects can be increased when Docetaxel is combined with Verteporfin.	72962	118886
16	**cabazitaxel**	verteporfin	The risk or severity of adverse effects can be increased when Cabazitaxel is combined with Verteporfin.	996051	118886
17	**aminolevulinic acid**	digoxin	Aminolevulinic acid may decrease the cardiotoxic activities of Digoxin.		3407
18	**temoporfin**	digoxin	Temoporfin may decrease the cardiotoxic activities of Digoxin.	115243	3407
19	**cyclophosphamide**	digoxin	Cyclophosphamide may decrease the cardiotoxic activities of Digoxin.	3002	3407

Fig. Mapped and Merged dataset

Milestone 2: NLP Model Integration for Drug Extraction and Interaction Understanding

This combined NLP pipeline—leveraging Hugging Face's domain-specific NER model and IBM Watson's advanced semantic analysis—enabled our system to precisely extract and interpret critical drug interaction information, forming a foundation for safer medical prescriptions.

Activity 2.1: Named Entity Recognition (NER) with Hugging Face Medical NER Model

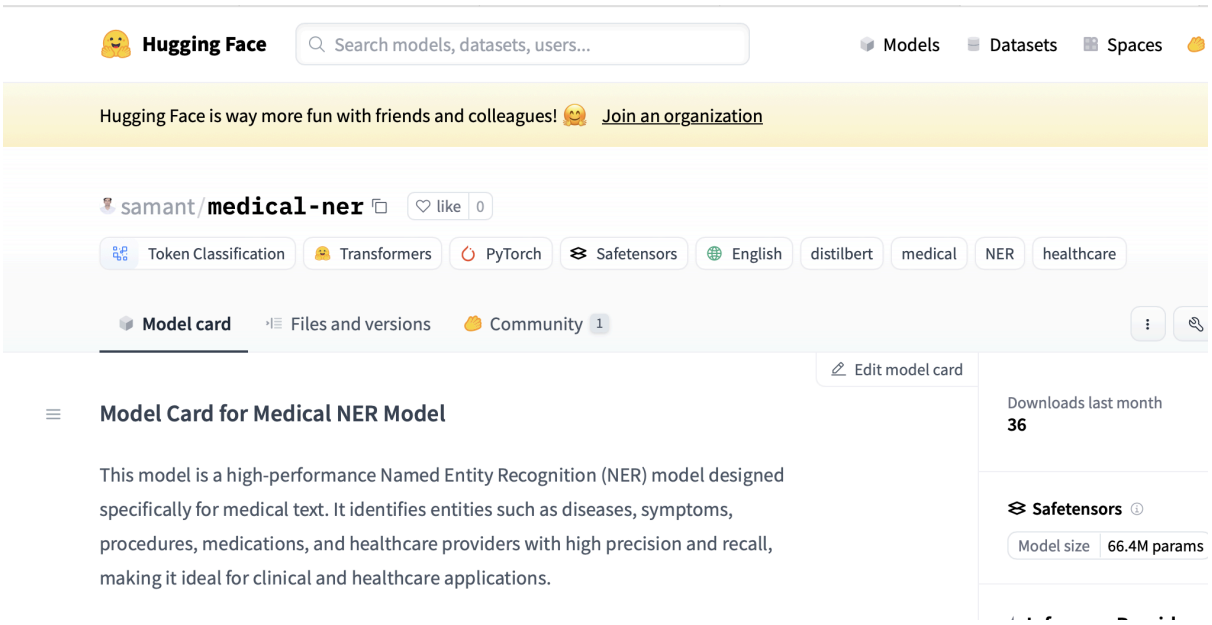
To accurately extract drug names and their dosages from medical texts, we leveraged the **samant/medical-ner** model available on Hugging Face. This model is specifically trained on medical datasets to identify entities such as drug names, dosages, and other related medical terms with high precision.

Registration and Access:

- We created a free account on Hugging Face to access their model hub and APIs.
- After account setup, we generated an API key from the user settings for authenticated requests.

Model Details:

- Model Link: samant/medical-ner
- Capabilities: Extracts drug entities along with dosage information from unstructured clinical or medical text.



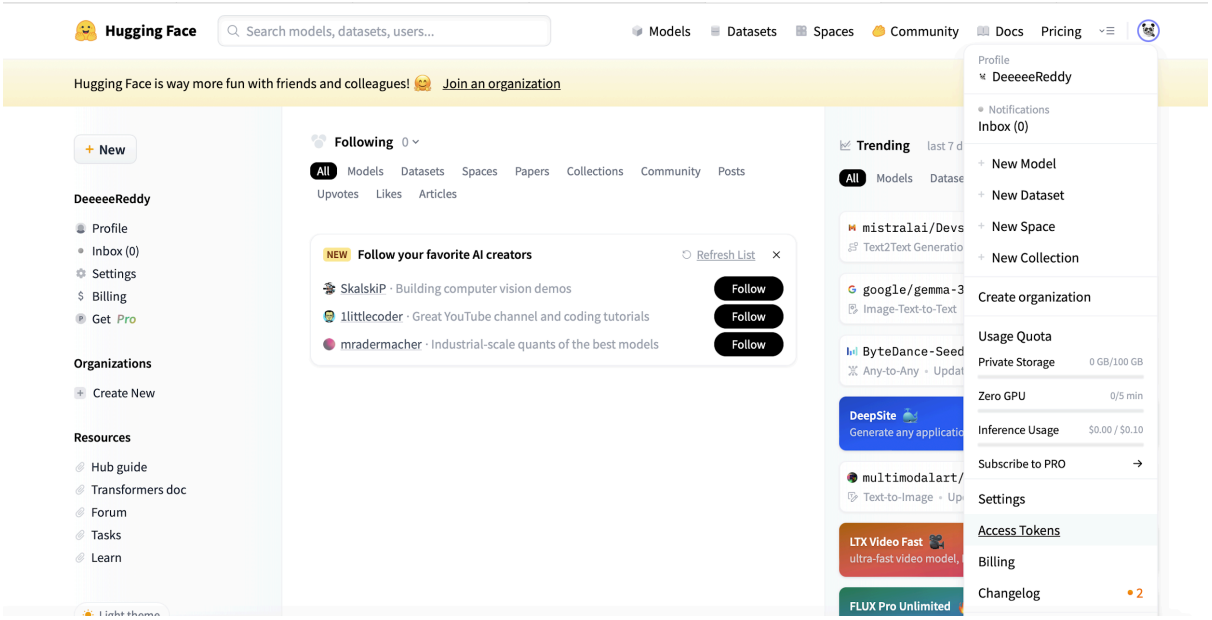
Integration:

- Using the Hugging Face Inference API, our backend sends input medical texts containing drug interaction information to the model endpoint.

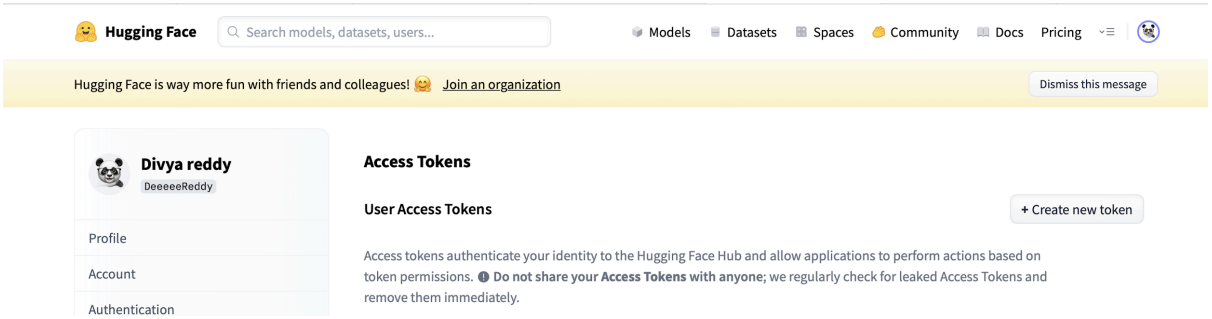
- The model returns structured annotations identifying drug names and dosages, which are then used to enrich our dataset and support further analysis.

To Generate an API Token:

- After logging in, click on your profile picture at the top right corner and select **Settings**.
- In the settings menu, navigate to **Access Tokens**.



- Click **New token**, provide a name (e.g., medical-ner-app), select the scope (choose **Read** for inference access), and click **Generate**.



- Copy and securely store the generated API token. This token will authenticate your backend requests.

Using the API Key:

- When sending requests to the Hugging Face Inference API, include your API token in the request header as:

  Authorization: Bearer YOUR_API_TOKEN

Once you get your token, we use this model to extract medicines names from the input prescription.

```
15
16  print(ddi_mapped_with_rxcui[["Drug 1 RxCUI", "Drug 2 RxCUI"]].dtypes)
17
18  def extract_medicines(text):
19      model_name = "samant/medical-ner"
20      nlp = pipeline("ner", model=model_name, framework="pt", aggregation_strategy="simple")
21      results = nlp(text)
22
23      medicines = []
24      for entity in results:
25          if entity['entity_group'].lower() == 'medicine':
26              medicines.append(entity['word'])
27
28      # Stitch together tokens like ["met", "##formin"] -> "metformin"
29      cleaned_medicines = []
30      current = ""
31      for token in medicines:
```

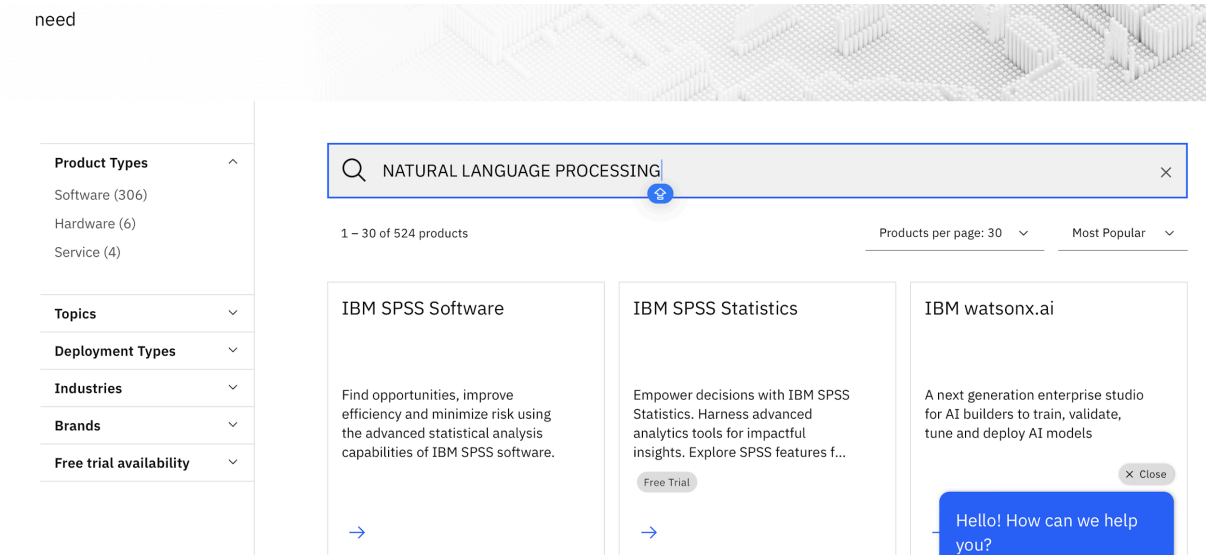
This function uses the Hugging Face samant/medical-ner model to identify and extract medicine names from medical text. It processes named entities tagged as medications, merges fragmented tokens, removes irrelevant words, and returns a clean list of drug names for further analysis or verification within the prescription validation workflow.

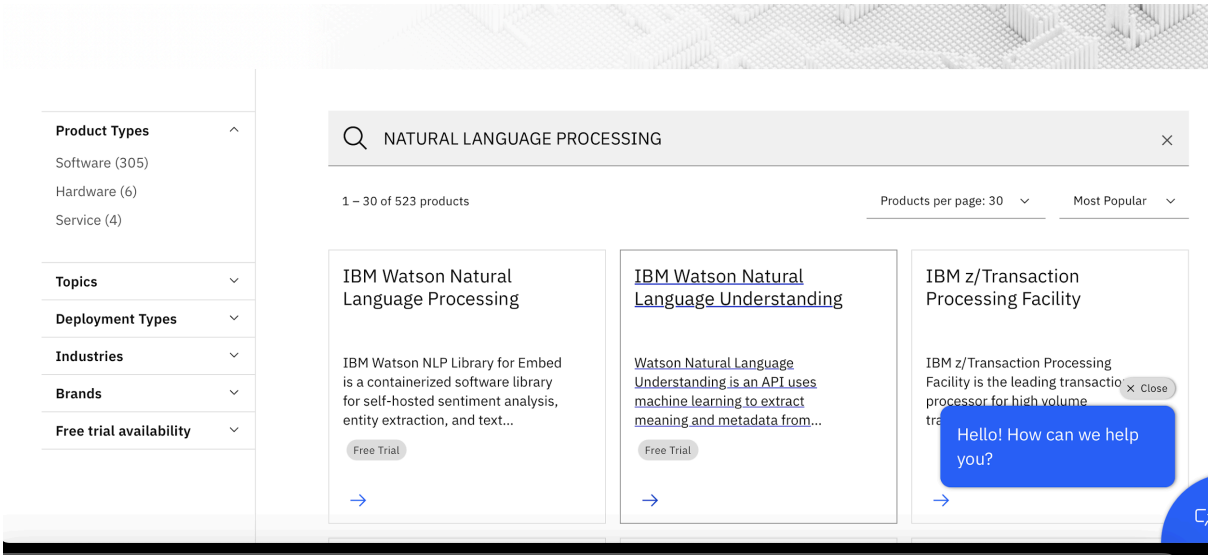
Activity 2.2: Drug Interaction Context Understanding with IBM Watson NLP

To enhance the understanding of drug interactions beyond mere extraction, we integrated **IBM Watson Natural Language Processing (NLP)** services, which provide advanced text analytics and semantic understanding.

Registration and Access:

- We registered for a free IBM Cloud account at IBM Cloud.
- After signing up, we navigated to the **IBM Watson Natural Language Understanding** service in the IBM Cloud catalog.





- We created an instance of the service, which generated an **API key** and **service URL** needed for programmatic access.

Service Details:

- IBM Watson NLP Link: IBM Watson Natural Language Understanding
- Features Used:
  - Entity recognition and classification
  - Semantic role labeling to understand drug interaction context
  - Sentiment and emotion analysis to detect risk or caution signals within interaction descriptions

```
def init_ibm_nlu():
    api_key = os.getenv("IBM_WATSON_API_KEY")
    service_url = os.getenv("IBM_WATSON_URL")

    if not api_key or not service_url:
        raise Exception("IBM Watson API key or URL not set in environment variables.")

    authenticator = IAMAuthenticator(api_key)
    nlu = NaturalLanguageUnderstandingV1(
        version='2021-08-01',
        authenticator=authenticator
    )
    nlu.set_service_url(service_url)
    return nlu

# Generate a contextual alert message from interaction description using IBM Watson NLU
def generate_alert(nlu, drug1, drug2, interaction_desc):
```

This code initializes the IBM Watson Natural Language Understanding (NLU) service and uses it to analyze drug interaction descriptions. It evaluates sentiment and emotions in the interaction text to generate personalized, context-aware alerts. These alerts help assess the severity of interactions between two drugs and guide medical decision-making.

Activity 2.3 Integration of both the models

The script begins by initializing the IBM Watson NLU client (nlu = init_ibm_nlu()). It then prompts the user to input a prescription text, from which medicines are extracted using a Hugging Face NER model (extract_medicines(text)). These extracted drug names are mapped to their corresponding **RxCUIs** using a preprocessed dataset (map_to_rxcui). Once mapped, the system checks for any known drug-drug interactions using those RxCUIs (check_interactions). If interactions are found, IBM Watson NLU is used to analyze the sentiment and emotion of the interaction descriptions. Based on the analysis, the system generates contextual alerts that warn or reassure the user depending on the interaction's severity.

```
121     nlu = init_ibm_nlu()  # Initialize IBM Watson client once
122
123     text = input("Enter prescription text:\n")
124     extracted_meds = extract_medicines(text)
125
126     if extracted_meds:
127         print("\n Extracted Medicines:")
128         for med in extracted_meds:
129             print(f" - {med}")
130
131         print("\n Mapping Medicines to RxCUIs...")
132         mapping = map_to_rxcui(extracted_meds, ddi_mapped_with_rxcui)
133
134         mapped_dict = {}
135         for m in mapping:
136             print(f"\nExtracted: {m['extracted_name']}")
137             print(f"Cleaned: {m['cleaned_name']}")
```

Milestone 3: Dosage Verification and Alternative Recommendations

This milestone focuses on validating drug dosage information and providing safer alternatives in case harmful interactions are detected. It leverages the **RxNorm API** to fetch standardized dosage details and suggest appropriate substitutions for conflicting medications.

Activity 3.1: RxNorm API Usage

To ensure dosage correctness and enrich drug information, we utilized the **RxNorm RESTful APIs** provided by the U.S. National Library of Medicine (NLM). RxNorm APIs allow us to retrieve:

- Standardized drug names
- Dosage forms (e.g., tablet, injection)
- Strengths and units (e.g., 500mg, 10ml)
- Related brand/generic names

Steps to use RxNorm API:

1. **Register for a UMLS API Key:**

- Go to: https://uts.nlm.nih.gov/
- Create an account and request access.
- Once approved, obtain your API key (used for authentication) as mentioned in Activity 1.1, Milestone 1.

2. **Use RxNorm endpoints**, such as:

- https://rxnav.nlm.nih.gov/REST/rxcui?name=DRUG_NAME
  → To get RxCUI for a drug
- https://rxnav.nlm.nih.gov/REST/rxcui/RXCUI/allrelated.json
  → To get drug variations and related ingredients
- https://rxnav.nlm.nih.gov/REST/rxcui/RXCUI/properties.json
  → To get strength and dosage information

```
102
103
104
105 def get_rxcui(drug_name):
106     url = f"https://rxnav.nlm.nih.gov/REST/rxcui.json?name={drug_name}"
107     resp = requests.get(url).json()
108     rxcui = resp.get('idGroup', {}).get('rxnormId', [None])[0]
109     return rxcui
110
111 def get_dosage_forms(rxcui):
112     # Get clinical drug forms (SCD) for dosage info
113     url = f"https://rxnav.nlm.nih.gov/REST/rxcui/{rxcui}/related.json?tty=SCD"
114     resp = requests.get(url).json()
115     related = resp.get('relatedGroup', {}).get('conceptGroup', [])
116
117     dosage_info = []
118     for group in related:
```

Relevant Functions:

1. get_rxcui(drug_name)

- Calls: https://rxnav.nlm.nih.gov/REST/rxcui.json?name=<drug_name>
- Output: RxNorm Concept Unique Identifier (**RxCUI**)
- Used to uniquely identify the drug.

2. get_dosage_forms(rxcui)

- Calls: https://rxnav.nlm.nih.gov/REST/rxcui/<rxcui>/related.json?tty=SCD
- Fetches **Structured Clinical Drug (SCD)** info.
- Output: List of valid dosage formulations (e.g., Paracetamol 500 MG Oral Tablet)

Activity 3.2: Alternative Safe Drug Suggestions

This part uses RxNorm to find alternative branded drugs with the same active ingredient.

Relevant Function:

get_alternatives(rxcui)

- Step 1: Find **ingredient-level RxCUI** using:
  - https://rxnav.nlm.nih.gov/REST/rxcui/<rxcui>/related.json?tty=IN
- Step 2: Use ingredient RxCUI to get branded drugs:
  - https://rxnav.nlm.nih.gov/REST/rxcui/<ingredient_rxcui>/related.json?tty=SBD
- Output: List of alternative drug names (e.g., brands like Crocin, Tylenol, etc.)

```
122     return dosage_info
123
124 def get_alternatives(rxcui):
125     # Get ingredient RxCUI (IN) for the drug
126     url = f"https://rxnav.nlm.nih.gov/REST/rxcui/{rxcui}/related.json?tty=IN"
127     resp = requests.get(url).json()
128     ingredients = resp.get('relatedGroup', {}).get('conceptGroup', [])
129     if not ingredients:
130         return []
131     ingredient_rxcui = ingredients[0].get('conceptProperties', [{}])[0].get('rxcui')
132     if not ingredient_rxcui:
133         return []
134
135     # Get branded drugs (SBD) for the same ingredient as alternatives
136     url2 = f"https://rxnav.nlm.nih.gov/REST/rxcui/{ingredient_rxcui}/related.json?tty=SBD"
137     resp2 = requests.get(url2).json()
138     brands = resp2.get('relatedGroup', {}).get('conceptGroup', [])
```

This Python script extracts medicine names, dosages, and forms from user input using a medical NER model and regex fallback. It cleans drug names, fetches RxCUI identifiers from RxNorm API, retrieves correct dosage forms, and suggests branded alternatives. It also allows user interaction through the console for real-time processing based on entered medication details and patient age.

Milestone 4: Backend and Frontend Development

This milestone establishes the complete application pipeline from backend logic to frontend accessibility.

Activity 4.1: FastAPI **Backend**

The backend is built using FastAPI, a high-performance web framework. It exposes two key API endpoints:

- **/check_interactions**
  Takes a prescription text, uses IBM Watson NLU for natural language understanding, extracts drug names, maps them to standardized RxCUIs, and checks for potential drug-drug interactions using a preloaded dataset (ddi_mapped_with_rxcui.csv). It then returns detailed interaction alerts including natural language explanations powered by IBM's NLU.

```python
@app.post("/check_interactions", response_model=List[InteractionResponse])
def check_interactions_api(request: InteractionRequest):
    if not nlu_client:
        raise HTTPException(status_code=500, detail="IBM NLU client not initialized")

    # Extract medicines
    meds = extract_medicines(request.prescription_text)
    if not meds:
        return []

    # Map medicines to RxCUI
    mapping = map_to_rxcui(meds, ddi_df)
    mapped_dict = {}
    for m in mapping:
        if m['matched_name']:
            mapped_dict[m['extracted_name']] = m['rxcuis']

```

- **/check_dosage**
  Accepts a prescription text and patient's age. It extracts medicine names and dosages, then compares them with recommended values. If incorrect, the system suggests adjustments and safe alternatives.

```python

@app.post("/check_dosage", response_model=List[DosageResponseItem])
def check_dosage_api(request: DosageRequest):
    try:
        results = dosage_checker.process_user_input(request.prescription_text, age=request.age)
        return results
    except ValueError as e:
        raise HTTPException(status_code=400, detail=str(e))
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Internal error: {e}")

```

Both endpoints handle error cases robustly using FastAPI's HTTP exception system, and data models are defined using pydantic.BaseModel for validation and clarity.

To run your FastAPI app, use the following command in your terminal (from the directory where your main.py file is located):

uvicorn api:app –reload

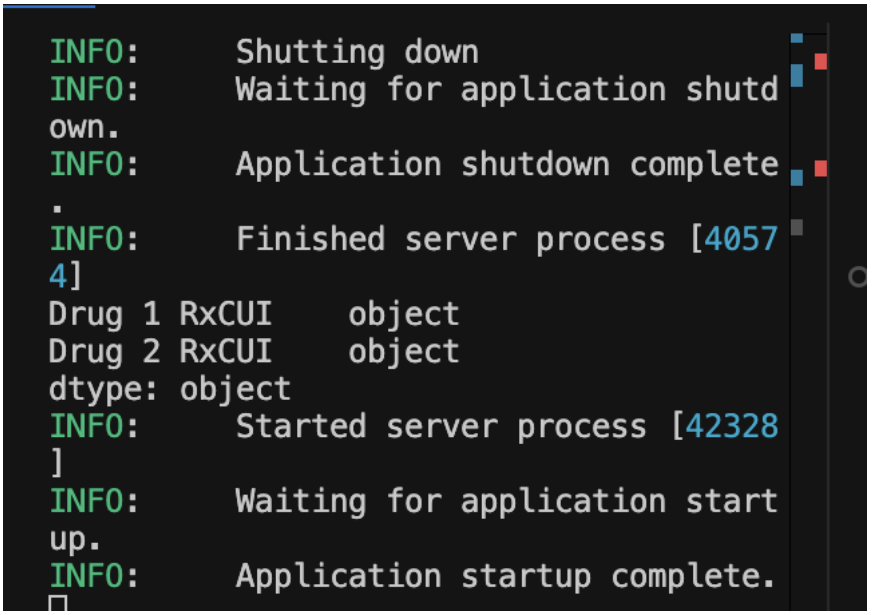


Fig. Make sure you see this message to ensure your fastapi routes are working and active.

After running, open your browser and go to:

- Swagger docs: http://127.0.0.1:8000/docs
- Redoc: http://127.0.0.1:8000/redoc

Activity 4.2: Streamlit Frontend

This activity involves building a simple and interactive user interface (UI) using Streamlit, a Python framework designed to create data apps and dashboards quickly. The frontend allows users to:

- Enter medication prescription text.
- Submit the input to the backend FastAPI endpoints you developed (for interaction and dosage checking).
- View the results (such as detected drug interactions or dosage recommendations) clearly and intuitively on the web page.

The frontend takes care of making API calls to your FastAPI backend, handles user input validation, and presents the processed output in tables, alerts, or lists. This enhances usability and makes the system accessible to non-technical users such as doctors, pharmacists, or patients.

1. Make sure your FastAPI backend is running on http://127.0.0.1:8000 (or update URLs accordingly if different).
2. Run the Streamlit app using the terminal: streamlit run frontend.py
3. Access the frontend in your browser at: http://localhost:8501

4. Use the UI to enter medication text, click buttons, and see results fetched from your FastAPI backend.

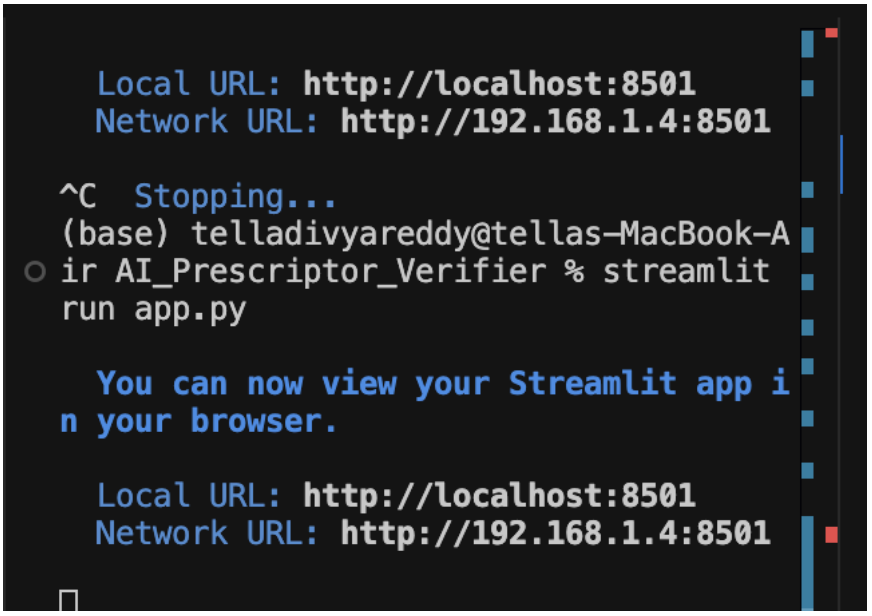
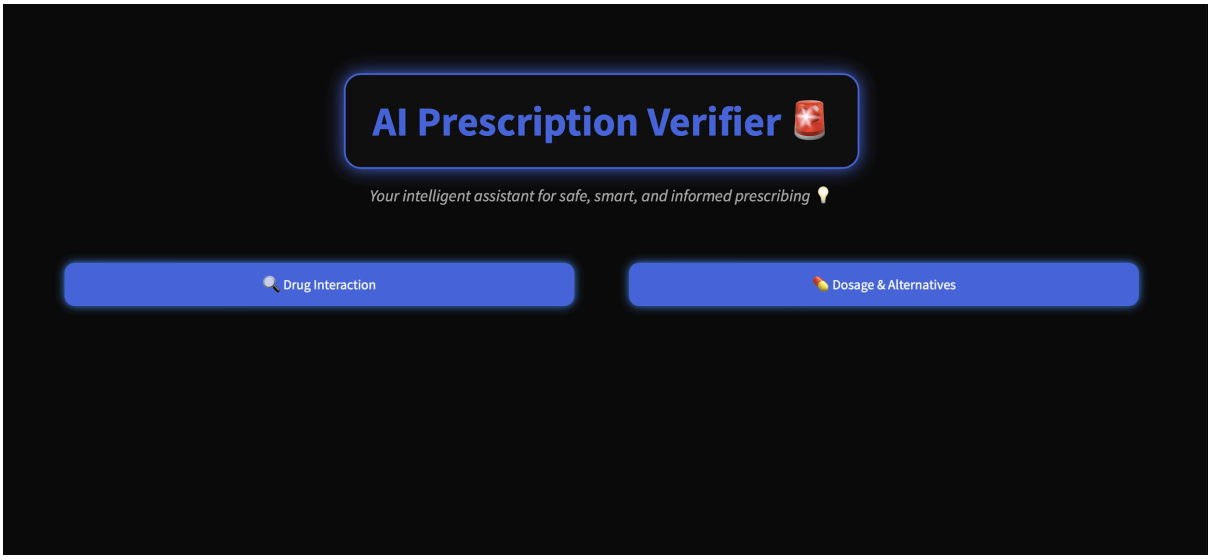


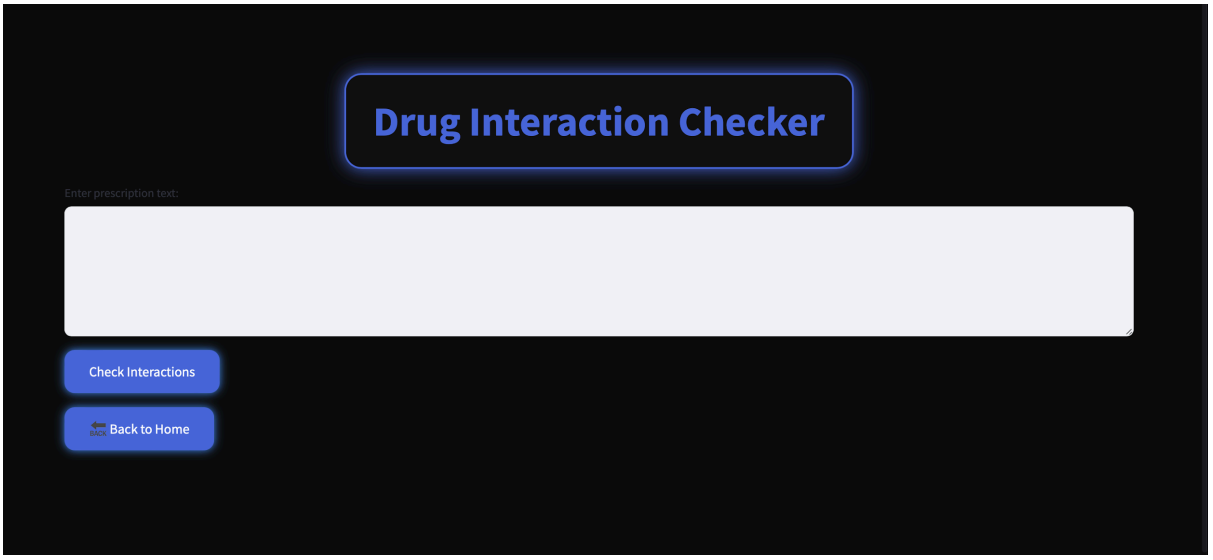
Fig. Showing your Frontend is ready to access.

Functional Testing:

Main Home Page:



Drug Interaction Checker:



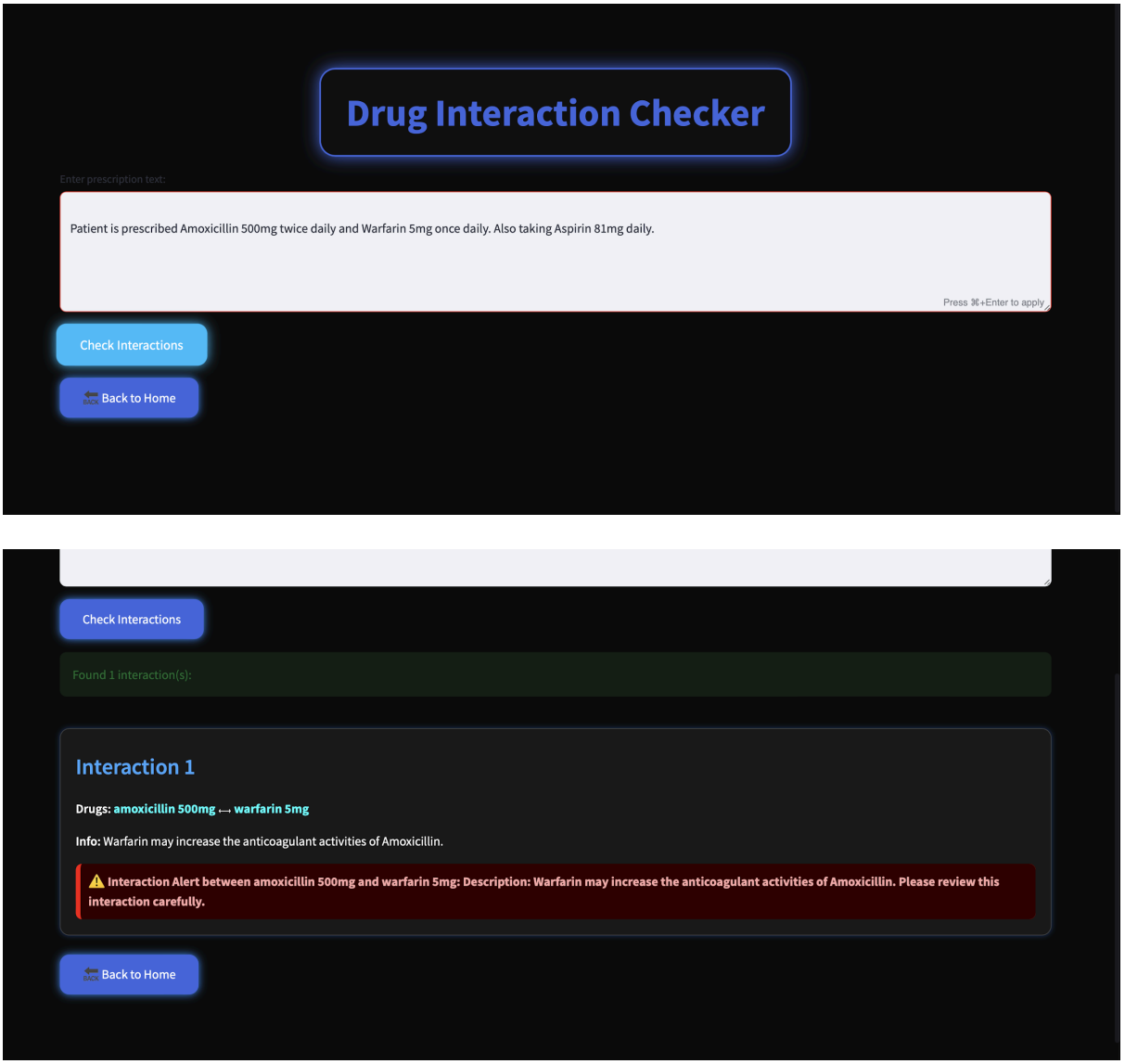
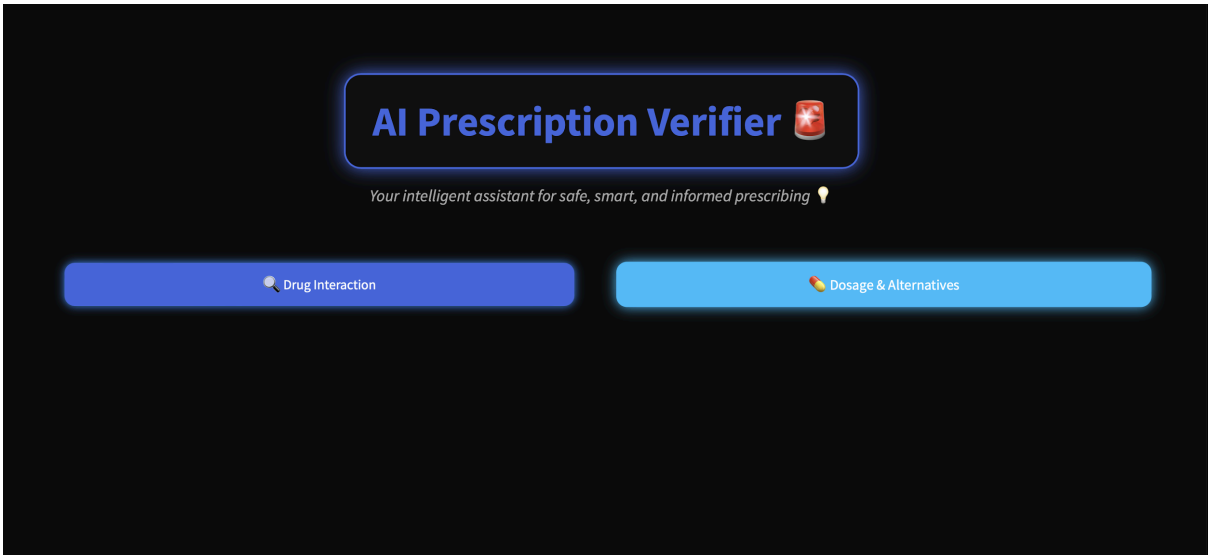
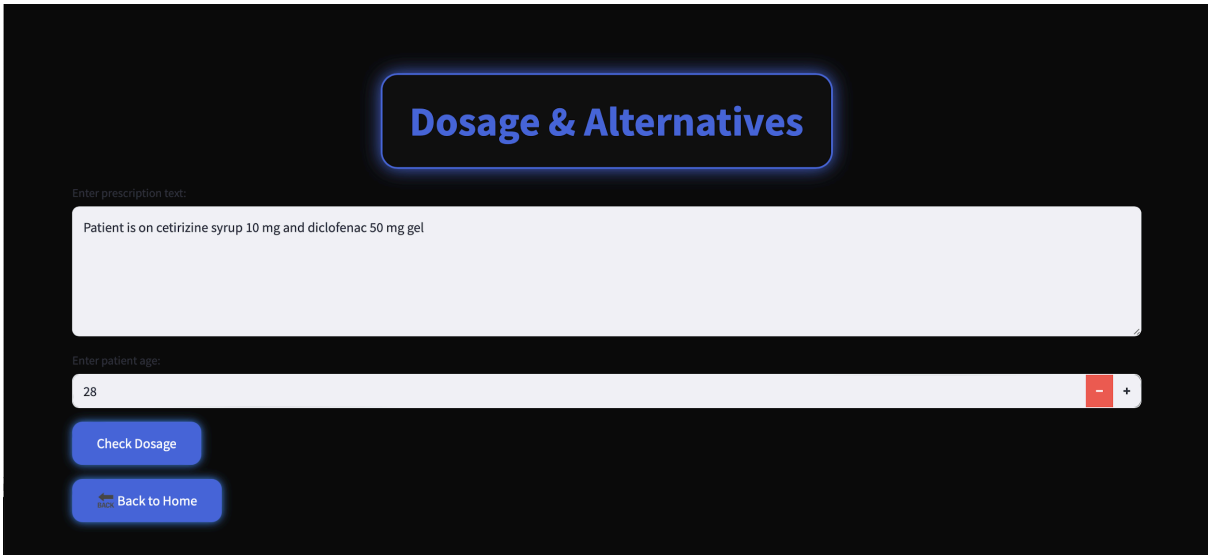


Fig. Successfully showing interactions between inputted prescribed drugs.

Drug Dosage & Alternatives:





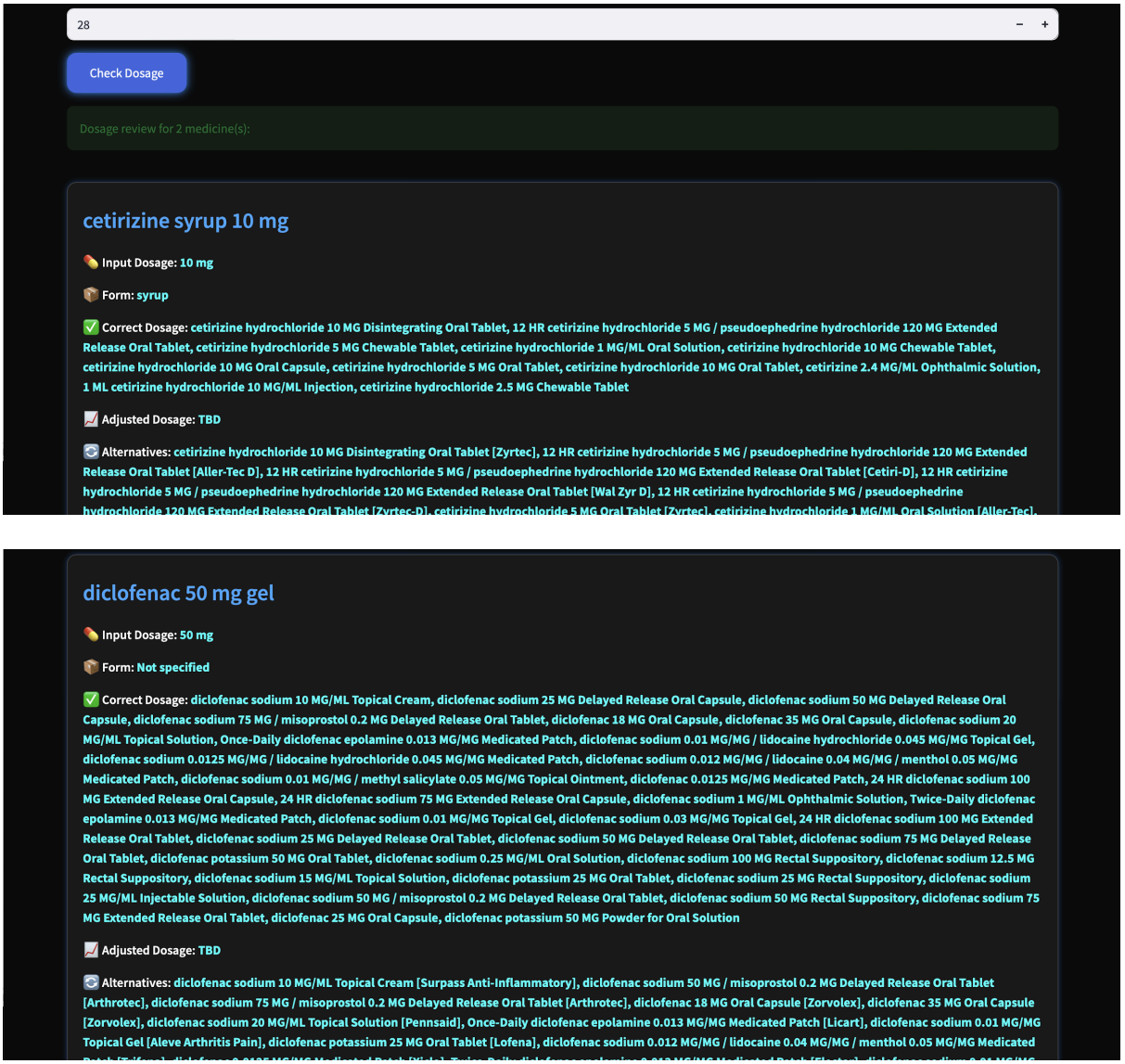


Fig. Successfully showing the dosage and its form and alternatives to the drugs based on age.

Conclusion

This project successfully developed an end-to-end system that empowers users to input medication prescriptions and receive detailed insights on potential drug interactions and dosage recommendations. By integrating advanced NLP techniques and healthcare datasets, the system automates critical checks traditionally performed by pharmacists, enhancing patient safety and medication management.

The core achievement lies in designing and implementing a robust backend API using FastAPI, which efficiently handles medicine extraction, maps medicines to RxCUI codes, and detects possible drug-drug interactions from a curated dataset. Additionally, the system incorporates IBM Watson Natural Language Understanding to generate meaningful alerts explaining interaction severity and consequences, providing valuable context to users. Alongside, a dosage checking module was integrated to analyze prescribed dosages relative to patient age and suggest optimal adjustments or alternatives. Complementing the backend, a user-friendly frontend was built with Streamlit, enabling intuitive input and clear presentation of interaction and dosage results. This combination of frontend and backend creates a seamless experience that can assist healthcare professionals and patients alike.

Throughout the development, several challenges were encountered. First, accurately extracting drug names from free-text prescriptions proved complex due to variability in medical terminology, spelling errors, and shorthand notations. To address this, multiple text-cleaning and mapping strategies were employed to reliably link extracted terms to standard RxCUI codes. Second, managing the large drug interaction dataset required efficient preprocessing and lookup methods to ensure fast response times within API calls. Third, integrating IBM Watson's NLU service introduced dependency and latency considerations; initializing and handling errors robustly was crucial to maintain system stability. Additionally, designing a simple yet functional frontend interface in Streamlit demanded balancing usability with technical constraints, especially in displaying multi-item interaction results clearly. Finally, coordinating data flow and response formatting between modules required careful planning to maintain modularity and ease

This project provided valuable hands-on experience in full-stack AI system development, combining data science, software engineering, and user experience design. Key learnings included mastering FastAPI for building scalable RESTful services and managing asynchronous tasks and exceptions. The importance of robust data preprocessing and normalization was highlighted when working with real-world medical datasets. The use of NLP APIs for domain-specific alert generation showcased the power and challenges of integrating third-party AI services. Streamlit proved to be an effective tool for rapidly prototyping interactive applications with minimal frontend expertise. Moreover, this project reinforced best practices in API design, modular programming, and documentation for collaborative development. The iterative debugging and optimization process deepened understanding of performance trade-offs in data-intensive applications.

There are several promising directions to extend this system. Incorporating additional clinical parameters such as patient weight, renal function, or allergies could refine dosage recommendations and interaction risk assessments, enabling personalized medication management. Expanding the drug interaction dataset with real-time updates from regulatory sources would improve reliability and comprehensiveness. Integrating with electronic health record (EHR) systems could facilitate automatic prescription input and clinical decision support for healthcare providers. Advanced NLP models trained specifically on medical texts could enhance drug extraction accuracy and enable explanation generation beyond alert texts, such as detailed patient-friendly summaries. A mobile app version with offline capabilities could increase accessibility in resource-limited settings. Finally, adding multi-language support and voice input could broaden usability for diverse populations. These enhancements could

transform the system into a versatile digital health assistant, significantly contributing to medication safety worldwide.