❖ Introduction

The QA system leverages Natural Language Processing (NLP) and Neo4j Graph Database to provide real-time, natural language query handling for movie datasets. The project aims to address movie-related queries, extract insights, and deliver an enhanced user experience.

❖ Platform/System Setup

- **Environment:** Python 3.12.3 with essential libraries such as Flask, pandas, pymongo, Neo4j, spacy, tqdm, and others.
- **Python IDE:** Pycharm Community version 2023.3.3
- **Neo4j Configuration:** Requires Neo4j running locally with the credential's user="neo4j", password="bigdata612".

  Made few config changes:
  - dbms.memory.heap.initial_size=6G
  - dbms.memory.heap.max_size=6G
  - dbms.memory.pagecache.size=8G
  - dbms.windows_service_name=neo4j-relate-dbms-a4ee4310-4e49-49e0-9082-926af4973254

```
# Neo4j Configuration
NEO4J_URI = "bolt://localhost:7687"
NEO4J_USER = "neo4j"
NEO4J_PASSWORD = "bigdata612"
```

- **MongoDB Configuration:** Local MongoDB instance to store intermediate datasets.

```
# MongoDB Setup
client = MongoClient("mongodb://localhost:27017/")
db = client['MoviesDB']
movies_collection = db['movies']
triples_collection = db['triples']
```

- **CoreNLP Server:** Stanza CoreNLP server for natural language processing tasks.
- Required Libraries: pandas, pymongo, psutil, neo4j, Flask, request, render_template.

❖ **Setup Procedures**

- Install the required packages: flask pandas pymongo spacy tqdm neo4j stanza
- Set up and run Neo4j locally, ensuring it listens on bolt://localhost:7687.
- Download the Stanza CoreNLP package and specify the path in NLP_triple_extract.py.

```
import stanza
stanza.install_corenlp()
```

```
# CoreNLP Path
CORENLP_PATH = "C:/Users/nitee/stanza_corenlp/*"
```

- Load the spaCy model for English: en_core_web_trf

```
# Load the transformer-based English model
nlp = spacy.load('en_core_web_trf')
```

❖ **Code Analysis**

Each file serves a specific purpose in the project pipeline:

---

1. **Dataset Cleaning (Dataset_cleaning.py)**

- **Purpose:** Cleans and processes raw movie data.
- **Steps:**
  - Handle missing values in columns like budget, revenue, runtime, etc.

```python
def handle_missing_values(df):
    df = df[df['title'].notna()]
    df = df[df['title'].str.strip() != ""]
    df['budget'] = df['budget'].mask(df['budget'] == 0, pd.NA)
    df['revenue'] = df['revenue'].mask(df['revenue'] == 0, pd.NA)
    df['runtime'] = df['runtime'].mask(df['runtime'] <= 0, pd.NA)
    textual_columns_to_fill = ['genres', 'director', 'imdb_id', 'spoken_languages',
                               'cast', 'production_companies', 'production_countries', 'writers',
                               'director_of_photography', 'producers', 'music_composer']
    for col in textual_columns_to_fill:
        if col in df.columns:
            df[col] = df[col].fillna("unknown")
    if 'overview' in df.columns:
        df['overview'] = df['overview'].str.strip().replace(["", "n/a", "none"], "unknown")
    return df
```

  - Standardize text by converting to lowercase and stripping whitespace.

```python
1 usage   new *
def standardize_text_columns(df, columns):
    for col in tqdm(columns, desc="Standardizing text columns"):
        df[col] = df[col].astype(str).str.strip().str.lower()
    return df
```

  - Normalize multi-value columns (e.g., genres, cast) by splitting strings into lists.

```python
1 usage   new *
def normalize_columns(df):
    columns_to_split = [
        'genres', 'spoken_languages', 'production_companies', 'production_countries',
        'cast', 'director', 'director_of_photography', 'writers', 'producers', 'music_composer'
    ]
    for col in tqdm(columns_to_split, desc="Normalizing columns"):
        if col in df.columns:
            df[col] = df[col].str.strip().str.split(",")
    return df
```

  - Remove duplicate records based on title and release_date.

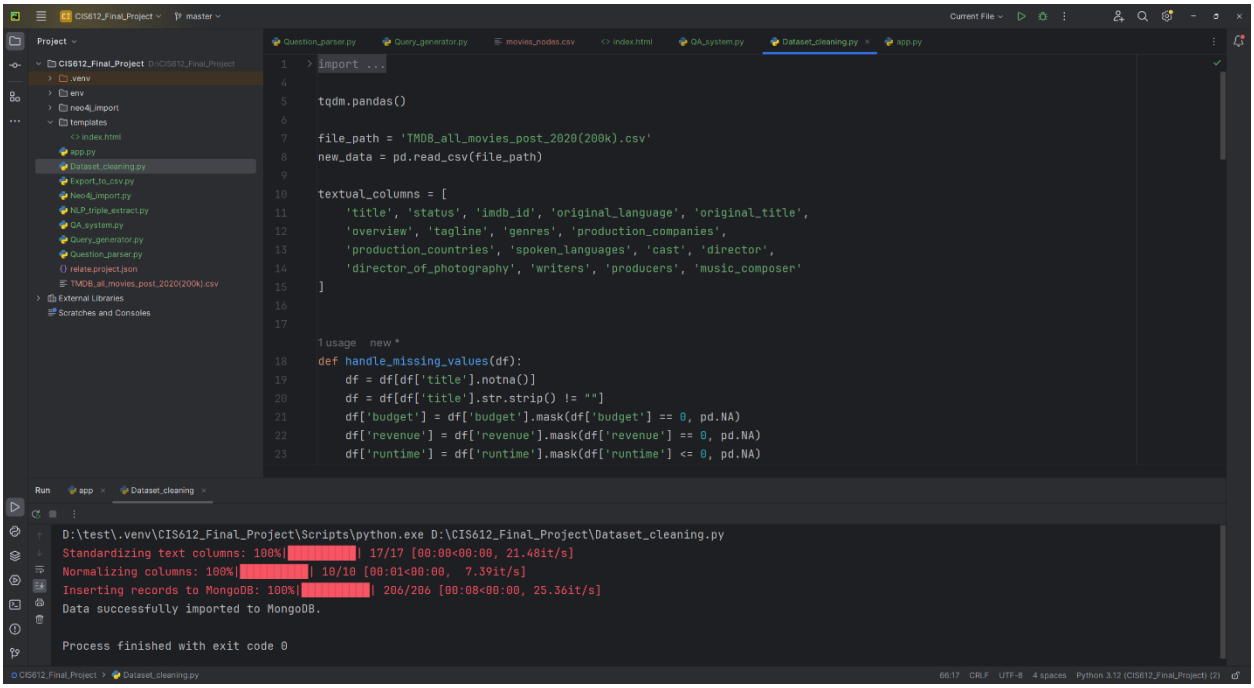
```
3 usages (2 dynamic)    new *
def remove_duplicates(df):
    df = df.drop_duplicates(subset=['title', 'release_date'], keep='first')
    return df
```

```
cleaned_data = (
    new_data
    .pipe(lambda df: tqdm.pandas(desc="Handling missing values") or handle_missing_values(df))
    .pipe(lambda df: tqdm.pandas(desc="Standardizing text") or standardize_text_columns(df, textual_columns))
    .pipe(lambda df: tqdm.pandas(desc="Correcting inconsistent data") or correct_inconsistent_data(df))
    .pipe(lambda df: tqdm.pandas(desc="Normalizing columns") or normalize_columns(df))
    .pipe(lambda df: tqdm.pandas(desc="Removing duplicates") or remove_duplicates(df))
)
```

- Save the cleaned dataset to MongoDB in the MoviesDB.movies collection.

```
batch_size = 1000
for i in tqdm(range(0, len(records), batch_size), desc="Inserting records to MongoDB"):
    batch = records[i:i+batch_size]
    collection.insert_many(batch)
```

- **Intermediate Output:** A clean and structured dataset in MongoDB.



---

2. Triple Extraction (NLP_triple_extract.py)

- **Purpose:** Extracts subject-predicate-object triples from movie overviews using CoreNLP.
- **Steps:**

- Resolves coreferences in text to improve triple extraction accuracy.

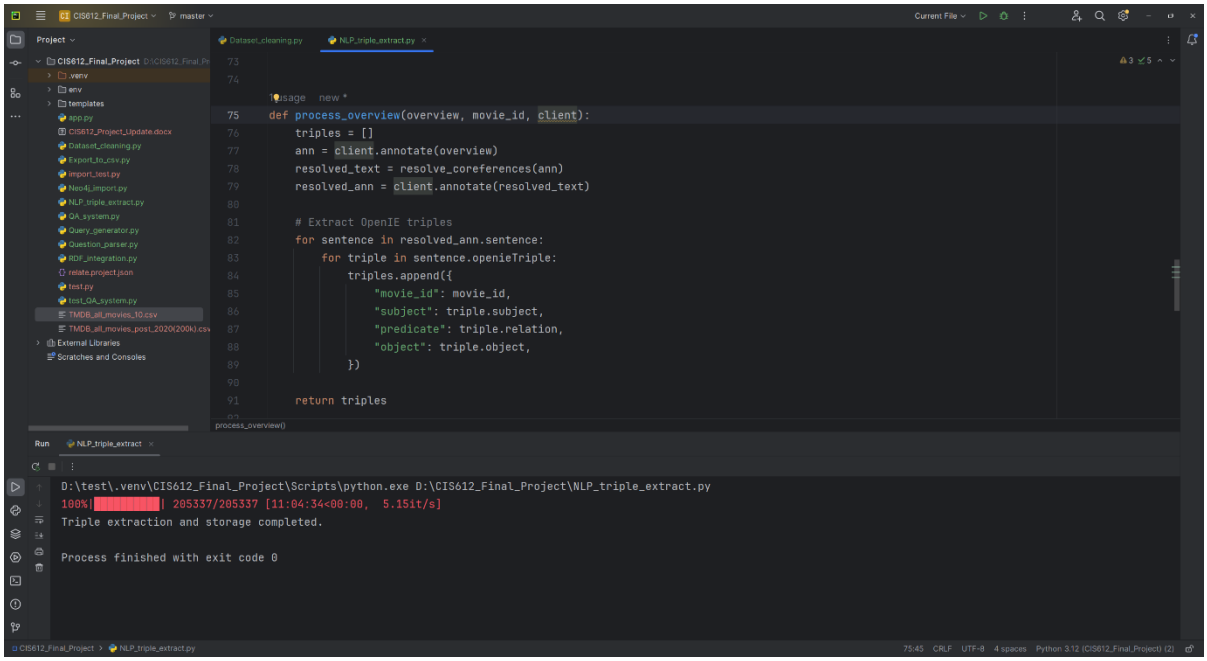
```python
def resolve_coreferences(annotated):
    if not annotated.corefChain:
        return annotated.text

    replacements = {}
    for chain in annotated.corefChain:
        representative = chain.mention[0]  # Representative mention
        rep_text = " ".join(
            token.word
            for token in annotated.sentence[representative.sentenceIndex].token[
                representative.beginIndex:representative.endIndex
            ]
        )

        for mention in chain.mention[1:]:
            sent_idx = mention.sentenceIndex
            token_range = range(mention.beginIndex, mention.endIndex)
            mention_text = " ".join(
                annotated.sentence[sent_idx].token[i].word for i in token_range
            )
            replacements[mention_text] = rep_text

    resolved_text = annotated.text
    for mention_text, rep_text in replacements.items():
        resolved_text = re.sub(r"\b" + re.escape(mention_text) + r"\b", rep_text, resolved_text)

    return resolved_text
```

- Annotates text using CoreNLP and extracts OpenIE triples.
- Saves triples to MongoDB in the MoviesDB.triples collection.

```python
        client.start()
        all_triples = []
        for _, row in tqdm(data.iterrows(), total=len(data)):
            if pd.notna(row.get("overview")):
                extracted_triples = process_overview(row["overview"], row["id"], client)
                all_triples.extend(extracted_triples)

        # Save to MongoDB
        triples_collection = db["triples"]
        triples_collection.drop()
        triples_collection.insert_many(all_triples)

        print("Triple extraction and storage completed.")
```

- **Issues Encountered:** Port conflicts with CoreNLP, resolved by dynamically assigning a free port.

- **Intermediate Output:** Triples extracted and stored in MongoDB.



---

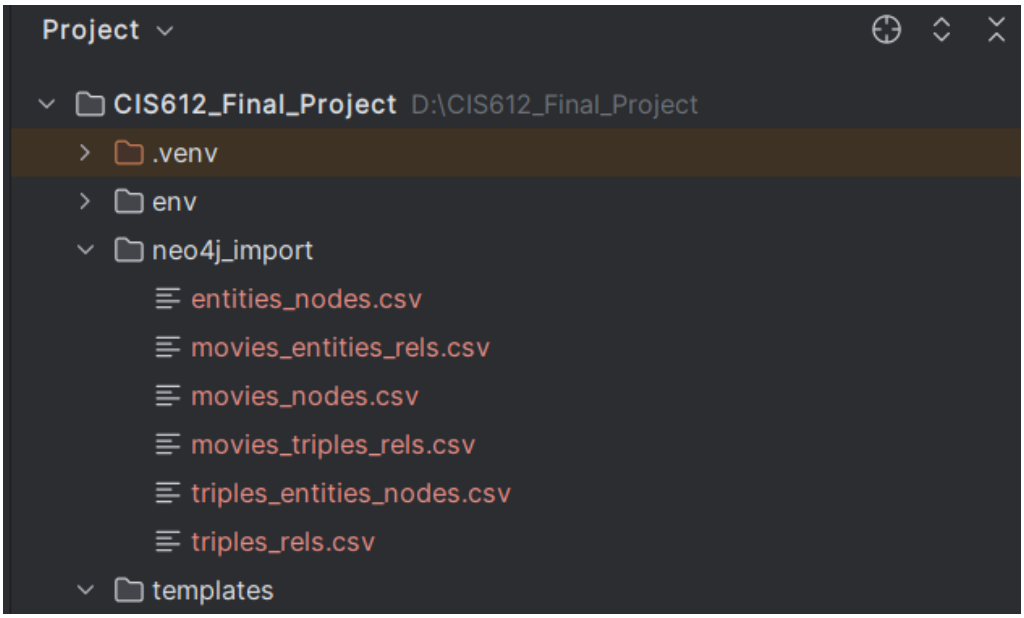
3. Export to CSV for Neo4j (Export_to_csv.py)

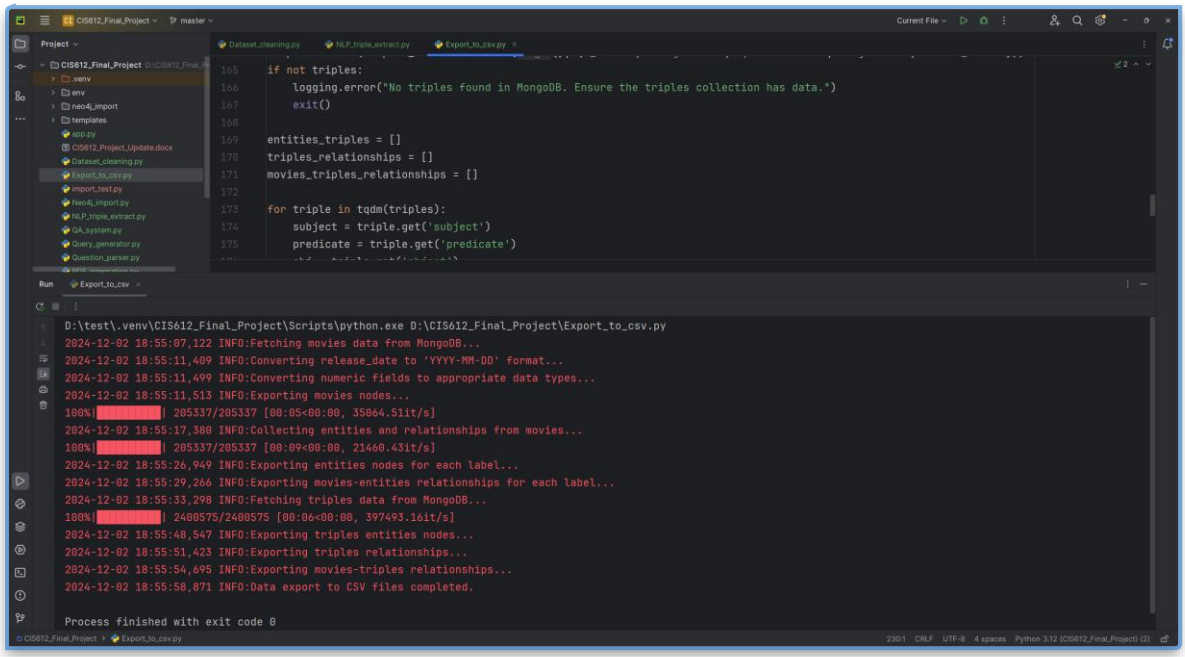
- **Purpose: Process** cleaned data to CSV for Neo4j import.
- **Steps:**
  - Fetch movies and relationships from MongoDB.

```
# Fetch movies data
logging.info("Fetching movies data from MongoDB...")
movies = list(movies_collection.find())
if not movies:
    logging.error("No movies found in MongoDB. Ensure the movies collection has data.")
    exit()
movies_df = pd.DataFrame(movies)
```

  - Normalize names and generate unique entity IDs.
  - Create node and relationship files for Neo4j import:
    - Entities nodes
    - Movies entities relations
    - Movies nodes
    - Movies triples relations

    - Triples entities nodes
    - Triple relations
- **Issues Encountered:** Data type mismatches in numeric fields, resolved by coercion and filling NaN values.
- **Intermediate Output:** CSV files stored in the neo4j_import directory.



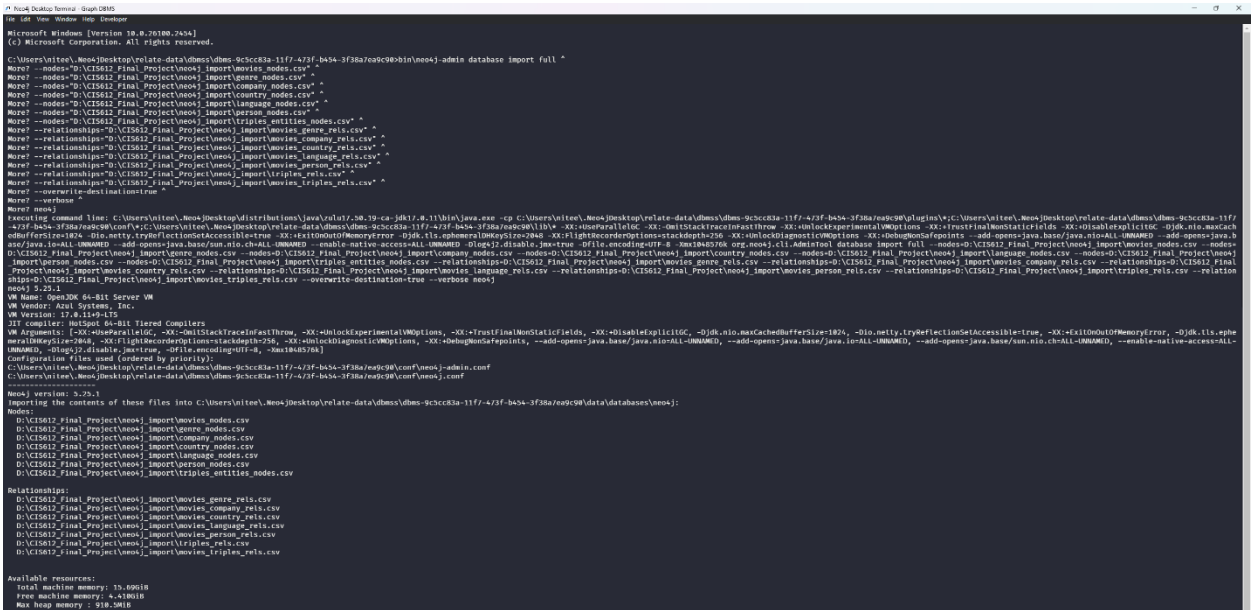


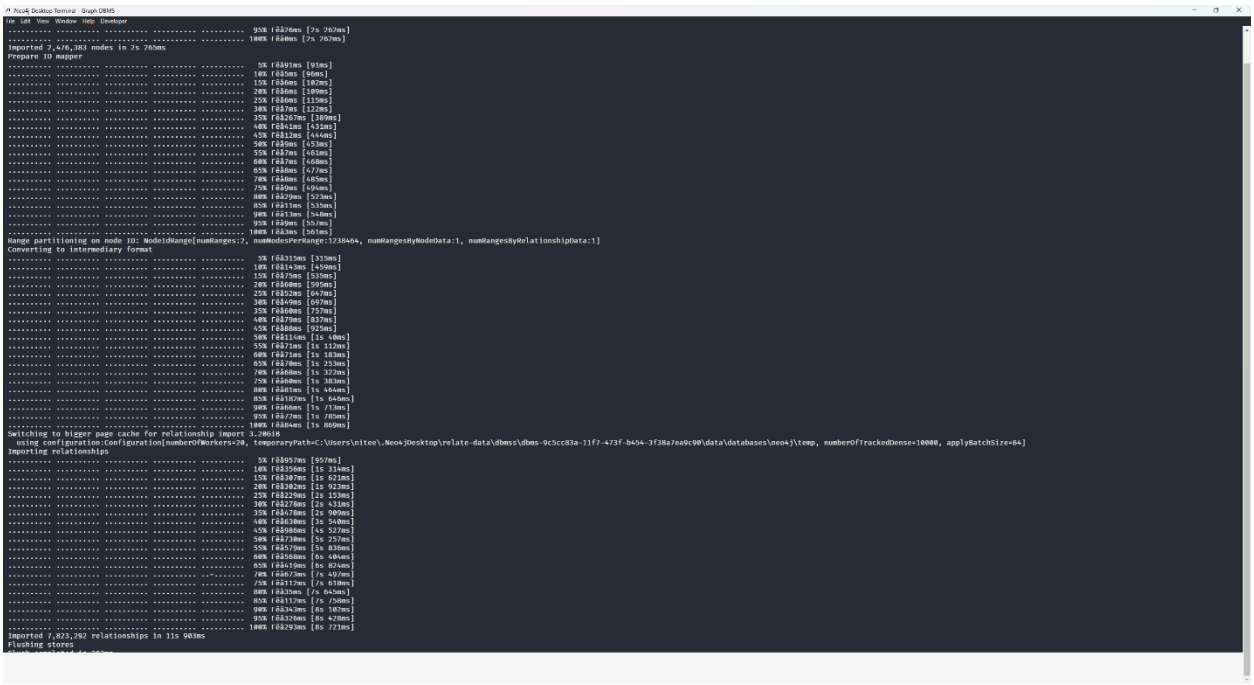
- **CSV files can be used to perform bulk neo4j-admin import:**

Command:

```
bin\neo4j-admin database import full ^
--nodes=Movie="D:\CIS612_Final_Project\neo4j_import\movies_nodes.csv" ^
--nodes="D:\CIS612_Final_Project\neo4j_import\entities_nodes.csv" ^
--nodes="D:\CIS612_Final_Project\neo4j_import\triples_entities_nodes.csv" ^
--relationships="D:\CIS612_Final_Project\neo4j_import\movies_entities_rels.csv" ^
--relationships="D:\CIS612_Final_Project\neo4j_import\triples_rels.csv" ^
--relationships="D:\CIS612_Final_Project\neo4j_import\movies_triples_rels.csv" ^
--overwrite-destination=true ^
--verbose ^
neo4j
```

Output:





---

4. Neo4j Import (Neo4j_import.py -for small data set)

- **Purpose:** Imports nodes and relationships from MongoDB into Neo4j.
- **Steps:**
    - Creates movie nodes with properties like title, budget, and runtime.

```python
def import_movies_and_relationships(movies, neo4j_manager):
    print("Importing movies and relationships into Neo4j...")

    for movie in movies:
        # Movie Node
        movie_id = movie.get("id")
        title = movie.get("title")
        release_date = movie.get("release_date")
        budget = movie.get("budget")
        runtime = movie.get("runtime")
        vote_average = movie.get("vote_average")
        status = movie.get("status")
        revenue = movie.get("revenue")
        original_title = movie.get("original_title")
        imdb_rating = movie.get("imdb_rating")
        imdb_votes = movie.get("imdb_votes")
        original_language = movie.get("original_language")
        popularity = movie.get("popularity")
```

- Establishes relationships such as DIRECTED_BY, BELONGS_TO_GENRE, etc.

```
        # Process Relationships
        relationships = [
            ("genres", "Genre", "BELONGS_TO_GENRE"),
            ("spoken_languages", "Language", "SPOKEN_IN"),
            ("production_companies", "Company", "PRODUCTION_COMPANY"),
            ("production_countries", "Country", "PRODUCED_IN"),
            ("cast", "Person", "ACTED_IN"),
            ("director", "Person", "DIRECTED_BY"),
            ("writers", "Person", "WRITTEN_BY"),
            ("producers", "Person", "PRODUCED_BY"),
            ("music_composer", "Person", "COMPOSED_BY"),
            ("director_of_photography", "Person", "DOP_BY")
        ]

        for field, label, rel_type in relationships:
            entities = movie.get(field, [])
            if not isinstance(entities, list):
                entities = [entities]

            for entity in entities:
                if not entity:
                    continue

                query_entity = f"""
                MERGE (n:{label} {{name: $entity_name}})
                MERGE (m:Movie {{id: $movie_id}})
                MERGE (m)-[:{rel_type}]->(n)
                """
                neo4j_manager.execute_query(query_entity, parameters={
                    "entity_name": entity,
                    "movie_id": movie_id
                })
```

- Imports extracted triples using APOC procedures.

```python
# Import Triples into Neo4j using APOC
1 usage  new *
def import_triples(triples, neo4j_manager):
    print("Importing triples into Neo4j...")
    query = """
    CALL apoc.periodic.iterate(
      'UNWIND $triples AS triple RETURN triple',
      'MERGE (s:Entity {name: triple.subject})
       MERGE (o:Entity {name: triple.object})
       MERGE (m:Movie {id: triple.movie_id})
       MERGE (s)-[r:RELATION {type: triple.predicate}]->(o)
       MERGE (m)-[:HAS_SUBJECT]->(s)
       MERGE (m)-[:HAS_OBJECT]->(o)',
      {batchSize: 1000, parallel: true, params: {triples: $triples}}
    )
    """
    neo4j_manager.execute_query(query, parameters={"triples": triples})
    print("Triples successfully imported into Neo4j.")


# Execute the import process
try:
    import_movies_and_relationships(movies, neo4j_manager)
    import_triples(triples, neo4j_manager)
finally:
    neo4j_manager.close()
```

- **Issues Encountered**: Missing data in triples, resolved by skipping incomplete records.

---

5. **Question Parsing (Question_parser.py)**

- **Purpose:** Determines user intent and extracts relevant entities from natural language questions.
- **Key Features:**
  - Regex-based patterns to identify intents like FindDirector, FindMoviesByGenre.

```python
# Define intent patterns
INTENT_PATTERNS = [
    {
        'intent': 'FindDirector',
        'pattern': r'who\s+(directed|is the director of)\s+(?P<Movie>.+)',
        'entities': ['Movie']
    },
    {
        'intent': 'FindActors',
        'pattern': r'who\s+(acted in|starred in|are the actors in)\s+(?P<Movie>.+)',
        'entities': ['Movie']
    },
    {
        'intent': 'FindMoviesByGenre',
        'pattern': r'(which|what)\s+movies\s+(are|belong to|fall under|classified as)\s+(?P<Genre>.+)\s+genre',
        'entities': ['Genre']
    },
    {
        'intent': 'FindMoviesByDirector',
        'pattern': r'which movies\s+(did|were)\s+(?P<Person>.+)\s+(direct|directed)',
        'entities': ['Person']
    },
```

- Fallback to spaCy NER for entity recognition.

```python
def parse_question(question):
    question = question.lower().strip('?').strip()
    for pattern_info in INTENT_PATTERNS:
        match = re.match(pattern_info['pattern'], question)
        if match:
            intent = pattern_info['intent']
            entities = {entity: match.group(entity).strip().lower() for entity in pattern_info['entities']}
            return intent, entities

    # Fallback to spaCy NER
    doc = nlp(question)
    entities = {}
    for ent in doc.ents:
        if ent.label_ == 'PERSON':
            entities['Person'] = ent.text
        elif ent.label_ in ['WORK_OF_ART', 'MOVIE']:
            entities['Movie'] = ent.text
        elif ent.label_ == 'LANGUAGE':
            entities['Language'] = ent.text
        elif ent.label_ == 'ORG':
            entities['Company'] = ent.text
        elif ent.label_ == 'GPE':
            entities['Country'] = ent.text
        elif ent.label_ == 'NORP':
            entities['Genre'] = ent.text
    intent = 'Unknown' if not entities else 'FindInformation'
    return intent, entities
```

- **Intermediate Output:** Intent and entities (e.g., {'intent': 'FindDirector', 'entities': {'Movie': 'Inception'}}).

---

6. Query Generation (Query_generator.py)

- **Purpose:** Generates and executes Cypher queries based on user intent.

- **Features:**
    - Predefined query templates for intents like FindActors, FindRevenue.

```
    def generate_query(self, intent, entities):
        if intent == 'FindDirector':
            movie_title = entities.get('Movie')
            query = """
            MATCH (m:Movie)<-[:DIRECTED_BY]->(d:Person)
            WHERE toLower(m.title) = toLower($movie_title)
            RETURN d.name AS director
            """
            parameters = {'movie_title': movie_title}
            return query, parameters

        elif intent == 'FindActors':
            movie_title = entities.get('Movie')
            query = """
            MATCH (a:Person)<-[:ACTED_IN]->(m:Movie)
            WHERE toLower(m.title) = toLower($movie_title)
            RETURN a.name AS actor
            """
            parameters = {'movie_title': movie_title}
            return query, parameters
```

    - Fetches results from Neo4j and formats responses.

```
    def get_response(self, intent, entities):
        query, parameters = self.generate_query(intent, entities)
        if not query:
            return "I'm sorry, I couldn't understand your question."

        results = self.execute_query(query, parameters)
        if not results:
            return "I'm sorry, I couldn't find any results."

        if intent == 'FindDirector':
            directors = [record['director'] for record in results]
            movie_title = entities.get('Movie').title()
            directors_list = ', '.join(director.title() for director in directors)
            return f"The director of '{movie_title}' is {directors_list}."

        elif intent == 'FindActors':
            actors = [record['actor'] for record in results]
            movie_title = entities.get('Movie').title()
            actors_list = ', '.join(actor.title() for actor in actors)
            return f"The actors in '{movie_title}' are: {actors_list}."
```

- **Issues Encountered:** Complex queries for trends required optimization by batching and limiting results.

---

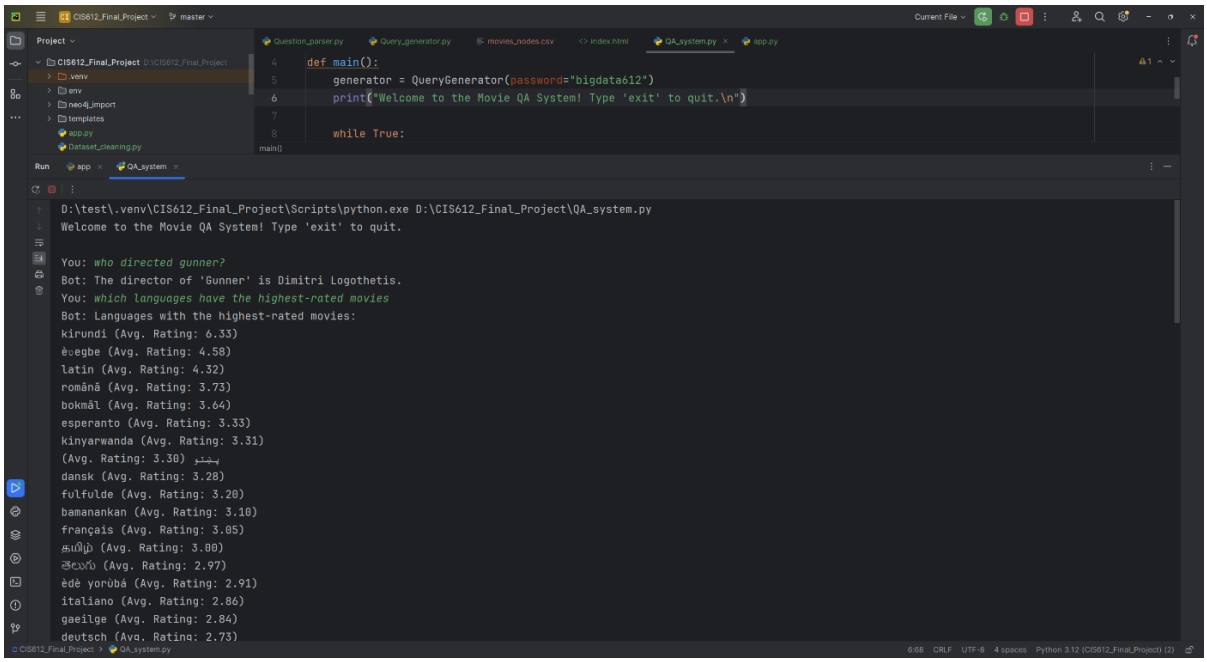
7. **QA System (QA_system.py and app.py)**
   - **Purpose:** Provides an interface for users to interact with the system.
   - **Modes:**
     - **Command-line Mode (QA_system.py):** Accepts user questions and prints responses.

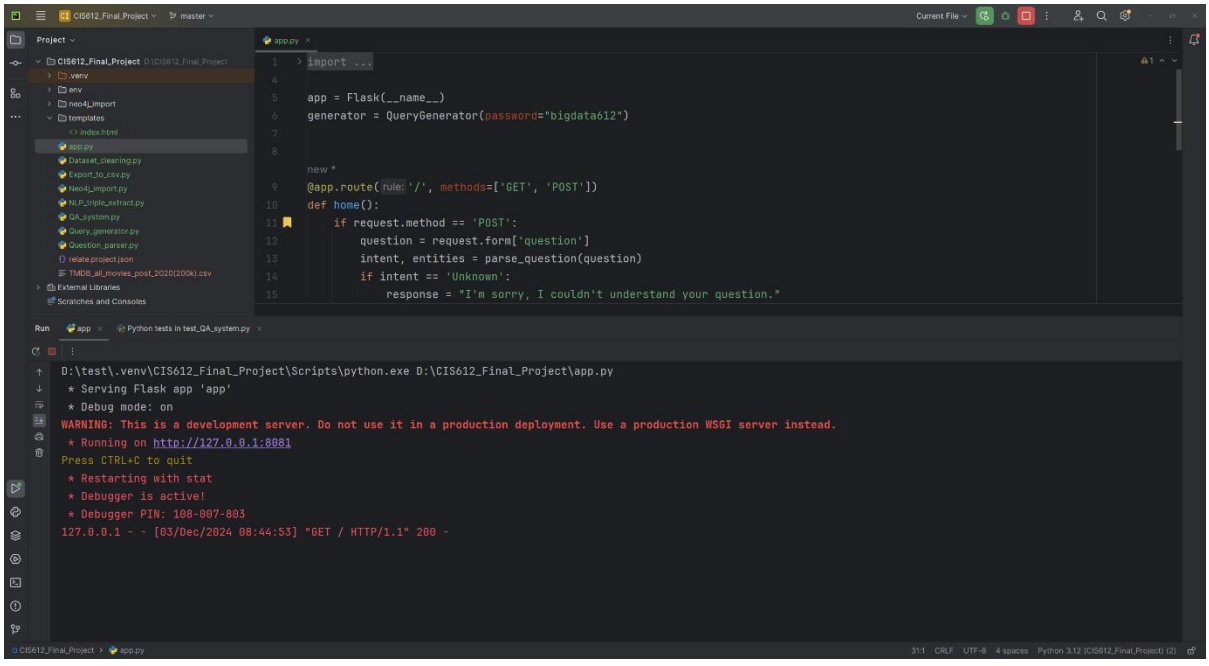
```python
from Question_parser import parse_question
from Query_generator import QueryGenerator


new *
def main():
    generator = QueryGenerator(password="bigdata612")
    print("Welcome to the Movie QA System! Type 'exit' to quit.\n")

    while True:
        question = input("You: ")
        if question.lower() in ['exit', 'quit']:
            break

        intent, entities = parse_question(question)
        if intent == 'Unknown':
            print("Bot: I'm sorry, I couldn't understand your question.")
            continue

        response = generator.get_response(intent, entities)
        print(f"Bot: {response}")

    generator.close()
    print("Goodbye!")


if __name__ == '__main__':
    main()
```

- **Web Interface (app.py):** Flask application with a form-based UI.

```python
app = Flask(__name__)
generator = QueryGenerator(password="bigdata612")


new *
@app.route( rule: '/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        question = request.form['question']
        intent, entities = parse_question(question)
        if intent == 'Unknown':
            response = "I'm sorry, I couldn't understand your question."
        else:
            response = generator.get_response(intent, entities)
        return render_template( template_name_or_list: 'index.html', question=question, response=response)
    return render_template('index.html')


2 usages (2 dynamic)   new *
@app.route('/shutdown')
def shutdown():
    generator.close()
    func = request.environ.get('werkzeug.server.shutdown')
    if func is None:
        raise RuntimeError('Not running with the Werkzeug Server')
    func()
    return 'Server shutting down...'


if __name__ == '__main__':
    app.run(debug=True, port=8081)
```

- **Execution:**


def main():
generator = QueryGenerator(password="bigdata612")
print("Welcome to the Movie QA System! Type 'exit' to quit.\n")
while True:
D:\test\.venv\CIS612_Final_Project\Scripts\python.exe D:\CIS612_Final_Project\QA_system.py
Welcome to the Movie QA System! Type 'exit' to quit.
You: who directed gunner?
Bot: The director of 'Gunner' is Dimitri Logothetis.
You: which languages have the highest-rated movies
Bot: Languages with the highest-rated movies:
kirundi (Avg. Rating: 6.33)
èʋegbe (Avg. Rating: 4.58)
latin (Avg. Rating: 4.32)
română (Avg. Rating: 3.73)
bokmål (Avg. Rating: 3.64)
esperanto (Avg. Rating: 3.33)
kinyarwanda (Avg. Rating: 3.31)
(Avg. Rating: 3.30) پښتو
dansk (Avg. Rating: 3.28)
fulfulde (Avg. Rating: 3.20)
bamanankan (Avg. Rating: 3.10)
français (Avg. Rating: 3.05)
தமிழ் (Avg. Rating: 3.00)
తెలుగు (Avg. Rating: 2.97)
èdè yorùbá (Avg. Rating: 2.91)
italiano (Avg. Rating: 2.86)
gaeilge (Avg. Rating: 2.84)
deutsch (Avg. Rating: 2.73)


import ...
app = Flask(__name__)
generator = QueryGenerator(password="bigdata612")
@app.route('/', methods=['GET', 'POST'])
def home():
if request.method == 'POST':
question = request.form['question']
intent, entities = parse_question(question)
if intent == 'Unknown':
response = "I'm sorry, I couldn't understand your question."
D:\test\.venv\CIS612_Final_Project\Scripts\python.exe D:\CIS612_Final_Project\app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:8081
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 108-007-803
127.0.0.1 - - [03/Dec/2024 08:44:53] "GET / HTTP/1.1" 200 -

❖ **Problems and Resolutions**

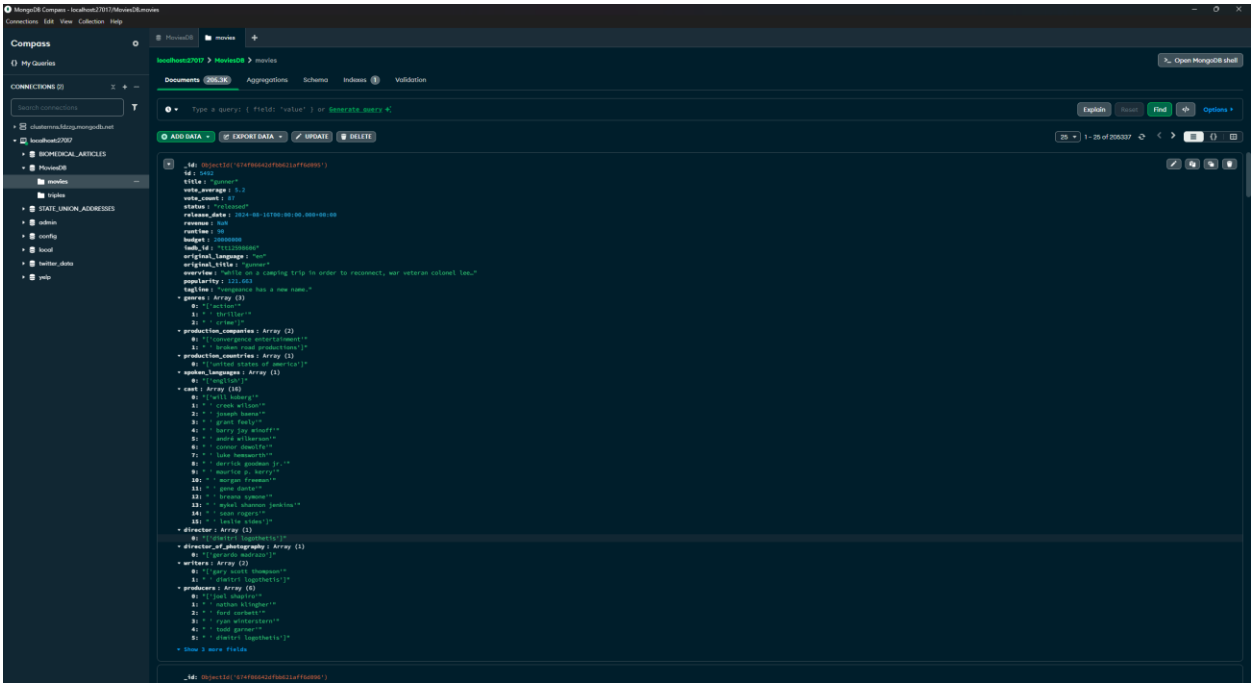
- **CoreNLP Port Conflicts:** Resolved by dynamic port allocation.

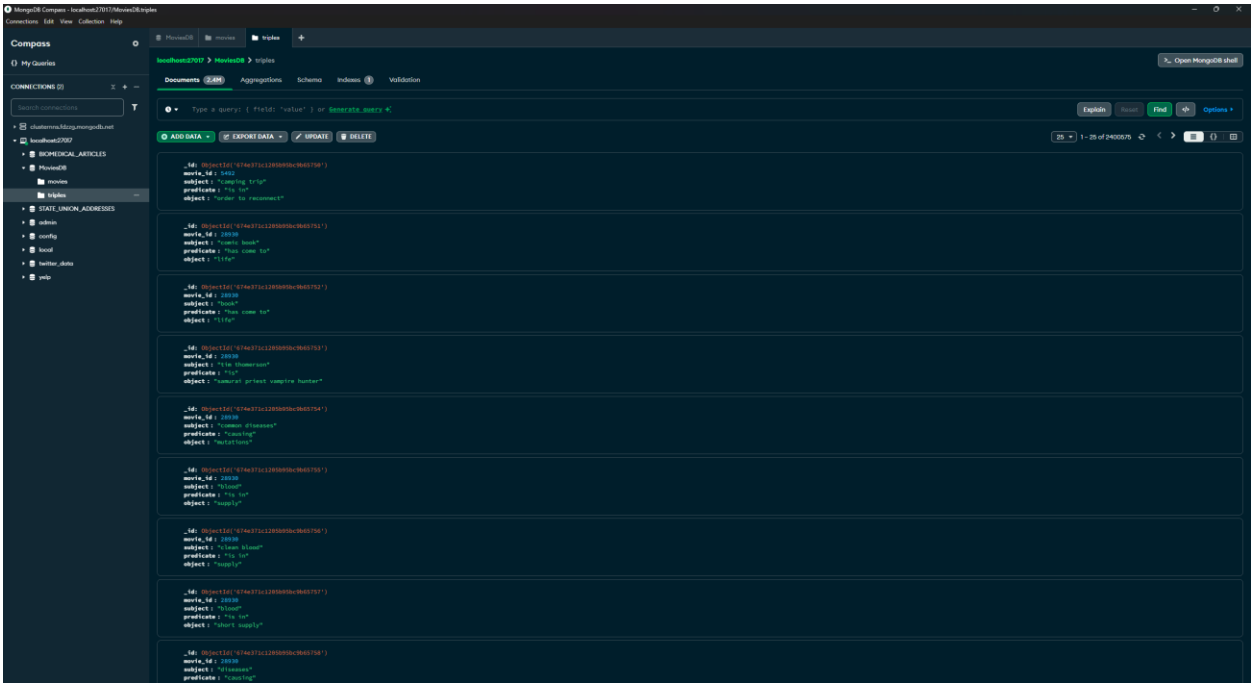
```python
def find_free_port():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(('', 0))
        return s.getsockname()[1]


1 usage   new *
def kill_corenlp_process(port=9000):
    for proc in psutil.process_iter():
        try:
            for conn in proc.connections(kind="inet"):
                if conn.laddr.port == port:
                    proc.terminate()
        except Exception:
            pass
```

- **Data Type Issues:** Resolved by explicit typecasting in Dataset_cleaning.py and Export_to_csv.py.
- **Neo4j import issue:** Importing a large dataset (~2.2M nodes, ~7.8M relationships, ~211K relationship types) caused bottlenecks.
    - Using Python Driver Import was not feasible due to prolonged import times and performance issues**.**
    - Solution: Neo4j Admin Import due to its ability to import large datasets in under a minute.
- **Building the QA System issue**: We planned to use Neosemantics (n10s) to build the QA system using SPARQL, which would enable semantic queries on RDF data.
    - The newer version of Neo4j no longer supports Neosemantics, limiting our ability to integrate RDF data and execute SPARQL queries.
    - Solution: Adapted the system to use Cypher Queries, leveraging Neo4j's native query language to design the Q&A functionality.

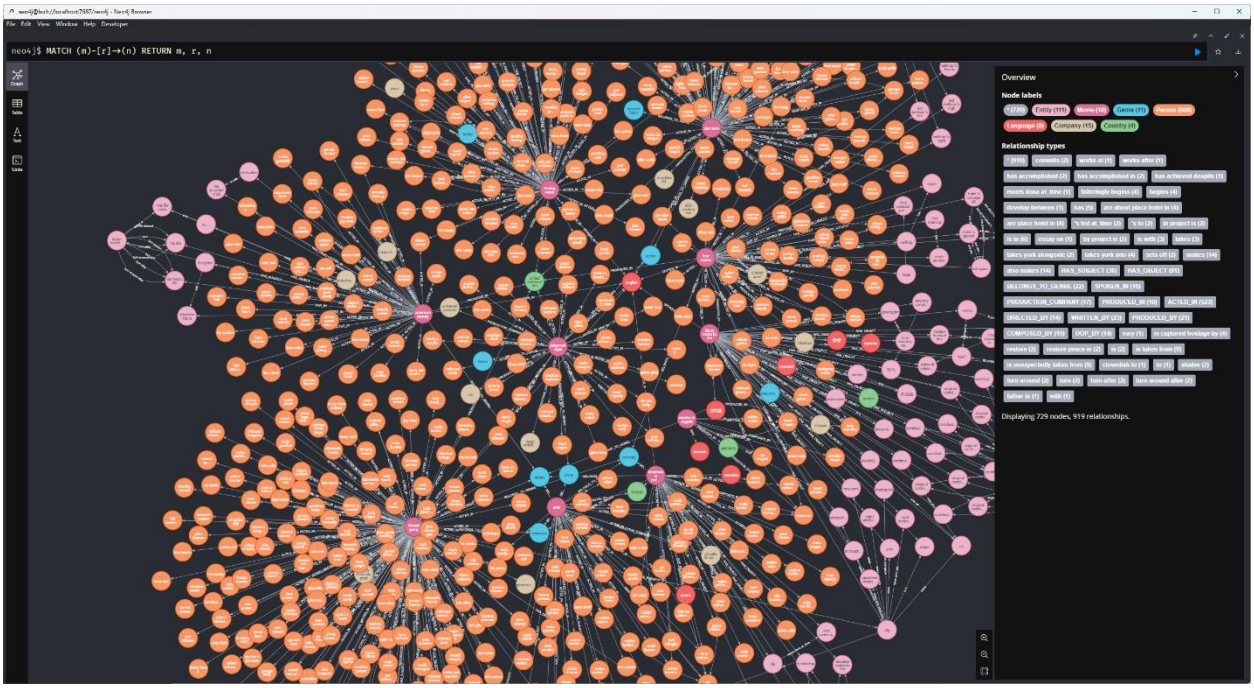
❖ Final Output

- Cleaned dataset stored in MongoDB.

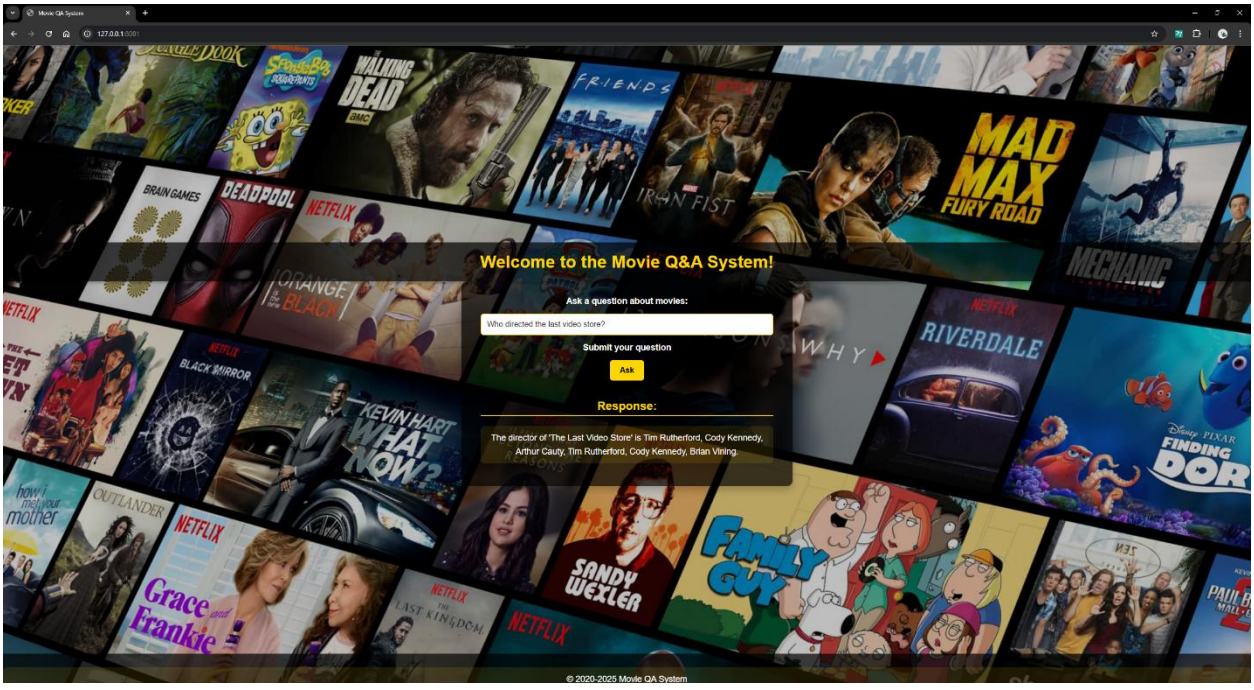


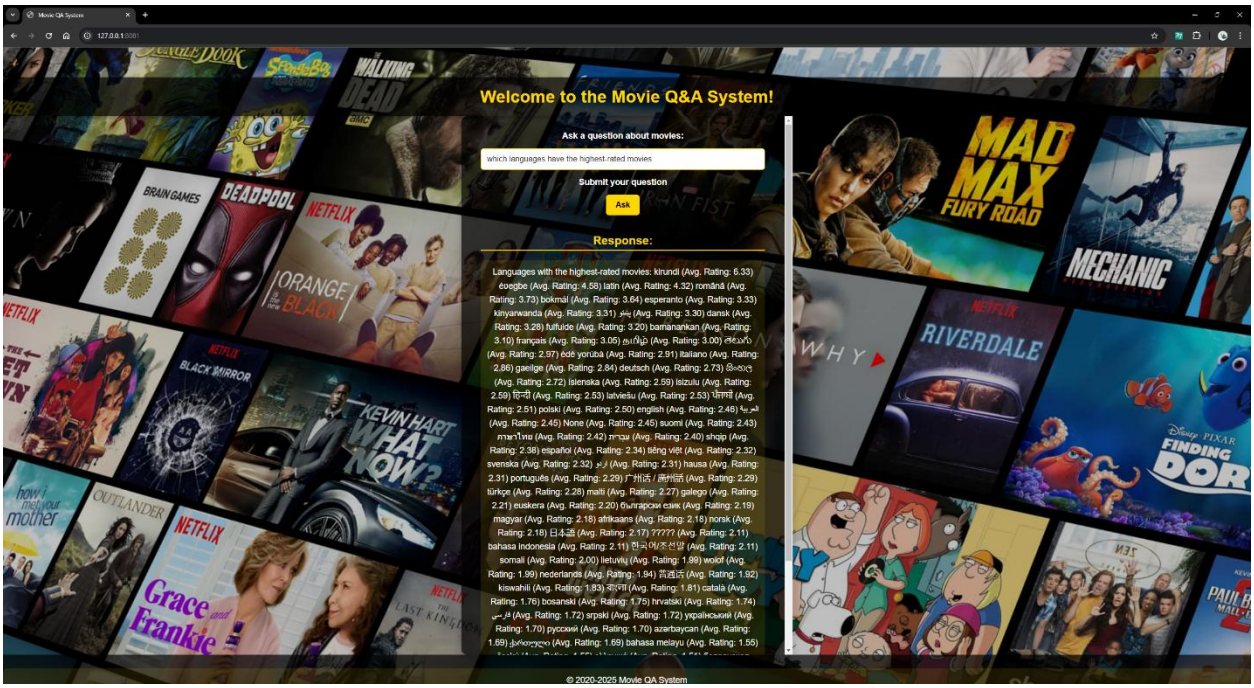


- Knowledge graph nodes and relationships imported into Neo4j.



- A functional QA system capable of answering movie-related questions using the knowledge graph.





❖ Conclusion

- Successfully integrated NLP and graph databases for robust query handling.
- Enhanced natural language understanding with CoreNLP and spaCy.
- Demonstrated scalability for larger datasets and real-world applications.