**Project Documentation**

1. **Database Schema**

Book\_management db createdin postgre sql.

Steps to connect to db ,

1. To start the server

Server start --pg\_ctl start -D "C:\PostgreSQL\17\data" -l logfile

1. To start sending queries,

psql -U postgres -h localhost

**Books Table**

sql

Copy code

CREATE TABLE books (

id SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

author VARCHAR(255) NOT NULL,

genre VARCHAR(100),

year\_published INT,

summary TEXT

);

**Reviews Table**

sql

Copy code

CREATE TABLE reviews (

id SERIAL PRIMARY KEY,

book\_id INT REFERENCES books(id),

user\_id VARCHAR(50),

review\_text TEXT,

rating INT CHECK (rating >= 1 AND rating <= 5)

);

References:

A black screen with white text

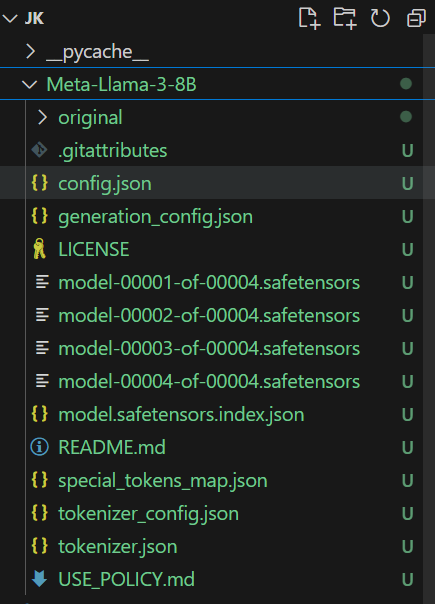
Description automatically generated

**2. Llama3 Integration**

**Setting Up Llama3 Model**

* **Model:** Meta Llama 3 8B
* **Location:** Cloned from [meta-llama/Meta-Llama-3-70B · Hugging Face](https://huggingface.co/meta-llama/Meta-Llama-3-70B)

This files should be inside it.



Running the script:

1. Scripts :
2. Summarizer.py: That has the model imported and basic function to generate summary from Llama model.

from transformers import AutoTokenizer, AutoModelForCausalLM

import torch

# Load model and tokenizer

tokenizer = AutoTokenizer.from\_pretrained("C:\\Users\\PE586UG\\OneDrive - EY\\Documents\\Gen AI\\jk\\Meta-Llama-3-8B")

model = AutoModelForCausalLM.from\_pretrained("C:\\Users\\PE586UG\\OneDrive - EY\\Documents\\Gen AI\\jk\\Meta-Llama-3-8B").to('cuda' if torch.cuda.is\_available() else 'cpu')

# Add padding token if not present

if tokenizer.pad\_token is None:

tokenizer.add\_special\_tokens({'pad\_token': tokenizer.eos\_token})

def generate\_summary(text, max\_length=150):

# Tokenize the input text

inputs = tokenizer(

text,

return\_tensors='pt',

padding=True,

truncation=True,

max\_length=max\_length

).to('cuda' if torch.cuda.is\_available() else 'cpu')

# Generate summary

try:

summary\_ids = model.generate(

inputs['input\_ids'],

attention\_mask=inputs['attention\_mask'],

max\_length=max\_length,

num\_beams=4,

length\_penalty=2.0,

early\_stopping=True,

pad\_token\_id=tokenizer.pad\_token\_id,

temperature=0.7,

top\_k=50,

top\_p=0.95

)

# Decode the summary

summary = tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)

return summary

except Exception as e:

print(f"Error during generation: {e}")

return None

"""

# Test the function

if \_\_name\_\_ == "\_\_main\_\_":

test\_text = "The Great Gatsby is a novel by American writer F. Scott Fitzgerald, published in 1925. It tells the story of Jay Gatsby, a wealthy man who is known for his lavish parties, and his obsession with the beautiful former debutante Daisy Buchanan."

summary = generate\_summary(test\_text)

print("Generated Summary:", summary)"""

1. Summarize\_books – In this , we have used generate\_summary function from our summarizer file and used it to generate summaries of the book based on contents.

Here, user needs to provide in input the book id and will get back the output.

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

from models import Book # Make sure your Book model is imported

from jk\_main import generate\_summary # Adjust the import based on your file structure

# Update with your actual database details

DATABASE\_URL = "postgresql://{user}:{password}@localhost:5432/book\_management"

# Create engine and session

engine = create\_engine(DATABASE\_URL, echo=True)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def test\_connection():

try:

# Create a connection to the database

with engine.connect() as connection:

print("Connection to the database is successful!")

except Exception as e:

print(f"Error connecting to the database: {e}")

def get\_book\_by\_id(db, book\_id: int):

return db.query(Book).filter(Book.id == book\_id).first()

def main(book\_id: int):

# Test the database connection

test\_connection()

# Create a new session

db = SessionLocal()

try:

# Get the book by ID

book = get\_book\_by\_id(db, book\_id)

if book:

print(f"Book Title: {book.title}")

print(f"Original Summary: {book.summary}")

# Generate summary

summary = generate\_summary(book.summary)

print(f"Generated Summary: {summary}")

# Optionally, you can update the book's summary in the database

book.summary = summary

db.commit()

print("Book summary updated in the database.")

else:

print("Book not found.")

except Exception as e:

print(f"An error occurred: {e}")

finally:

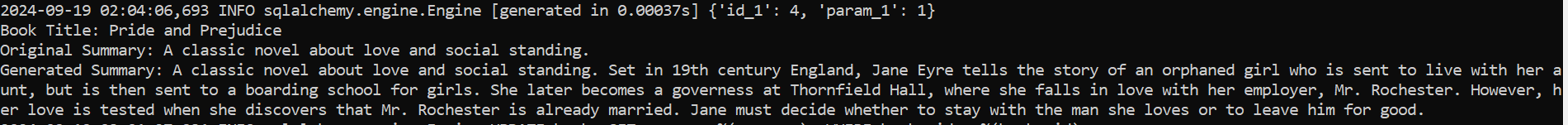
db.close()

if \_\_name\_\_ == "\_\_main\_\_":

book\_id = int(input("Enter the book ID: ")) # Get the book ID from user input

main(book\_id)

Reference, you can see the generates summary from the model.



**3.Machinelearning Models**

1. Create new table named,

CREATE TABLE books\_rating (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

rating\_dist\_1 VARCHAR(255),

pages\_number INT,

rating\_dist\_4 VARCHAR(255),

rating\_dist\_total VARCHAR(255),

publish\_month INT CHECK (publish\_month BETWEEN 1 AND 12),

publish\_day INT CHECK (publish\_day BETWEEN 1 AND 31),

publisher VARCHAR(255),

counts\_of\_review INT,

publish\_year INT CHECK (publish\_year >= 1900 AND publish\_year <= EXTRACT(YEAR FROM CURRENT\_DATE)),

language VARCHAR(50),

authors VARCHAR(255),

rating DECIMAL(3, 2) CHECK (rating >= 0 AND rating <= 5),

rating\_dist\_2 VARCHAR(255),

rating\_dist\_5 VARCHAR(255),

isbn VARCHAR(20),

rating\_dist\_3 VARCHAR(255)

);

1. Downloaded dataset named book1-100k.csv from [Goodreads Book Datasets With User Rating 2M (kaggle.com)](https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m?resource=download)
2. Cleaned it using pandas and wrote to sql table named new\_books\_rating

Script doing so is –load\_testdata.py.

import pandas as pd

from sqlalchemy import create\_engine

import pandas as pd

# Load the CSV file with headers

df = pd.read\_csv(r'C:\Users\PE586UG\OneDrive - EY\Documents\Gen AI\jk\book\_ratings.csv')

# Display the first few rows to inspect the data

print("Original DataFrame:")

print(df.head())

# Clean the 'Name' column and any other necessary columns

df['Name'] = df['Name'].str.replace('Harry Potter', 'Harry Potter', regex=False)  # Adjust as needed

# Display the cleaned DataFrame

print("Cleaned DataFrame:")

print(df.head())

# Optionally, you can check for missing values

print("Missing values in each column:")

print(df.isnull().sum())

# Load the dataset

final\_dataset = pd.read\_csv("book\_rating\_data.csv")

# Create a SQLAlchemy engine

#add your url

DATABASE\_URL = "postgresql://{user}:{password}@localhost:5432/book\_management"

engine = create\_engine(DATABASE\_URL)

# Write the DataFrame to a new SQL table

final\_dataset.to\_sql('new\_books\_rating', engine, if\_exists='replace', index=False)

1. Script having the ML code. ‘book\_recommendation.py’

This will take your book name and recommend based on your interest.

import pandas as pd

from sklearn.neighbors import NearestNeighbors

import numpy as np

import re

# Load your dataset

df = pd.read\_csv("book\_rating\_data.csv")

# Enhanced data cleaning function

def clean\_data(df):

    # Remove unwanted characters and convert to numeric

    df['RatingDistTotal'] = df['RatingDistTotal'].apply(lambda x: re.sub(r'\D', '', str(x)))  # Keep only digits

    df['RatingDistTotal'] = pd.to\_numeric(df['RatingDistTotal'], errors='coerce')  # Convert to numeric

    df['CountsOfReview'] = pd.to\_numeric(df['CountsOfReview'], errors='coerce')  # Convert to numeric

    df['pagesNumber'] = pd.to\_numeric(df['pagesNumber'], errors='coerce')  # Convert to numeric

    df['Rating'] = pd.to\_numeric(df['Rating'], errors='coerce')  # Convert to numeric

    # Drop rows with NaN values in important columns

    df = df.dropna(subset=['RatingDistTotal', 'CountsOfReview', 'pagesNumber', 'Rating'])

    return df

# Clean the DataFrame

df = clean\_data(df)

# Train the Nearest Neighbors model using additional features

features = df[['RatingDistTotal', 'CountsOfReview', 'pagesNumber', 'Rating']]

model = NearestNeighbors(n\_neighbors=5)

model.fit(features)

def recommend\_books(book\_name, model, df):

    """

    Recommend books based on the input book name.

    Parameters:

    - book\_name: str, the name of the book for which recommendations are needed.

    - model: NearestNeighbors model, the trained model for making recommendations.

    - df: DataFrame, the dataset containing book information.

    Returns:

    - List of recommended book names based on the input book.

    """

    book\_name\_normalized = book\_name.strip()

    if book\_name\_normalized in df['Name'].values:

        idx = df[df['Name'] == book\_name\_normalized].index[0]

        # Extract features for the selected book

        book\_features = df[['RatingDistTotal', 'CountsOfReview', 'pagesNumber', 'Rating']].iloc[idx].values.reshape(1, -1)

        # Get the nearest neighbors

        distances, indices = model.kneighbors(book\_features)

        # Get recommended book names based on the indices

        recommended\_books = df['Name'].iloc[indices.flatten()].tolist()

        return recommended\_books

    else:

        return []

# Get user input for the book name

input\_book\_name = input("Enter the name of the book you're interested in: ")

# Get recommendations

recommended\_books = recommend\_books(input\_book\_name, model, df)

# Output the recommended books

if recommended\_books:

    print("Recommended books based on your interest:")

    for book in recommended\_books:

        print(book)

else:

    print("Sorry, the book you entered is not in the dataset.")

Based on rating, total\_reviews,etc the recommendation gets generated.

A screen shot of a computer

Description automatically generated