Infosys Springboard Virtual Inter nship

“Knowmap Cross Domain Knowledge Mapping UsingAI

Milestone - Admin Tools,Feedback System and Deployment.

Name: Rekhansh
Date : 01-11-2025

Admin Tools,Feedback System and Deployment.

1.Introduction

The objective of Milestone 4 is to enhance the Knowledge Graph system by integrating advanced administrative tools, user feedback mechanisms, and deployment capabilities. This milestone focuses on building a comprehensive Admin Dashboard that allows monitoring and management of the entire knowledge extraction pipeline. It provides key metrics such as total entities, relations, and estimated pipeline accuracy while enabling manual correction and node management within the knowledge graph.
Additionally, this milestone introduces a feedback system where users can rate the relevance and usefulness of the generated graphs.

2.Objective

- Build an admin dashboard for monitoring graph data.
- Show metrics like entities, relations, and accuracy. summarization and paraphrasing accuracy. Store summaries, paraphrases, and readability analysis results in dedicated
- database tabl
- Support visualization and management of extracted triples.
-
  1)Summaries History
  2)Paraphrases History
  3)Readability Analysis History
- Allow users to download history as CSV for offline use.
- Ensure seamless integration with existing authentication and profile system.

3.Workflow

- Users input text for summarization; system generates results using chosen models and saves them with scores.
- For paraphrasing, text is rephrased with selected models and stored with complexity/creativity metadata.
- ROUGE scoring evaluates generated summaries for quality.
- All outputs (summaries, paraphrases, readability) are stored in the database.
- A dedicated History tab retrieves past analyses with quick previews.
- Users can expand individual records for full content.
- Export options allow downloading history as CSV files.
- Integration ensures smooth navigation alongside Dashboard and Profile features.

4.Code Implementation

dashboard.py :

```
 with main_tab2:
st.subheader("Summarize Text or PDF")
col1, col2 = st.columns(2)
with col1:


     input_type = st.radio("Choose input type:", ["Plain Text","Text File","PDF File"])
     model_choice = st.selectbox(
       "Select Model",
       options=["pegasus", "bart", "flan-t5"],
       index=0
     )
     s u m m a r y _ l e n g t h   =   s t . s e l e c t b o x (
       "Summary Length",
       options=["short", "medium", "long"],
       index=1
     )
 with col2:
     st.write("Instructions:")
     st.markdown("- Paste text or upload a .txt or .pdf file.\n- Choose a model and
summary length.\n- Click 'Generate Summary'.")
uploaded_file = None

   text_input=""
   if input_type == "Plain Text":
     text_input = st.text_area("Paste your text here", height=200)
   elif input_type == "Text File":
     uploaded_file = st.file_uploader("Upload a TXT file", type=["txt"],
key="file_uploader_txt")       elif
input_type == "PDF File":
     uploaded_file = st.file_uploader("Upload a PDF file", type=["pdf"],
key="file_uploader_pdf")


   reference_input = st.text_area(
     "Reference Summary (optional for ROUGE evaluation)",
     height=150,
     help="Paste a human-written summary here to compute ROUGE metrics."
   )
```

```
    if'last_summary' not in st.session_state:
      st.session_state['last_summary']  =  None
      st.session_state['last_model']     =    None
      st.session_state['last_length'] = None

    ifst.button("Generate Summary"):
      try:
        original_text_display = ""
        ifinput_type == "Plain Text":
          original_text_display = text_input.strip()
        elifinput_type == "Text File" and uploaded_file:
              original_text_display = uploaded_file.getvalue().decode("utf-
8").strip()
        elifinput_type == "PDF File" and uploaded_file:
            withpdfplumber.open(io.BytesIO(uploaded_file.getvalue())) as
pdf:
                original_text_display = "\n".join(page.extract_text() or "" for
page in pdf.pages).strip()
        if not original_text_display:
          st.error("No valid text found to summarize.")
        else:
          withst.spinner("Generating summary..."):
            model_map = {
              "pegasus": "google/pegasus-xsum",
              "bart": "facebook/bart-large-cnn",
              "flan-t5": "google/flan-t5-large"
            }
            summarizer = load_summarizer(model_map[model_choice])
                length_map = {"short": (20, 60), "medium": (60, 120), "long":
(120, 200)}
            min_len, max_len = length_map[summary_length]
            result= summarizer(original_text_display, min_length=min_len,
max_length=max_len, do_sample=False)
            summary_text_display = result[0]['summary_text'].strip()
            wc_orig = len(original_text_display.split())
            wc_sum = len(summary_text_display.split())
             compression = (1 - (wc_sum / wc_orig)) * 100 if wc_orig > 0 else
0
            col_orig, col_sum = st.columns(2)
            with col_orig:
              st.markdown("#### Original Text")
          st.caption(f"{wc_orig} words")
          st.text_area("Original", value=original_text_display[:15000], height=260)
        withcol_sum:
        st.markdown("#### Summary")
        st.caption(f"{wc_sum} words")
        st.text_area("Summary", value=summary_text_display, height=260)
        st.markdown(f"**Compression:** {compression:.0f}%")
        st.session_state['last_summary'] = summary_text_display
        st.session_state['last_model'] = model_choice
        st.session_state['last_length'] = summary_length
        scores = {}
        if reference_input.strip():
          rouge = evaluate.load("rouge")
          scores=rouge.compute(predictions=[summary_text_display], references=
[reference_input.strip()])
          st.markdown("### ROUGE Evaluation")
          st.json(scores)
          df_scores=pd.DataFrame(list(scores.items()), columns=["Metric",
" Score"])
          st.dataframe(df_scores)
          csv_bytes = df_scores.to_csv(index=False).encode()
          st.download_button("Download ROUGE CSV", data=csv_bytes,
 file_name="rouge_scores.csv")
          st .markdown("""
           🔍 **Whatthis shows:**
           Thebarsbelow compare your **generated summary** with the
**referencesummary**.
           -**ROUGE-1** → word overlap
           -**ROUGE-2** → two-word phrase overlap
           -**ROUGE-L** → longest common sequence
           Highervalues = summary is closer in meaning to the reference.
          """)
          fig,ax=plt.subplots()
          ax.bar(df_scores['Metric'], df_scores['Score'], color="#3b82f6")
          ax.set_ylim(0, 1)
          ax.set_ylabel("Score")
          ax.set_title("ROUGE Scores")
          st .pyplot(fig)
        store_summary_db(
          user_email=user_email,
            o riginal_text=original_text_display,
           summary_text=summary_text_display,
           model _used=model _choice,
           summary_length=summary_length,
           reference_summary=reference_input.strip(),
           rouge_scores=scores if reference_input.strip() else {}
         )
     except Exception as e:
       st.error(f"An error occurred: {e}")



  defstore_paraphrase_db(user_email, original_text, paraphrased_results,
model_used, creativity, complexity_level, rouge_scores, readability_scores):
    payload = {
      "user_email": user_email,
      "original_text": original_text,
      "paraphrased_options": paraphrased_results,
      "model_used": model_used,
      "creativity": creativity,
      "complexity_level": complexity_level,
      "rouge_scores": rouge_scores,
      "readability_scores": readability_scores
    }
    try:
      resp = requests.post(f"{API_URL}/store_paraphrase", json=payload, timeout=10)
      ifresp.status_code == 200:
        st.success("All paraphrases stored successfully in DB")
      else:
        st.error(f"Failed to store paraphrases: {resp.text}")
    except requests.exceptions.RequestException as e:
      st.error(f"Could not connect to backend: {e}")

  def paraphrasing_ui(user_email):
    st.subheader("Paraphrasing & Analysis")
    input_method = st.radio("Choose input method:", ["Text Input", "File Upload"],
horizontal=True)
    original_text = ""
    ifinput_method == "Text Input":
      original_text = st.text_area("Enter text to paraphrase", height=200)
    else:
      uploaded_file = st.file_uploader("Upload a .txt, .pdf, or .docx file", type=
["txt","pdf","docx"])
```

```
        if uploaded_file:
        file_bytes = uploaded_file.getvalue()
        if uploaded_file.type == "application/pdf":
          import PyPDF2, io
          pdf_reader = PyPDF2.PdfReader(io.BytesIO(file_bytes))
          original_text="".join(page.extract_text() or "" for page in
pdf_reader.pages)
        elif uploaded_file.type == "application/vnd.openxmlformats-
officedocument .wordprocessingml .document ":
          importdocx,io
          doc = docx.Document(io.BytesIO(file_bytes))
          original_text="\n".join(para.text for para in doc.paragraphs)
        else:
          original_text = file_bytes.decode("utf-8")
    col1, col2 = st.columns(2)
    with col1:
      creativity=st.slider("Creativity", 0.5, 1.5, 1.0, 0.1)
    with col2:
      complexity_level=st.selectbox("Complexity Level", ["Beginner",
"Intermediate", "Advanced"])
    paraphrase_models = {
      "T5Paraphraser(Humarin)": "humarin/chatgpt_paraphraser_on_T5_base",
      "Pegasus (Google)": "tuner007/pegasus_paraphrase",
      "BART (Facebook)": "eugenesiow/bart-paraphrase"
    }
    selected_model=st.selectbox("Select Model",
list(paraphrase_models.keys()))
    complexity_map={"Beginner": 128, "Intermediate": 256, "Advanced": 512}
    max_len = complexity_map[complexity_level]
    complexity_prompt_map = {
      "Beginner":"Paraphrasethe following text in simple and clear language
suitable for beginners:",
      "Intermediate":"Paraphrase the following text with moderate complexity
suitable for intermediate readers:",
      "Advanced":"Paraphrasethe following text with advanced vocabulary and
sentencestructuresuitablefor expert readers:"
    }
    prompt_text=complexity_prompt_map[complexity_level] + "\n" +
original_text
    ifst.button("Generate&Analyze", type="primary") and original_text.strip():
      para_pipe = load_paraphraser(paraphrase_models[selected_model])
    outputs=para_pipe(
      prompt _text ,
      num_return_sequences=3,
      num_beams=5,
      temperature=creativity,
      max_length=max_len,
      truncation=True
    )
    paraphrased_results= [o["generated_text"] for o in outputs]
    st.subheader("Paraphrased Options")
    fori,txtinenumerate(paraphrased_results, 1):
      st.write(f"**Option{i}:**")
      st .info(txt)
    importtextstat,pandas as pd, plotly.express as px
    complexity_data=[{"Source": "Original", "Score":
textstat.flesch_reading_ease(original_text)}]
    fori,txtinenumerate(paraphrased_results, 1):
      complexity_data.append({"Source": f"Option {i}", "Score":
textstat .flesch_reading _ease(txt)})
    df_complexity = pd.DataFrame(complexity_data)
    st.subheader("Readability Analysis")
    fig=px.bar(df_complexity, x="Source", y="Score",
        color="Source",title="Flesch Reading Ease", template="plotly_white")
    fig.update_layout(showlegend=False)
    st.plotly_chart(fig,use_container_width=True)
    fromrouge_scoreimport rouge_scorer
    scorer=rouge_scorer.RougeScorer(['rouge1','rouge2','rougeL'], use_stemmer=True)
    scores_data = []
    fori,txtinenumerate(paraphrased_results, 1):
      scores=scorer.score(original_text, txt)
      scores_data.append({
        "Option": f"Option {i}",
        "ROUGE-1":scores['rouge1'].fmeasure,
        "ROUGE-2": scores['rouge2'].fmeasure,
        "ROUGE-L":scores['rougeL'].fmeasure
      })
    df_scores = pd.DataFrame(scores_data)
    st.subheader("ROUGE Comparison")
    fig2 = px.bar(
      df_scores.melt(id_vars="Option", var_name="Metric", value_name="Score"),
      x="Option",y="Score", color="Metric", barmode="group",
      title="ROUGEF1-Scores vs Original", template="plotly_white"
    )
    st.plotly_chart(fig2,use_container_width=True)
    fromnltk.sentiment.vader import SentimentIntensityAnalyzer
    sid = SentimentIntensityAnalyzer()
    sentiment_orig = sid.polarity_scores(original_text)
    st.subheader("Sentiment Analysis (Original Text)")
    pie_data_orig = {k: v for k, v in sentiment_orig.items() if k != 'compound'}
    fig3 = px.pie(names=list(pie_data_orig.keys()),
values=list(pie_data_orig.values()),
        title="Original Text Sentiment", template="plotly_white")
    st.plotly_chart(fig3, use_container_width=True)
    st.json(sentiment_orig)

    sentiment_list = [sid.polarity_scores(txt) for txt in paraphrased_results]
    avg_sentiment = {k: sum(d[k] for d in sentiment_list)/len(sentiment_list) for k
in sentiment_list[0] if k != 'compound'}
    st.subheader("Average Sentiment (Paraphrased Texts)")
    fig4 = px.pie(names=list(avg_sentiment.keys()),
values=list(avg_sentiment.values()),
        title="Paraphrases Average Sentiment", template="plotly_white")
    st.plotly_chart(fig4, use_container_width=True)
    st.json(avg_sentiment)

    combined_text = "Original:\n" + original_text + "\n\n"
    for i, txt in enumerate(paraphrased_results, 1):
      combined_text += f"Option {i}:\n{txt}\n\n"

    store_paraphrase_db(
      user_email=user_email,
      original_text=original_text,
      paraphrased_results=paraphrased_results,
      model_used=selected_model,
      creativity=creativity,
      complexity_level=complexity_level,
      rouge_scores=df_scores.to_dict(orient="records"),
      readability_scores=df_complexity.to_dict(orient="records")
    )


    st.download_button("📥 Download Paraphrases",
data=combined_text.encode("utf-8"),
         file_name="paraphrased_results.txt", mime="text/plain")

 with main_tab3:
paraphrasing_ui(user_email)
```

```
 def show_history(user_email):
  st.subheader("Your History")
  history_tab1, history_tab2, history_tab3 = st.tabs(["Summaries", "Paraphrases",
"Readability Analysis"])
  with history_tab1:
   st.markdown("### Summaries History")
   try:
    resp = requests.get(f"{API_URL}/history/summaries/{user_email}", timeout=10)
    ifresp.status_code == 200:
     data = resp.json()
     if data:
      df = pd.DataFrame(data)
      st.dataframe(df[["original_text", "summary_text", "model_used",
"summary_length", "created_at"]])
      csv_bytes = df.to_csv(index=False).encode()
      st.download_button("Download CSV", data=csv_bytes,
file_name="summaries_history.csv")
     else:
      st.info("No summaries found.")
    else:
     st.error(f"Failed to fetch summaries: {resp.text}")
   except requests.exceptions.RequestException as e:
    st.error(f"Could not connect to backend: {e}")
  with history_tab2:
   st.markdown("### Paraphrases History")
   try:
    resp = requests.get(f"{API_URL}/history/paraphrases/{user_email}",
timeout=10)
    ifresp.status_code == 200:
     data = resp.json()
     if data:
      df = pd.DataFrame(data)
      st.dataframe(df[["original_text", "paraphrased_options", "model_used",
"complexity_level", "created_at"]])
      csv_bytes = df.to_csv(index=False).encode()
      st.download_button("Download CSV", data=csv_bytes,
file_name="paraphrases_history.csv")
     else:
      st.info("No paraphrases found.")
    else:
     st.error(f"Failed to fetch paraphrases: {resp.text}")
   except requests.exceptions.RequestException as e:
    st.error(f"Could not connect to backend: {e}")
 #  3⃣  Readability  Analysis  History
with history_tab3:
   st.markdown("### Readability Analysis History")
   try:
    resp = requests.get(f"{API_URL}/history/uploaded_files/{user_email}",
timeout=10)
    if resp.status_code == 200:
     data = resp.json()
     if data:
      df = pd.DataFrame(data)
      st.dataframe(df[["filename", "filetype", "filesize", "uploaded_at"]])
      # Use expanders for each file
      for i, row in df.iterrows():
       with st.expander(f"View Content: {row['filename']}"):
        try:
         content_resp = requests.get(f"
{API_URL}/history/uploaded_files/content/{row['id']}", timeout=10)
         if content_resp.status_code == 200:
          file_content = content_resp.json().get("content", "")
          # Add unique key here
          st.text_area("File Content", value=file_content, height=300,
key=f"file_content_{row['id']}")
         else:
          st.error(f"Failed to fetch content: {content_resp.text}")
        except requests.exceptions.RequestException as e:
         st.error(f"Could not connect to backend: {e}")
      csv_bytes = df.to_csv(index=False).encode()
      st.download_button("Download CSV", data=csv_bytes,
file_name="readability_history.csv")
     else:
      st.info("No readability analysis found.")
    else:
     st.error(f"Failed to fetch uploaded files: {resp.text}")
   except requests.exceptions.RequestException as e:
    st.error(f"Could not connect to backend: {e}")


 with main_tab4:
show_history(user_email)
```

api.py:

```
from fastapi import FastAPI, UploadFile, File, Form, HTTPException
from fastapi.responses import StreamingResponse
import io
import mysql.connector
from dotenv import load_dotenv
import os
from pydantic import BaseModel
import json
from datetime import datetime

dotenv_path = os.path.join(os.path.dirname(os.path.dirname(__file__)), ".env")
load_dotenv(dotenv_path)
app = FastAPI()
def get_db_connection():
try:
conn = mysql.connector.connect(
    host=os.getenv("MYSQL_HOST"),
    user=os.getenv("MYSQL_USER"),
    password=os.getenv("MYSQL_PASSWORD"),
    database=os.getenv("MYSQL_DB")
  )
  r e t u r n   c o n n
 except mysql.connector.Error as err:
raise Exception(f"Database connection failed: {err}")
def save_file_to_db(user_email, filename, filetype, filesize, data):
conn = get_db_connection()
cursor = conn.cursor()
try:
  cursor.execute(
    "INSERT INTO uploaded_files (user_email, filename, filetype, filesize, filedata)
VALUES (%s,%s,%s,%s,%s)",
    (user_email, filename, filetype, filesize, data)
  )
 conn.commit()
except mysql.connector.Error as err:
  conn.rollback()
  raise err
 finally:
  cursor.close()
  conn.close()
```

```
@app.post("/upload")
async def upload_file(user_email: str = Form(...), uploaded_file: UploadFile = File(...)):
try:
data = await uploaded_file.read()
save_file_to_db(user_email, uploaded_file.filename, uploaded_file.content_type, len(data),
data)
return {"message": "File uploaded successfully"}
except mysql.connector.Error as err:
raise HTTPException(status_code=400, detail=f"MySQL Error: {err}")
except Exception as e:
raise HTTPException(status_code=500, detail=f"Unexpected Error: {str(e)}")
@app.get("/download/{file_id}")
def download_file(file_id: int):
try:
   conn = get_db_connection()
   cursor = conn.cursor()
 cursor.execute("SELECT filename, filedata FROM uploaded_files WHERE id=%s", (file_id,))
   result = cursor.fetchone()
   cursor.close()
   conn.close()
   if result:
     filename, data = result
     return StreamingResponse(
       io.BytesIO(data),
       media_type="application/octet-stream",
       headers={"Content-Disposition": f"attachment; filename={filename}"}
     )
 raise HTTPException(status_code=404, detail="File not found")
except mysql.connector.Error as err:
raise HTTPException(status_code=400, detail=f"MySQL Error: {err}")
except Exception as e:
raise HTTPException(status_code=500, detail=f"Unexpected Error: {str(e)}")
class SummaryEvaluation(BaseModel):
user_email: str
original_text: str
summary_text: str
model_used: str
summary_length: str
reference_summary: str = ""
rouge_scores: dict = {}
@app.post("/store_evaluation")
def store_evaluation(evaluation: SummaryEvaluation):
try:
   conn = get_db_connection()
   cursor = conn.cursor()
   cursor.execute("""
     INSERT INTO summaries
     (user_email, original_text, summary_text, model_used, summary_length, reference_summary,
rouge_scores, created_at)
     VALUES (%s,%s,%s,%s,%s,%s,%s,%s)
 """, (
evaluation.user_email,
evaluation.original_text,
evaluation.summary_text,
evaluation.model_used,
evaluation.summary_length,
evaluation.reference_summary,
json.dumps(evaluation.rouge_scores),
 datetime.now()
 ))
conn.commit()
cursor.close()
conn.close()
return {"message": "Evaluation stored successfully"}
except mysql.connector.Error as err:
raise HTTPException(status_code=400, detail=f"MySQL Error: {err}")
except Exception as e:
raise HTTPException(status_code=500, detail=f"Unexpected Error: {str(e)}")

class ParaphraseEvaluation(BaseModel):
user_email: str
original_text: str
paraphrased_options: list
  model_used: str
 creativity: float
complexity_level: str
  rouge_scores: list
 readability_scores: list
@app.post("/store_paraphrase")
def store_paraphrase(evaluation: ParaphraseEvaluation):
try:
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
      INSERT INTO paraphrases
                   (user_email, original_text, paraphrased_options, model_used,           creativity,
complexity_level, rouge_scores, readability_scores, created_at)
      VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)
    """, (
      evaluation.user_email,
      evaluation.original_text,
      json.dumps(evaluation.paraphrased_options),
      evaluation.model_used,
      evaluation.creativity,
      evaluation.complexity_level, # changed here
      json.dumps(evaluation.rouge_scores),
      json.dumps(evaluation.readability_scores),
      datetime.now() ))
```

```
        conn.commit()
        cursor.close()
        conn.close()
        return {"message": "Paraphrase stored successfully"}
    except mysql.connector.Error as err:
        raise HTTPException(status_code=400, detail=f"MySQL Error: {err}")
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Unexpected Error: {str(e)}")
@app.get("/history/summaries/{user_email}")
def     get_summary_history(user_email:     str):
try:
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
      SELECT id, original_text, summary_text, model_used, summary_length,
reference_summary, rouge_scores, created_at
      FROM summaries
      WHERE user_email=%s
      ORDER BY created_at DESC
 """, (user_email,))
    data = cursor.fetchall()
    cursor.close()
    conn.close()
    return data
 except Exception as e:
raise HTTPException(status_code=500, detail=f"Error fetching summary history:
{str(e)}")
@app.get("/history/paraphrases/{user_email}")
def get_paraphrase_history(user_email: str):
try:
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
      SELECT id, original_text, paraphrased_options, model_used, creativity,
complexity_level, rouge_scores, readability_scores, created_at
      FROM paraphrases
      WHERE user_email=%s
      ORDER BY created_at DESC
 """, (user_email,))
    data = cursor.fetchall()
    cursor.close()
    conn.close()
    return data
 except Exception as e:
raise HTTPException(status_code=500, detail=f"Error fetching paraphrase
history: {str(e)}")
```

```
@app.get("/history/uploaded_files/{user_email}")
def get_readability_history(user_email: str):
try:
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
      SELECT id, filename, filetype, filesize, uploaded_at
      FROM uploaded_files
      WHERE user_email=%s
      ORDER BY uploaded_at DESC
 """, (user_email,))
    files = cursor.fetchall()
    cursor.close()
    conn.close()
    return files
 except Exception as e:
raise HTTPException(status_code=500, detail=f"Error fetching readability history:
{str(e)}")
@app.get("/history/uploaded_files/content/{file_id}")
def get_uploaded_file_content(file_id: int):
try:
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
      SELECT filename, filetype, filedata
      FROM uploaded_files
      WHERE id=%s
    """, (file_id,))
    file = cursor.fetchone()
    cursor.close()
    conn.close()
    if not file:
      raise HTTPException(status_code=404, detail="File not found")
    content = ""
    if "text" in file["filetype"]:
      content = file["filedata"].decode("utf-8")
    elif "pdf" in file["filetype"]:
      import pdfplumber
      with pdfplumber.open(io.BytesIO(file["filedata"])) as pdf:
        content = "\n".join(page.extract_text() or "" for page in pdf.pages)
    else:
      content = f"Cannot display this file type: {file['filetype']}"
 return {"filename": file["filename"], "content": content}
except Exception as e:
raise HTTPException(status_code=500, detail=f"Error fetching file content: {str(e)}")
```

5.Explanation of code

dashboard.py

1.Readability Analysis:

- Input text manuallyor upload .txt / .pdf.
- Uses Textstat to calculate readability scores (Flesch, FK Grade, SMOG, ARI).
- Resultsare color-coded (green/yellow/red) and visualized with matplotlib.

2.Summarization

- Accepts plain text, .txt, or .pdf.
- Supports BART, PEGASUS, T5 models.
- Configurable summary length: short, medium, long.
- Computes word count, compression ratio.
- Optional ROUGE evaluation if a reference summary is provided.
- Results exported as CSV and visualized in bar charts.
- Summaries stored in MySQL via FastAPI.

3.Paraphrasing & Analysis

- Inputvia text,.txt,.pdf, or .docx.
- Models: T5 (Humarin), Pegasus, BART.
- Adjustable creativity (temperature) and complexity (Beginner, Intermediate, Advanced).
- Generates 3 paraphrase options.
- Analyzes:
- Readability (Textstat + Plotly)
- ROUGE overlap vs original
- Sentiment Analysis (NLTK VADER) – Pie chart visualization.
- Stores paraphrases + metrics in DB.
- Allows download of results.

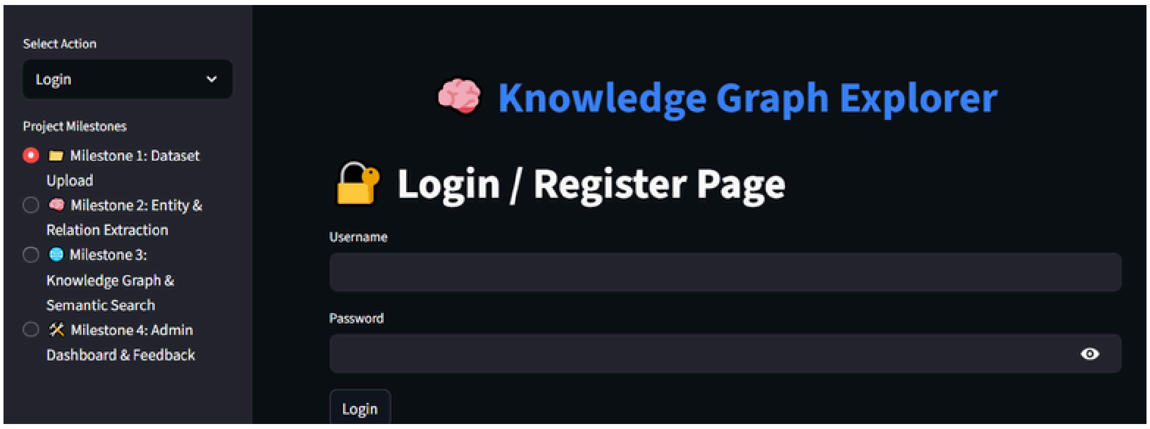
4.History

- Displays user-specific history across modules:
  1)Summaries – text, model, length, timestamp.
  2)Paraphrases – original, options, model, complexity.
  3)Readability Files – filename, type, size, upload time.’
- Expander option to view stored file content.
- History downloadable as CSV.

api.py

- Endpoints extended:
  1)POST /store_evaluation – Store summaries + ROUGE.
  2)POST /store_paraphrase – Store paraphrasing results.
  3)GET /history/* – Retrieve summaries, paraphrases, and file history.
- Maintains user-specific records in MySQL.

6.Output Screenshots



Select Action

Login

Project Milestones

● 📁 Milestone 1: Dataset Upload

○ 🧠 Milestone 2: Entity & Relation Extraction

○ 🌐 Milestone 3: Knowledge Graph & Semantic Search

○ 🛠 Milestone 4: Admin Dashboard & Feedback

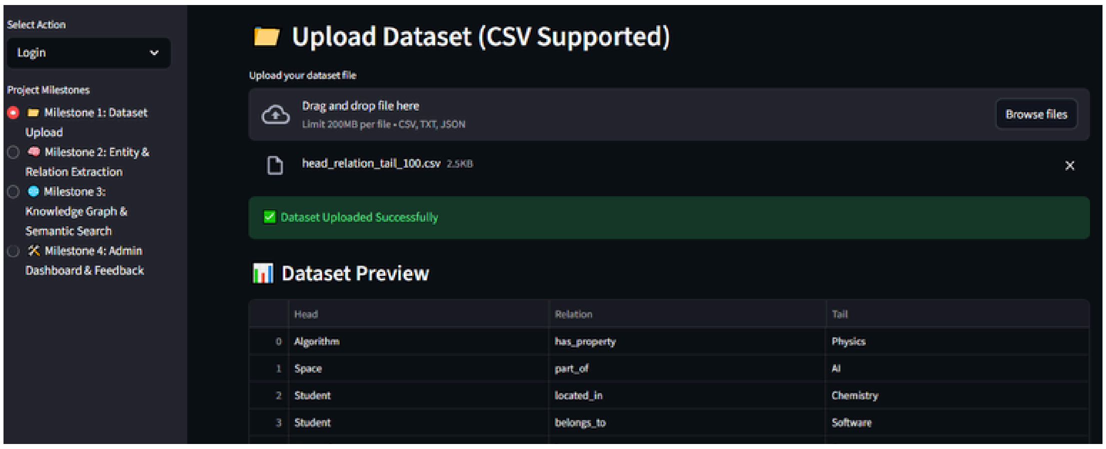
🧠 Knowledge Graph Explorer

🔐 Login / Register Page

Username

Password

Login



Select Action

Login

Project Milestones

● 📁 Milestone 1: Dataset Upload

○ 🧠 Milestone 2: Entity & Relation Extraction

○ 🌐 Milestone 3: Knowledge Graph & Semantic Search

○ 🛠 Milestone 4: Admin Dashboard & Feedback

📁 Upload Dataset (CSV Supported)

Upload your dataset file

Drag and drop file here

Limit 200MB per file • CSV, TXT, JSON

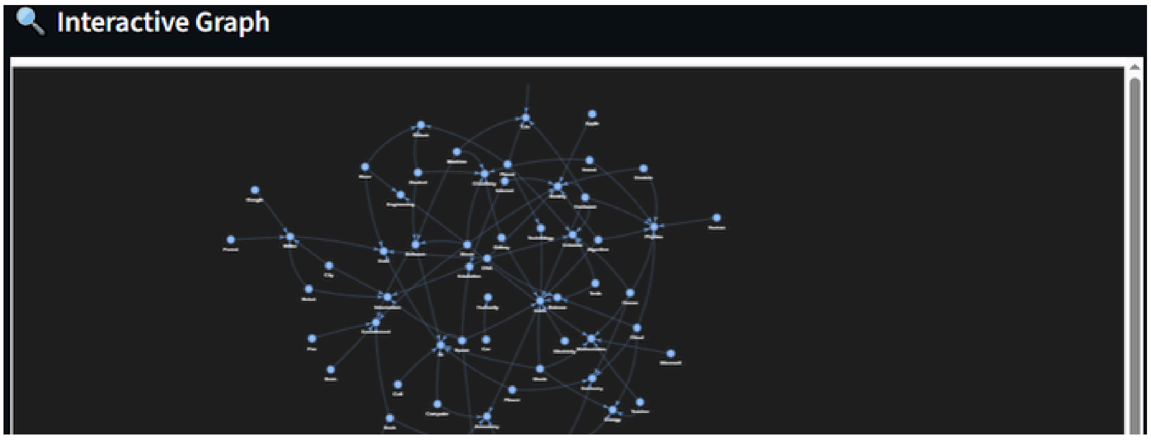
Browse files

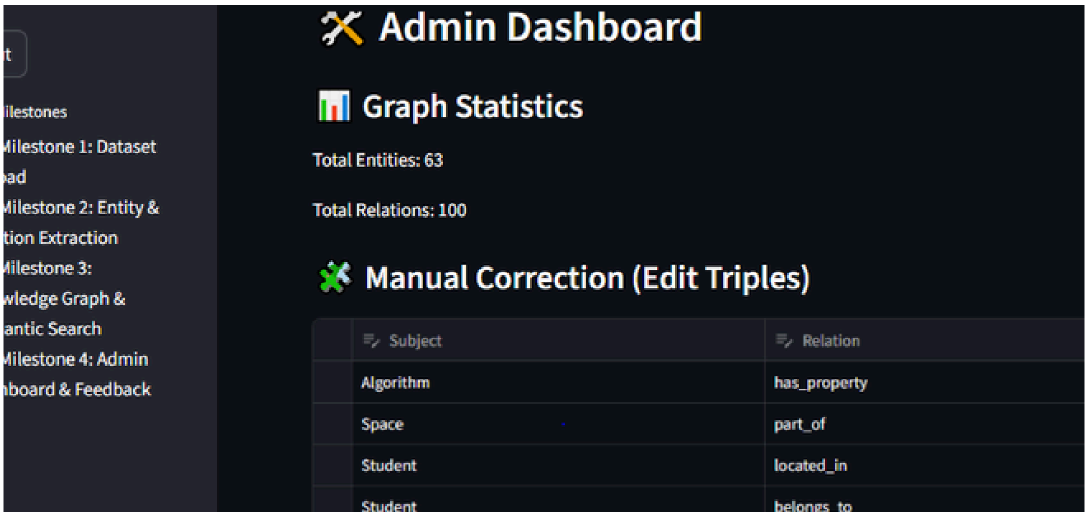
head_relation_tail_100.csv 2.5KB

✅ Dataset Uploaded Successfully

📊 Dataset Preview

	Head	Relation	Tail
0	Algorithm	has_property	Physics
1	Space	part_of	AI
2	Student	located_in	Chemistry
3	Student	belongs_to	Software





Admin Dashboard

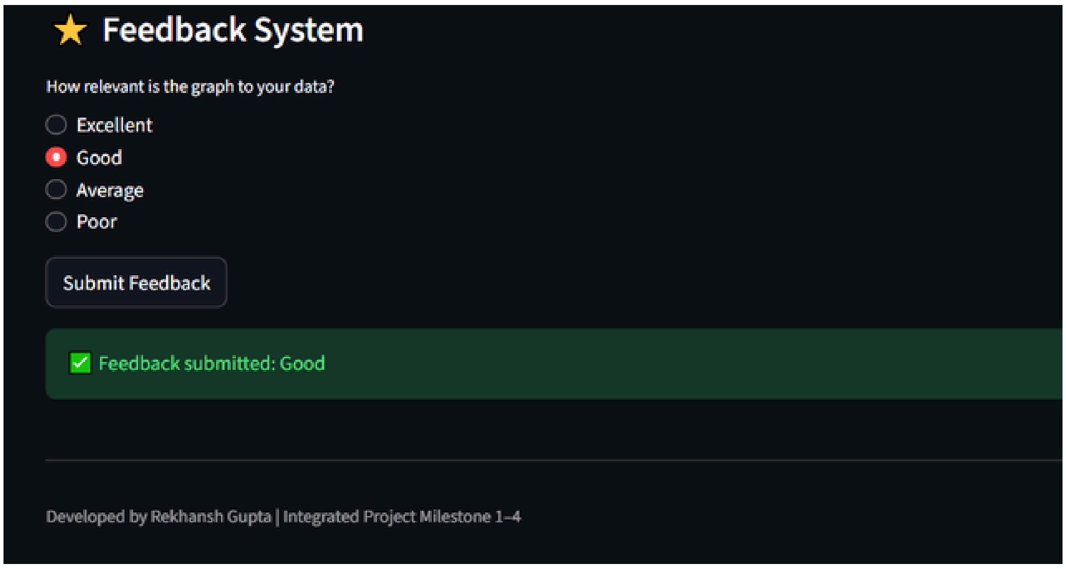
Graph Statistics

Total Entities: 63

Total Relations: 100

Manual Correction (Edit Triples)

Subject	Relation
Algorithm	has_property
Space	part_of
Student	located_in
Student	belongs_to



Feedback System

How relevant is the graph to your data?

Excellent [✓ Good] Average Poor

✓ Feedback submitted: Good

Developed by Rekhansh Gupta | Integrated Project Milestone 1–4

7.Conclusion :

This project demonstrates the development of a comprehensive, intelligent knowledge management system capable of transforming unstructured textual data into an interactive and meaningful knowledge graph. It integrates Natural Language Processing (NLP), semantic search, visualization, and summarization into a unified Streamlit-based application that can run efficiently on both local and cloud environments.

Through the combination of triple extraction, semantic similarity models, and transformer-based Q&A pipelines, the system enables users to gain deeper insights from text and navigate complex information effortlessly. The use of spaCy for linguistic analysis and SentenceTransformer for semantic embeddings provides high accuracy in entity-relation extraction and contextual similarity matching.

The addition of an Admin Dashboard enhances usability by allowing data visualization, pipeline monitoring, and performance evaluation. It also supports node editing, merging, and feedback collection to refine graph accuracy and user satisfaction over time. Furthermore, deployment via Docker and ngrok makes the system scalable, portable, and easy to share or host online.

Overall, the project not only showcases advanced technical integration but also addresses real-world challenges in knowledge representation and data understanding. It serves as a practical foundation for future improvements such as automated ontology building, larger dataset integration, and multi-language support—paving the way for smarter, data-driven decision systems.