```
1.
       # utils/gemini_client.py
2.
3.
       import os
       import google.generativeai as genai
4.
5.
       from dotenv import load dotenv
6.
7.
       load_dotenv()
8.
9.
       API_KEY = os.getenv("GEMINI_API_KEY")
10.
       if not API_KEY:
11.
           raise ValueError("GEMINI_API_KEY not found in environment
12.
           variables.")
13.
14.
       genai.configure(api key=API KEY)
15.
16.
       model = genai.GenerativeModel("gemini-2.0-flash")
17.
18.
       def generate_answer(prompt: str) -> str:
19.
           try:
20.
               response = model.generate_content(
21.
                   prompt,
                   generation_config={"temperature": 0.7}
22.
23.
24.
               return response.text.strip()
           except Exception as e:
               return f"[ERROR] {str(e)}"
26.
1.
       # utils/io_utils.py
2.
3.
       import json
5.
       def load json(path: str):
           with open(path, "r", encoding="utf-8") as f:
6.
               return json.load(f)
7.
8.
9.
       def save_json(data, path: str):
           with open(path, "w", encoding="utf-8") as f:
10.
               json.dump(data, f, indent=2, ensure_ascii=False)
11.
       # main.py
1.
2.
3.
       from utils.io_utils import load_json, save_json
4.
       from utils.gemini_client import generate_answer
5.
       import re
       import os
6.
```

```
7.
       import argparse
8.
       import pandas as pd
9.
       import time
10.
11.
       def parse_args():
           parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
12.
           parser.add_argument(
13.
                "--output",
14.
15.
               type=str,
16.
                default="generate_res/dummy.json",
               help="Output file path (default: generate_res/dummy.json)"
17.
18.
19.
           parser.add_argument(
                "--task",
20.
21.
               type=str,
22.
               default="musique",
               help="Dataset type ? (default: musique)"
23.
           )
24.
25.
           return parser.parse_args()
26.
27.
28.
       def get_controller(task):
           if task == "musique":
29.
30.
               from musique_controller import MusiqueController
31.
               return MusiqueController()
           elif task == "2wiki":
               from wiki_controller import WikiController
33.
34.
                return WikiController()
35.
           else:
36.
                raise ValueError(f"Unsupported task: {task}")
37.
38.
       def main():
39.
           args = parse_args()
40.
           print("Arguments parsed:")
41.
           for arg, value in vars(args).items():
               print(f"{arg}: {value}")
42.
           print("\n")
43.
44.
45.
           controller = get_controller(args.task)
46.
47.
           # get decomposition chains
           controller.solve()
48.
49.
       if __name__ == "__main__":
50.
51.
           main()
```

```
1.
       # wiki_controller.py
2.
3.
       from utils.io utils import load json, save json
       from utils.gemini_client import generate_answer
4.
       import prompts.wiki.template as wiki template
5.
6.
       import re
7.
       import os
       import time
8.
9.
       import json
10.
       import ast
11.
       class WikiController:
12.
           def __init__(self):
13.
14.
               self.dataset = load_json('dataset/2wiki.json')
15.
           def print questions(self):
               """Prints all questions from the dataset."""
17.
               for entry in self.dataset:
18.
19.
                    print(entry['question'])
20.
           def replace_references(self, s, result_array):
21.
22.
               def replacer(match):
23.
                   index = int(match.group(1))-1 # Ambil angka dari #3 misalnya
24.
                    if 0 <= index < len(result_array):</pre>
25.
                        return result_array[index]
26.
                   else:
                        return f"<Invalid index #{index}>"
27.
28.
               # Ganti semua #angka dengan nilai yang sesuai
29.
30.
               return re.sub(r"#(\d+)", replacer, s)
31.
32.
           def get_generated_answer(self, ans):
               if "Answer:" in ans:
33.
                   process_answer = ans.split("Answer: ", 1)[1].strip()
34.
35.
               else:
36.
                    process_answer = ans.strip()
37.
               return process_answer
38.
           def clean_text(self, text):
39.
               return re.sub(r'\(.*?\)|\[.*?\]', '', text).strip()
41.
           def get references(self, qs):
42.
43.
               references = re.findall(r'#(\d+)', qs)
44.
               return [int(ref) for ref in references]
45.
           def get_references_array(self, qs, result_answers):
46.
47.
               references = result_answers[self.get_references(qs)[0]-1]
               array_ver = json.loads(references)
48.
```

```
49.
               return array_ver
50.
           def implic RAG(self, question, entry):
51.
               formatted_context_list = self.get_formatted_context(entry)
52.
               fulltext_context = ""
53.
54.
               for context in formatted_context_list:
                   fulltext_context += context
55.
56.
57.
               prompt = wiki_template.impli_rag_template.replace('{question}',
                        question).replace('{context}', fulltext_context)
               generated_ans = generate_answer(prompt)
58.
59.
               start = generated_ans.find("C1:")
60.
               extracted_output = generated_ans[start:]
62.
63.
               return extracted output
64.
65.
           def chain_processing(self, qs_lines, entry):
66.
67.
               result_answers = []
68.
               for index, qs in enumerate(qs_lines):
69.
                   # Cari yang dalam tanda kurung bulat (command)
70.
71.
                   command_match = re.search(r"\[(.*?)\]", qs)
72.
                   command = command_match.group(1) if command_match else ''
73.
                   process_answer = ""
74.
75.
                   if command == "get ent qa":
76.
77.
                        formatted_qs = self.replace_references(qs, result_answers)
78.
                        prompt = wiki_template.get_ent_qa_template.replace('{input}',
                                 formatted_qs).replace('{context}', entry['question'])
79.
                        generated_ans = generate_answer(prompt)
                        print(f"Generated answer for get_ent_qa: {generated_ans}")
80.
81.
                        process_answer = self.get_generated_answer(generated_ans)
82.
                        print(f"Processed answer for get_ent_qa: {process_answer}")
83.
                   elif command == "get atr qa":
84.
                        references = self.get_references_array(qs, result_answers)
85.
                        ans list = []
86.
                        for ref in references:
87.
                            # format and generate prompt template
88.
89.
                            formatted_qs = re.sub(r"#\d+", ref, qs)
                            cleaned_text = self.clean_text(formatted_qs)
90.
91.
                            print(f"Formatted question: {cleaned_text}")
92.
93.
                            context = self.implic_RAG(cleaned_text, entry)
                            print(f"Context: {context}\n")
94.
```

```
95.
                            prompt = wiki_template.get_atr_qa_template
                                      .replace('{question}', cleaned_text)
                                     .replace('{context}', context)
96.
97.
                            generated ans = generate answer(prompt)
98.
                            formatted_ans = self.get_generated_answer(generated_ans)
99.
                            ans_list.append(f"{ref} => {formatted_ans}")
100.
101.
                            print(ans_list)
102.
103.
                        process_answer = ans_list
104.
                   elif command == "comp_qa":
                        latest_ans = result_answers[-1]
105.
                       full_context = ""
106.
107.
                       for ans in latest ans:
108.
                            full context += f"{ans}\n"
                        prompt = wiki_template.comp_qa_template.replace('{question}',
109.
       qs).replace('{context}', full_context)
110.
                        generated_ans = generate_answer(prompt)
111.
                        process_answer = self.get_generated_answer(generated_ans)
112.
113.
                        print(f"Processed answer for comp_qa: {process_answer}")
                   elif command == "EOQ":
114.
115.
                        Break
116.
117.
                   print(f"Processing step {index+1}: {command} -> {process answer}")
118.
                    result_answers.append(process_answer)
119.
120.
121.
               return result_answers
122.
123.
           def get_formatted_context(self, entry):
124.
125.
               formatted_data = []
126.
               formatted_entry_context = json.loads(entry['context'])
127.
               for index, context in enumerate(formatted_entry_context):
                   fulltext = ""
128.
129.
                   title = context[0]
130.
                   for text in context[1]:
131.
                        fulltext += text
132.
                   formatted context text = f"{index+1}. {title} => {fulltext}\n"
133.
134.
                    formatted_data.append(formatted_context_text)
135.
136.
               return formatted data
137.
138.
           def generate_chain(self, question):
               template = wiki_template.decomp_template
139.
```

```
140.
               prompt = template.replace('{input}', question)
141.
               decomp_chain = generate_answer(prompt)
142.
               qs_lines = [line.replace("QS: ", "").strip() for line in
143.
                           decomp_chain.splitlines() if line.startswith("QS:")]
144.
               print(f"Decomposition chain generated: {qs_lines}")
145.
               return qs_lines
146.
147.
           def solve(self):
148.
149.
               output_log = []
150.
               score = 0
151.
152.
               for entry in self.dataset:
153.
154.
                   qs_lines = self.generate_chain(entry['question'])
                   output = self.chain_processing(qs_lines, entry)
155.
156.
157.
                   def check_answer(output):
158.
                        return entry['answer'].lower() == output[-1].lower() if output else
                               False
159.
160.
                   output_log.append({
161.
                        'question': entry['question'],
162.
                        'qs_lines': qs_lines,
                        "output": output,
163.
164.
                        "is_correct": check_answer(output),
165.
                   })
166.
167.
                   current_score = 1 if check_answer(output) else 0
168.
                    score += current_score
169.
                   save_json(output_log, 'generate_res/dummy.json')
170.
171.
                   time.sleep(60) # To avoid hitting rate limits
172.
173.
               print(f"final score: {score}")
174.
               return True
       # musique_controller.py
1.
2.
3.
       from utils.io_utils import load_json, save_json
4.
       from utils.gemini_client import generate_answer
       import prompts.musique.template as musique template
5.
6.
       import re
7.
       import os
8.
       import time
9.
       import json
```

```
10.
       import ast
11.
12.
       class MusiqueController:
           def __init__(self):
13.
               self.dataset = load json('dataset/musique.json')
14.
15.
           def print_questions(self):
16.
               """Prints all questions from the dataset."""
17.
               for entry in self.dataset:
18.
19.
                   print(entry['question'])
20.
21.
           def get_generated_answer(self, ans):
               if "Answer:" in ans:
22.
23.
                   process_answer = ans.split("Answer: ", 1)[1].strip()
               else:
25.
                   process_answer = ans.strip()
26.
               return process_answer
27.
28.
           def get_fullcontext (self, paragraphs):
               fulltext_context = ""
29.
30.
               for index, paragraph in enumerate(paragraphs):
                   fulltext_context += f"{paragraph['title']} =>
31.
                                        {paragraph['paragraph_text']}\n"
32.
               return fulltext_context
33.
           def implic RAG(self, question, paragraphs):
34.
35.
               fulltext_context = self.get_fullcontext(paragraphs)
37.
38.
               prompt = musique_template.impli_rag_template.replace('{question}',
                        question).replace('{context}', fulltext_context)
39.
               generated_ans = generate_answer(prompt)
40.
               start = generated_ans.find("C1:")
41.
42.
               extracted_output = generated_ans[start:]
43.
44.
               output_lines = extracted_output.splitlines()
               output cleaned = [line.split(":", 1)[-1].strip() for line in output lines]
45.
46.
47.
               # Gabungkan kembali konteks menjadi string yang bersih
               cleaned_output = "\n".join(output_cleaned)
48.
49.
50.
               return cleaned_output
51.
52.
           def chain_processing(self, qs_lines, paragraphs):
53.
               result_answers = []
54.
55.
               for index, qs in enumerate(qs lines):
```

```
process_answer = ""
56.
57.
58.
                    if index == 0:
59.
                        context = self.get_fullcontext(paragraphs)
60.
                        prompt = musique_template.starter_qa.replace('{question}',
61.
                                 qs['question']).replace('{context}', context)
62.
                        generated_ans = generate_answer(prompt)
63.
                        process_answer = self.get_generated_answer(generated_ans)
64.
65.
                   else:
66.
                        prev_ans = result_answers[-1]
                        context = self.get_fullcontext(paragraphs)
67.
68.
69.
                        formatted_qs = re.sub(r"#\d+", prev_ans, qs['question'])
                        prompt = musique_template.finisher_template.replace('{question}',
71.
                                 formatted_qs).replace('{context}', context)
72.
                        generated_ans = generate_answer(prompt)
73.
                        process_answer = self.get_generated_answer(generated_ans)
74.
75.
                    result_answers.append(process_answer)
76.
               return result_answers
77.
           def solve(self):
78.
79.
80.
               output_log = []
81.
               score = 0
82.
83.
               for entry in self.dataset:
84.
85.
                   output = self.chain_processing(entry['question_decomposition'],
       entry['paragraphs'])
                   print("processing index: ", entry['id'])
86.
87.
                    output_log.append({
88.
                        'question': entry['question'],
89.
                        'qs_lines': entry['question_decomposition'],
                        'result': output,
90.
91.
                        'expected_answer': entry['answer'],
92.
                        "is_correct": entry['answer'].lower() == output[-1].lower()
93.
                   })
94.
95.
                   # break
96.
                   save_json(output_log, 'generate_res/dummy.json')
97.
                   time.sleep(20) # To avoid hitting rate limits
98.
99.
               print(f"final score: {score}")
               return True
100.
```

```
1.
       # prompts/wiki/template.py
2.
       decomp_template = '''
3.
4.
       QC: Which film came out first, The Love Route or Engal Aasan?
5.
       QS: [get_ent_qa] Which films are being compared in the question?
6.
       QS: (foreach)[get atr qa] What is the release date of #1?
7.
       QS: [comp_qa] Which film came out first based on the release date?
8.
9.
       QC: Are Matraville Sports High School and Wabash High School both located in the
       same country?
10.
       QS: [get_ent_qa] Which schools are being compared in the question?
       QS: (foreach)[get_atr_qa] What is the location of #1?
11.
       QS: [comp_qa] Are the schools located in the same country?
12.
13.
14.
       QC: Are Alison Skipper and Diane Gilliam Fisher from the same country?
15.
       QS: [get ent qa] Who are the people being compared in the question?
       QS: (foreach)[get_atr_qa] What is the nationality of #1?
16.
17.
       QS: [comp_qa] Do they have the same nationality?
18.
19.
       QC: Do the movies Bloody Birthday and The Beckoning Silence, originate from the
       same country?
20.
       QS: [get_ent_qa] Which movies are being compared in the question?
       QS: (foreach)[get_atr_qa] What is the country of origin of #1?
21.
       QS: [comp_qa] Do the movies originate from the same country?
22.
23.
24.
       QC: Are both businesses, Vakifbank and Infopro Sdn Bhd, located in the same
       country?
25.
       QS: [get_ent_qa] Which businesses are being compared in the question?
26.
       QS: (foreach)[get atr qa] What is the location of #1?
27.
       QS: [comp_qa] Are the businesses located in the same country?
28.
       QC: Does Mukasa Mbidde have the same nationality as Erich Maas?
29.
30.
       QS: [get_ent_qa] Who are the people being compared in the question?
       QS: (foreach)[get_atr_qa] What is the nationality of #1?
31.
32.
       QS: [comp_qa] Do they have the same nationality?
33.
34.
       QC: {input}
35.
36.
       Output:
37.
       QS: <QS-1>
38.
       QS: <QS-2>
       . . . . .
40.
       QS: <QS-N>
41.
       \mathbf{r} + \mathbf{r}
42.
43.
```

## Richard Rafer Guy – 222117056

```
44.
       get_ent_qa_template = '''
45.
       QC: Are Matraville Sports High School and Wabash High School both located in the
       same country?
       Q: Which schools are being compared in the question?
46.
47.
       Answer: ["Matraville Sports High School", "Wabash High School"]
48.
       QC: Are Alison Skipper and Diane Gilliam Fisher from the same country?
49.
50.
       Q: Which people are being compared in the question?
51.
       Answer: ["Alison Skipper", "Diane Gilliam Fisher"]
52.
53.
       QC: Do the movies Bloody Birthday and The Beckoning Silence, originate from the
       same country?
54.
       Q: Which movies are being compared in the question?
       Answer: ["Bloody Birthday", "The Beckoning Silence"]
55.
56.
       QC: Are both businesses, Vakifbank and Infopro Sdn Bhd, located in the same
57.
       country?
       Q: Which businesses are being compared in the question?
58.
59.
       Answer: ["Vakifbank", "Infopro Sdn Bhd"]
60.
61.
       QC: Did the movies Pony Express (Film) and The Da Vinci Code (Film), originate from
       the same country?
       Q: Which people are being compared in the question?
62.
       Answer: ["Sam Earle", "Felix Luckeneder"]
63.
64.
65.
       QC: Are Sam Earle and Felix Luckeneder from the same country?
66.
       Q: Which locations are being compared in the question?
       Answer: ["Lesser Slave Lake", "Medeweger See"]
67.
68.
69.
       QC: Are both Lesser Slave Lake and Medeweger See located in the same country?
70.
       Q: Which films are being compared in the question?
71.
       Answer: ["Alsino And The Condor", "1922 (2017 Film)"]
72.
       QC: Are Alsino And The Condor and 1922 (2017 Film) both from the same country?
73.
74.
       Q: Which bands are being compared in the question?
       Answer: ["Cinematic Sunrise", "Kingston Falls"]
75.
76.
77.
       QC: Are the movies Carnival Of Souls and Uvanga, from the same country?
78.
       Q: Which schools are being compared in the question?
79.
       Answer: ["St. Mary High School (Rutherford, New Jersey)", "Mother Teresa High
       School"]
80.
81.
       Input:
82.
       QC: {context}
83.
       Q: {input}
84.
85.
       Output (your answer MUST be in the same format "Answer: <your answer here>", you
       MUST add "Answer: " before your answer):
```

```
86.
       Answer: <your answer here>
87.
88.
       get_atr_qa_template = '''
89.
90.
      Q: {question}
91.
      Context:
92.
      {context}
93.
94.
      Output:
95.
       Answer: <Your answer>
96.
97.
       comp_qa_template = '''
98.
99.
       QS: {question}
100.
       Context:
101.
       {context}
102.
103.
       Output (your answer MUST be in the same format "Answer: <your answer here>", you
       MUST add "Answer: " before your answer):
104.
       Answer: <your answer here>
105.
       1,1,1
106.
       impli_rag_template = '''
107.
108.
       Given the following context data and a specific question, please provide the top 3
       most relevant contexts that help answer the question. The output should be
       formatted as follows:
109.
110.
       Context:
111. C1: "..."
112. C2: "..."
113. C3: "..."
114.
115. Context Data:
116.
     {context}
117.
118.
     Question: {question}
119.
120.
     Output:
121. C1: <Best Context>
122. C2: <Second Best Context>
123. C3: <Third Best Context>
124.
       # prompts/musique/template.py
1.
2.
       starter_qa = '''
3.
4.
       Input:
       Q: {question}
5.
```

## Richard Rafer Guy – 222117056

```
Context:
6.
7.
       {context}
8.
9.
       Output (your answer MUST be in the same format "Answer: <your answer here>", you
       MUST add "Answer: " before your answer):
10.
       Answer: <your answer here>
       1.1.1
11.
12.
13.
       finisher_template = '''
14.
       Input:
       Q: {question}
15.
       Context:
16.
17.
       {context}
18.
       Output (your answer MUST be in the same format "Answer: <your answer here>", you
19.
       MUST add "Answer: " before your answer):
20.
       Answer: <your answer here>
       \mathbf{r} + \mathbf{r}
21.
22.
       impli_rag_template = '''
23.
24.
       Given the following context data and a specific question, please provide the top 3
       most relevant contexts that help answer the question. The output should be
       formatted as follows:
25.
       Context:
26.
       C1: "..."
       C2: "..."
28.
29.
       C3: "..."
30.
31.
       Context Data:
       {context}
32.
33.
       Question: {question}
34.
35.
36.
      Output:
37.
      C1: <Best Context>
38.
      C2: <Second Best Context>
39.
      C3: <Third Best Context>
40.
```