```python
# utils/gemini_client.py

import os
import google.generativeai as genai
from dotenv import load_dotenv

load_dotenv()

API_KEY = os.getenv("GEMINI_API_KEY")
if not API_KEY:
        raise ValueError("GEMINI_API_KEY not found in environment variables.")
        genai.configure(api_key=API_KEY)

model = genai.GenerativeModel("gemini-2.0-flash")

def generate_answer(prompt: str) -> str:
    try:
        response = model.generate_content(
            prompt,
            generation_config={"temperature": 0.7}
        )
        return response.text.strip()
    except Exception as e:
        return f"[ERROR] {str(e)}"
```

```python
# utils/io_utils.py

import json

def load_json(path: str):
        with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

def save_json(data, path: str):
    with open(path, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
```

```python
# main.py - setup

from utils.io_utils import load_json, save_json
from utils.gemini_client import generate_answer
import re
from prompts.templates import basic_math_template, plan_and_solve_template
import os
import argparse
import time
```

```
    INPUT_FILE = "dataset/gsm8k.json"
    OUTPUT_FILE = "generate_res/gsm8k.json"

    # Mapping teknik ke fungsi template
    from prompts.templates import (
        basic_math_template,
        plan_and_solve_template,
        plan_and_solve_fewshot_template,
        aqua_plan_and_solve_template,
        aqua_plan_and_solve_fewshot_template,
        lastletter_plan_and_solve,
        lastletter_plan_and_solve_fewshot
    )

    TEMPLATE_MAP = {
        "basic": basic_math_template,
        "plan-and-solve": plan_and_solve_template,
        "plan-and-solve-fewshot": plan_and_solve_fewshot_template,
        "aqua-plan-and-solve": aqua_plan_and_solve_template,
        "aqua-plan-and-solve-fewshot": aqua_plan_and_solve_fewshot_template,
        "lastletter-plan-and-solve": lastletter_plan_and_solve,
        "lastletter-plan-and-solve-fewshot": lastletter_plan_and_solve_fewshot,
    }

    def parse_args():
        parser = argparse.ArgumentParser(description="Run prompting with Gemini API.")
        parser.add_argument(
            "--technique",
            choices=TEMPLATE_MAP.keys(),
            default="basic",
            help="Prompting technique to use (default: basic)"
        )
        parser.add_argument(
            "--output",
            type=str,
            default="generate_res/dummy.json",
            help="Output file path (default: generate_res/gsm8k.json)"
        )
        parser.add_argument(
            "--dataset-type",
            choices=["gsm8k", "aqua", "lastletter"],
            required=True,
            help="Dataset type: gsm8k or aqua"
        )
        parser.add_argument(
            "--dataset",
            type=str,
```

```
            default="dataset/gsm8k.json",
            help="Dataset file path (default: dataset/gsm8k.json)"
        )

        return parser.parse_args()
```

```
    # Main.py - function main
    def main():
        args = parse_args()
        technique = args.technique
        dataset_type = args.dataset_type
        prompt_fn = TEMPLATE_MAP[technique]
        dataset = load_json(args.dataset)

        os.makedirs(os.path.dirname(args.output), exist_ok=True)

        if dataset_type == "gsm8k":
            results, correct_count = process_gsm8k(dataset, prompt_fn, technique)
        elif dataset_type == "aqua":
            results, correct_count = process_aqua(dataset, prompt_fn, technique)
        elif dataset_type == "lastletter":
            results, correct_count = process_lastletter(dataset, prompt_fn, technique)
        else:
            raise ValueError("Unsupported dataset type")

        accuracy = correct_count / len(results) * 100
        print(f"\n☑ Accuracy: {correct_count}/{len(results)} correct →
             {accuracy:.2f}%")
        save_json(results, args.output)

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

```
    # main.py - processing lastletter dataset
    def process_lastletter(dataset, prompt_fn, technique):
        results = []
        correct_count = 0

        for i, entry in enumerate(dataset):
            question = entry["question"]
            expected_answer = entry["answer"]

            prompt = prompt_fn(question)
            print(f"[{i+1}] LastLetter: {question[:60]}...")

            model_answer = generate_answer(prompt)
```

```
14.
15.             if "Solve:" in model_answer:
16.                 plan_part, solve_part = model_answer.split("Solve:", 1)
17.                 reasoning = plan_part.strip() + "\nSolve:\n" + solve_part.strip()
18.                 full_text = solve_part.strip()
19.             else:
20.                 reasoning = model_answer.strip()
21.                 full_text = model_answer.strip()
22.
23.             match = re.search(r"Answer:\s*([a-zA-Z]+)", full_text)
24.             final_answer = match.group(1).lower() if match else None
25.
26.             is_correct = final_answer == expected_answer if final_answer else False
27.             if is_correct:
28.                 correct_count += 1
29.
30.             if technique == "plan-and-solve-fewshot" or technique == "lastletter-plan-
                                and-solve-fewshot":
31.                 plan_match = re.search(r"Plan:\s*(.*?)Solve:", model_answer, re.DOTALL)
32.                 solve_match = re.search(r"Solve:\s*(.*?)Answer:", model_answer,
                                 re.DOTALL)
33.
34.                 results.append({
35.                     "question": question,
36.                     "expected_answer": expected_answer,
37.                     "plan": plan_match.group(1).strip() if plan_match else "",
38.                     "solve": solve_match.group(1).strip() if solve_match else "",
39.                     "generated_answer": final_answer,
40.                     "correct": is_correct
41.                 })
42.             else:
43.                 results.append({
44.                     "question": question,
45.                     "expected_answer": expected_answer,
46.                     "reasoning": reasoning,
47.                     "generated_answer": final_answer,
48.                     "correct": is_correct
49.                 })
50.
51.             if (i+1) % 15 == 0:
52.                 print(f"Delaying for 1 minute after {i+1} iterations...")
53.                 time.sleep(60)
54.
55.         return results, correct_count
```

```
1.      # main.py - processing gsm8k dataset
2.      def process_gsm8k(dataset, prompt_fn, technique):
```

```
        results = []
        correct_count = 0

        for i, entry in enumerate(dataset):
            question = entry["question"]
            expected_answer = entry["answer"]

            prompt = prompt_fn(question)
            print(f"[{i+1}] GSM8K: {question[:60]}...")

            model_answer = generate_answer(prompt)

            if "Solve:" in model_answer:
                plan_part, solve_part = model_answer.split("Solve:", 1)
                reasoning = plan_part.strip() + "\nSolve:\n" + solve_part.strip()
                full_text = solve_part.strip()
            else:
                reasoning = model_answer.strip()
                full_text = model_answer.strip()

            match = re.search(r"Answer:\s*([-+]?\d*\.\d+|\d+)", full_text)
            final_answer = float(match.group(1)) if match else None

            is_correct = (round(final_answer, 2) == round(expected_answer, 2)) if
                        final_answer is not None else False
            if is_correct:
                correct_count += 1

            if technique == "plan-and-solve-fewshot":
                plan_match = re.search(r"Plan:\s*(.*?)Solve:", model_answer, re.DOTALL)
                solve_match = re.search(r"Solve:\s*(.*?)Answer:", model_answer,
                              re.DOTALL)

                results.append({
                    "question": question,
                    "expected_answer": expected_answer,
                    "plan": plan_match.group(1).strip() if plan_match else "",
                    "solve": solve_match.group(1).strip() if solve_match else "",
                    "generated_answer": final_answer,
                    "correct": is_correct
                })
            else:
                results.append({
                    "question": question,
                    "expected_answer": expected_answer,
                    "reasoning": reasoning,
                    "generated_answer": final_answer,
                    "correct": is_correct
```

```
                })

            if (i+1) % 15 == 0:
                print(f"Delaying for 1 minute after {i+1} iterations...")
                time.sleep(60)

        return results, correct_count
```

```
    # main.py - processing aqua dataset
    def process_aqua(dataset, prompt_fn, technique):
        results = []
        correct_count = 0

        for i, entry in enumerate(dataset):
            question = entry["question"]
            options = entry["options"]
            expected_choice = entry["correct"]

            prompt = prompt_fn(question, options)
            print(f"[{i+1}] AQUA: {question[:60]}...")

            model_answer = generate_answer(prompt)

            # Ekstrak pilihan jawaban akhir
            choice_match = re.search(r"Answer:\s*([A-E])", model_answer)
            predicted_choice = choice_match.group(1) if choice_match else ""

            is_correct = predicted_choice == expected_choice
            if is_correct:
                correct_count += 1

            # Handle output format tergantung teknik
            if technique == "aqua-plan-and-solve":
                # Ambil seluruh reasoning sebelum "Answer:"
                reasoning_match = re.search(r"(.*)Answer:\s*[A-E]", model_answer,
                                            re.DOTALL)
                reasoning_text = reasoning_match.group(1).strip() if reasoning_match
                                           else model_answer.strip()

                results.append({
                    "question": question,
                    "options": options,
                    "plan-and-solve": reasoning_text,
                    "generated_answer": predicted_choice,
                    "expected_answer": expected_choice,
                    "correct": is_correct
                })
```

```
            else:
                # Default untuk few-shot dan lainnya
                plan_match = re.search(r"Plan:\s*(.*?)(Solve:|Answer:)", model_answer,
                             re.DOTALL | re.IGNORECASE)
                solve_match = re.search(r"Solve:\s*(.*?)(Answer:)", model_answer,
                              re.DOTALL | re.IGNORECASE)

                plan = plan_match.group(1).strip() if plan_match else ""
                solve = solve_match.group(1).strip() if solve_match else ""

                results.append({
                    "question": question,
                    "options": options,
                    "plan": plan,
                    "solve": solve,
                    "generated_answer": predicted_choice,
                    "expected_answer": expected_choice,
                    "correct": is_correct
                })

            if (i+1) % 15 == 0:
                print(f"Delaying for 1 minute after {i+1} iterations...")
                time.sleep(60)

        return results, correct_count
```

```
# prompts/templates.py

def basic_math_template(question: str) -> str:
    return f"""
You are a helpful and accurate math problem solver.

Question:
{question}

Provide only the final numerical answer.
"""

def plan_and_solve_template(question: str) -> str:
    return f"""
You are a helpful and logical math tutor. When solving a math problem, follow these
steps:

1. First create a plan with bullet points.
2. Then solve the problem step-by-step.
3. Finally, output the final answer in the format: Answer: <FINAL ANSWER MUST BE
NUMBER ONLY>.
```

```
20.
21.     Question:
22.     {question}
23.
24.     Plan:
25.     -
26.
27.     Solve:
28.     """
29.
30.     def plan_and_solve_fewshot_template(question: str) -> str:
31.         few_shot_examples = """
32.     You are a logical and helpful math tutor. For each question, follow this format:
33.
34.     Plan:
35.     - step 1
36.     - step 2
37.     ...
38.
39.     Solve:
40.     <detailed solution steps>
41.
42.     Answer: <final numeric answer>
43.
44.     ---
45.
46.     Question: Jamie buys 3 pens for $2 each and a notebook for $4. What is the total
        cost?
47.
48.     Plan:
49.     - Calculate cost of pens.
50.     - Add cost of notebook.
51.
52.     Solve:
53.     3 pens × $2 = $6
54.     $6 + $4 = $10
55.
56.     Answer: 10
57.
58.     ---
59.
60.     Question: A box contains 5 red balls, 3 green balls, and 2 blue balls. How many
        total balls are there?
61.
62.     Plan:
63.     - Add all the balls.
64.
65.     Solve:
```

```
    5 + 3 + 2 = 10

    Answer: 10

    ---

    Question: {question}

    Plan:
    -
    """
        return few_shot_examples.strip().replace("{question}", question)

    def aqua_plan_and_solve_template(question: str, options: list[str]) -> str:
        options_text = "\n".join(options)
        return f"""
    You are a helpful and logical math tutor. For the following multiple-choice math question, follow these steps:
    1. **Analyze** the question carefully and determine the key values.
    2. **Plan** how to solve the question using logical steps.
    3. **Solve** the problem step-by-step with clarity.
    4. At the end, clearly write your final answer in the format: Answer: <MUST BE FINAL OPTION LETTER ONLY>

    Question:
    {question}

    Options:
    {options_text}

    Begin your solution:

    Plan:
    -
    """.strip()

    def aqua_plan_and_solve_fewshot_template(question: str, options: list[str]) -> str:
        options_text = "\n".join(options)
        few_shot_examples = """
    You are a logical and accurate math tutor. For each multiple-choice math problem, follow this format:
    - First write a step-by-step plan
    - Then solve the question
    - Then select the correct answer using: Answer: <option letter>

    ---

```

```
    Question: Sarah has 5 pencils. She gives 2 to her friend. How many does she have left?

    Options:
    A) 1
    B) 2
    C) 3
    D) 4
    E) 5

    Plan:
    - Start with total pencils
    - Subtract the number she gave away

    Solve:
    5 - 2 = 3

    Answer: C

    ---

    Question: {question}

    Options:
    {options_text}

    Plan:
    -
    """.strip()
        return few_shot_examples.replace("{question}", question).replace("{options_text}", options_text)

    def lastletter_plan_and_solve(question: str) -> str:

        return f"""
    You are a helpful and logical assistant. When solving a word problem, follow these steps:

    1. First create a plan with bullet points.
    2. Then solve the problem step-by-step.
    3. Finally, output the final answer in the format: Answer: <FINAL ANSWER MUST BE A LOWERCASE STRING>.

    Question:
    {question}

    Plan:
    -
```

```
Solve:
""".strip()

def lastletter_plan_and_solve_fewshot(question: str) -> str:
    few_shot_examples = """
You are a logical and helpful assistant. For each string manipulation task, follow this format:

Plan:
- Break the sentence into individual words.
- Take the last letter of each word.
- Concatenate the letters.
- Convert to lowercase if needed.

Solve:
Whitney Erika Tj Benito → ["Whitney", "Erika", "Tj", "Benito"]
Last letters → ["y", "a", "j", "o"]
Concatenate → "yajo"

Answer: yajo

---

Question: Lucky Mireya Jj Kc

Plan:
- Break the sentence into individual words.
- Take the last letter of each word.
- Concatenate the letters.
- Convert to lowercase if needed.

Solve:
Lucky Mireya Jj Kc → ["Lucky", "Mireya", "Jj", "Kc"]
Last letters → ["y", "a", "j", "c"]
Concatenate → "yajc"

Answer: yajc

---

Question: Caleb Chase Eleazar Chanel

Plan:
- Break the sentence into individual words.
- Take the last letter of each word.
- Concatenate the letters.
- Convert to lowercase if needed.

```

```
    Solve:
    Caleb Chase Eleazar Chanel → ["Caleb", "Chase", "Eleazar", "Chanel"]
    Last letters → ["b", "e", "r", "l"]
    Concatenate → "berl"

    Answer: berl

    ---

    Question: {question}

    Plan:
    -
    """.strip()
        return few_shot_examples.replace("{question}", question)
```