Supermarket Bill AI Agent — Solution Documentation (HW1)

Student Name: CHAN Chun Siu

Student ID: 1155250674

This solution answers **two supported questions** over a *set of receipt images*:

- **Q1**: "How much money did I spend in total?" → sum of **final paid amounts (after discount)**
- **Q2**: "How much would I have paid without discount?" → sum of **original totals (before discount)**
- Anything else → **REJECT**

---

1) End-to-end workflow (high level)

1. **Preprocess inputs**

   Convert each local receipt image into a **Base64 Data URL** so the multimodal LLM can read it.

2. **Route the user query** (Router Chain)

   Classify the query into exactly one label: `"Q1"`, `"Q2"`, or `"REJECT"`.

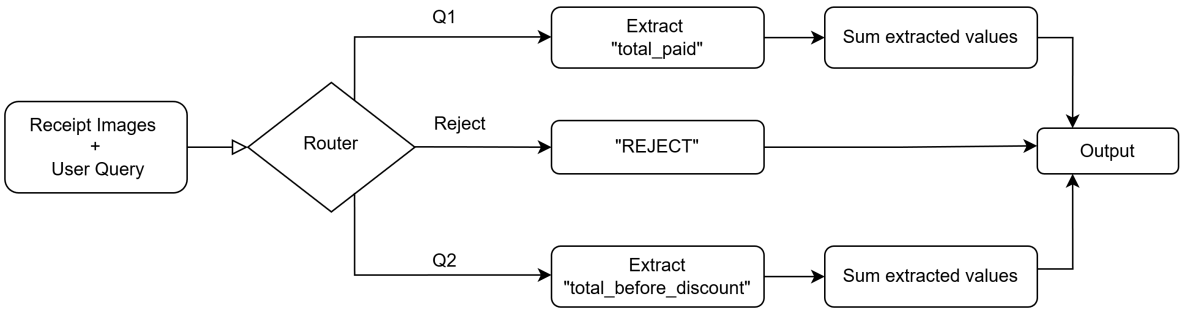
3. **Delegate to a handler** (Q1 / Q2 / Reject)

   - Q1 handler extracts `total_paid` from each receipt (batch).
   - Q2 handler extracts `total_before_discount` from each receipt (batch).
   - Reject handler returns `"REJECT"`.

4. **Aggregate results**

   Sum extracted values across all bills and return `{ items: [...], total: <sum or REJECT> }`.

5. **Coordinator** orchestrates steps 2→4 into a single runnable agent.



---

2) Input + preprocessing (images → Data URLs)

Receipts are passed to the LLM as `image_url` fields using Base64 Data URLs.

```
import base64
import mimetypes
from typing import List, Dict, Any

def image_to_base64(img_path: str) -> str:
    with open(img_path, "rb") as img_file:
        return base64.b64encode(img_file.read()).decode("utf-8")

def get_image_data_url(image_path: str) -> str:
    mime_type, _ = mimetypes.guess_type(image_path)
    if mime_type is None:
        mime_type = "image/png"
    encoded_string = image_to_base64(image_path)
    return f"data:{mime_type};base64,{encoded_string}"

def build_receipt_inputs(image_paths: List[str], user_query: str) -> Dict[str, A
ny]:
    image_data_urls = [get_image_data_url(p) for p in image_paths]
    return {
        "images": image_data_urls,  # list[str]
        "query": user_query         # str
    }
```

3) Router Chain (query classifier)

The router is a **strict classifier** that returns *exactly one token*:

- `"Q1"` / `"Q2"` / `"REJECT"`

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

router_system_prompt = """You are a strict query classifier for a receipt assist
ant.

The assistant ONLY supports:
- Q1: total money spent / total paid
- Q2: total amount without discount / original total before discount
- REJECT: If the request does not clearly fit either category.

Return EXACTLY word: "Q1", "Q2", or "REJECT"
"""

router_prompt = ChatPromptTemplate.from_messages([
    ("system", router_system_prompt),
    ("human", "{query}")
])
```

```
router_chain = router_prompt | llm | StrOutputParser()
```

4) Q1 Handler (extract final paid amount)

Q1 Receipt Extraction Chain

Reads the receipt image, extracts the total amount, and returns **JSON only**:

- `{ "total_paid": <number> }` or `{ "total_paid": null }`

```
from langchain_core.output_parsers import JsonOutputParser

q1_system_prompt = """
You are an expert at reading supermarket receipt images.

Extract the concise numbers from the receipt:
total_paid: FINAL amount paid by the customer (after discount).

Return JSON ONLY:
{"total_paid": <number>}

If cannot find them, return:
{"total_paid": null}
"""

q1_prompt = ChatPromptTemplate.from_messages([
    ("system", q1_system_prompt),
    ("human", [
        {"type": "text", "text": "{query}"},
        {"type": "image_url", "image_url": {"url": "{image}"}}
    ])
])

q1_receipt_chain = q1_prompt | llm | JsonOutputParser()
```

Q1 Handler (batch + aggregate)

Runs extraction over all receipts, then sums totals.

```
def q1_handler(payload):
    """
    payload = {"images": [...], "query": "..."}
    Return: {"items": [{"total_paid": ...}, ...], "total": float}
    """
    parsed = q1_receipt_chain.batch(
        [{"image": img, "query": payload["query"]} for img in payload["image
s"]],
        config={"max_concurrency": 10}
    )

    items, total = [], 0.0
```

```
    for r in parsed:
        v = r.get("total_paid")
        items.append({"total_paid": v})
        if v is not None:
            total += v

    return {"items": items, "total": float(total)}
```

5) Q2 Handler (extract total before discount)

Q2 Receipt Extraction Chain

Reads the receipt image and returns **JSON only**:

- `{ "total_before_discount": <number> }` or `{ "total_before_discount": null }`

```
q2_system_prompt = """
You are an expert at reading supermarket receipt images.

Extract the concise numbers from the receipt:
total_before_discount: amount the customer would pay WITHOUT discount.

Return JSON ONLY:
{"total_before_discount": <number>}

If cannot find them, return:
{"total_before_discount": null}
"""

q2_prompt = ChatPromptTemplate.from_messages([
    ("system", q2_system_prompt),
    ("human", [
        {"type": "text", "text": "{query}"},
        {"type": "image_url", "image_url": {"url": "{image}"}}
    ])
])

q2_receipt_chain = q2_prompt | llm | JsonOutputParser()
```

Q2 Handler (batch + aggregate)

```
def q2_handler(payload):
    """
    payload = {"images": [...], "query": "..."}
    Return: {"items": [{"total_before_discount": ...}, ...], "total": float}
    """
    parsed = q2_receipt_chain.batch(
        [{"image": img, "query": payload["query"]} for img in payload["image
s"]],
        config={"max_concurrency": 10}
    )
```

```
    items, total = [], 0.0
    for r in parsed:
        v = r.get("total_before_discount")
        items.append({"total_before_discount": v})
        if v is not None:
            total += v

    return {"items": items, "total": float(total)}
```

6) Reject Handler (out-of-scope queries)

If the question is not Q1/Q2 (e.g., “What supermarket names are these?”), return `"REJECT"`.

```
def reject_handler(payload):
    return {"items": [], "total": "REJECT"}
```

7) Coordinator (router + delegation + unified output)

The coordinator:

1. Uses the router to produce `decision`
2. Branches into Q1/Q2/REJECT
3. Returns only the final `output`

```
from langchain_core.runnables import (
    RunnableLambda,
    RunnablePassthrough,
    RunnableBranch,
)

branches = {
    "Q1": RunnablePassthrough.assign(output=lambda x: q1_handler(x["input"])),
    "Q2": RunnablePassthrough.assign(output=lambda x: q2_handler(x["input"])),
    "REJECT": RunnablePassthrough.assign(output=lambda x: reject_handler(x["inpu
t"])),
}

delegation_branch = RunnableBranch(
    (lambda x: x["decision"].strip().upper() == "Q1", branches["Q1"]),
    (lambda x: x["decision"].strip().upper() == "Q2", branches["Q2"]),
    branches["REJECT"],  # default
)

coordinator_agent = {
    "decision": (RunnableLambda(lambda x: {"query": x["query"]}) | router_chai
n),
    "input": RunnablePassthrough()
} | delegation_branch | (lambda x: x["output"])
```