```
# app/agent.py
import os
import json
from collections.abc import AsyncGenerator
from typing import Any, Optional
import httpx
from openai import OpenAI
from pydantic import BaseModel


# --- 外部API: Frankfurter --------------
def get_exchange_rate(
    currency_from: str = "USD",
    currency_to: str = "EUR",
    currency_date: str = "latest",
):
    """Frankfurter API から為替レートを取得"""
    try:
        resp = httpx.get(
            f"https://api.frankfurter.app/{currency_date}",
            params={"from": currency_from, "to": currency_to},
            timeout=10.0,
        )
        resp.raise_for_status()
        data = resp.json()
        if "rates" not in data:
            return {"error": "Invalid API response format."}
        return data
    except httpx.HTTPError as e:
        return {"error": f"API request failed: {e}"}
    except ValueError:
        return {"error": "Invalid JSON response from API."}


# --- 互換のための型（内部用） ----------
class ResponseFormat(BaseModel):
    status: str = "input_required"
    message: str = ""


class CurrencyAgent:
    """CurrencyAgent """

    SYSTEM_INSTRUCTION = (
        "You are a specialized assistant for currency conversions. "
        "Only help with currency exchange rate questions. "
        "Always extract parameters via the provided tools."
    )

    SUPPORTED_CONTENT_TYPES = ["text", "text/plain"]

    def __init__(self):
        self.client = OpenAI(
            api_key=os.getenv("API_KEY", "EMPTY"),
            base_url=os.getenv("TOOL_LLM_URL", None),
        )
        self.model = os.getenv("TOOL_LLM_NAME", "gpt-4o-mini")
        self.timezone = os.getenv("A2A_TIMEZONE", "Asia/Tokyo")
        self.context_id2query = {}

    # ========================
    # LLM 1: パラメータ抽出
    # ========================
    def _extract_params(self, query: str):
        tools = [
            {
                "type": "function",
                "function": {
                    "name": "extract_currency_query",
                    "description": (
                        "Extract currency parameters from user input.¥n"
                        "- currency_from: ISO-4217 3-letter uppercase "
                        "(e.g., USD)¥n"
                        "- currency_to: ISO-4217 3-letter uppercase "
                        "(e.g., EUR)¥n"
                        "- currency_date_raw: free-form date text or "
                        "'latest'¥n"
                        "If a field is missing, return an empty string."
                    ),
                    "parameters": {
                        "type": "object",
                        "properties": {
                            "currency_from": {"type": "string"},
                            "currency_to": {"type": "string"},
                            "currency_date_raw": {
                                "type": "string",
                                "description":
                                "Free-form date expression or 'latest'."
```

```
                    },
                },
                "required":
                ["currency_from", "currency_to", "currency_date_raw"],
            },
        },
    },
]

messages = [
    {"role": "system", "content": self.SYSTEM_INSTRUCTION},
    {"role": "user", "content": query.strip()},
]

resp = self.client.chat.completions.create(
    model=self.model,
    messages=messages,
    temperature=0,
    tools=tools,
    tool_choice={"type": "function",
                 "function": {"name": "extract_currency_query"}},
)

if not resp.choices:
    return None, None, None, "No response from LLM."
msg = resp.choices[0].message
if not msg.tool_calls:
    return None, None, None, "LLM did not call the extraction tool."

args_raw = msg.tool_calls[0].function.arguments
try:
    args = (
        json.loads(args_raw)
        if isinstance(args_raw, str) else (args_raw or {})
    )
except Exception:
    args = {}

c_from = (args.get("currency_from") or "").strip().upper() or None
c_to = (args.get("currency_to") or "").strip().upper() or None
date_raw = (args.get("currency_date_raw") or "").strip()
if not date_raw:
    date_raw = "latest"

return c_from, c_to, date_raw, None

# ====================================
# LLM 2: 日付の自然言語 → Frankfurter形式
# ====================================
def _normalize_date(self, date_raw: str) -> tuple[str, Optional[str]]:
    """
    モデルに自然言語日付を Frankfurter API 用の単一日付へ正規化させる。
    - 入力が 'latest' → そのまま返す
    - そうでなければ、'YYYY-MM-DD' を必ず返す（どうしても不確実なときのみ 'latest'）
    """
    import json
    import zoneinfo
    from datetime import datetime

    if date_raw.lower() == "latest":
        return "latest", None

    # 現在日時（JSTなど）を明示して相対日時を解釈させやすくする
    tz = zoneinfo.ZoneInfo(self.timezone)
    now = datetime.now(tz)
    today_str = now.strftime("%Y-%m-%d")
    weekday = ["Mon", "Tue", "Wed", "Thu",
               "Fri", "Sat", "Sun"][now.weekday()]

    # モデルが「出力」を引数として埋める function schema
    tools = [
        {
            "type": "function",
            "function": {
                "name": "resolve_date_for_frankfurter",
                "description": (
                    "Choose a SINGLE calendar day for "
                    "the Frankfurter API. "
                    "Return it in ISO 'YYYY-MM-DD'. Only return 'latest' "
                    "when the text explicitly means "
                    "the most recent available rate or "
                    "when the date truly cannot be determined. "
                    "Prefer a concrete past calendar day when possible. "
                    "Honor the provided timezone and 'today'."
                ),
                "parameters": {
                    "type": "object",
```

```
                "properties": {
                    "normalized_date": {
                        "type": "string",
                        "description":
                        "The resolved date for Frankfurter "
                        "in 'YYYY-MM-DD' or 'latest'."
                    },
                    "confidence": {
                        "type": "number",
                        "description":
                        "0.0-1.0 subjective confidence of the mapping",
                        "minimum": 0.0, "maximum": 1.0
                    },
                    "policy_applied": {
                        "type": "string",
                        "description":
                        "A short note of the disambiguation rule "
                        "you applied."
                    }
                },
                "required": ["normalized_date"]
            },
        },
    }
]
# 強い指示＋少数ショット例で「latest」濫用を抑制
system = (
    "You are a precise date normalizer for currency queries.¥n"
    "Rules:¥n"
    "1) Output exactly one calendar day in 'YYYY-MM-DD'.¥n"
    "2) Only return 'latest' when the user literally means 'latest' "
    "or the date cannot be determined.¥n"
    "3) Prefer a concrete, most likely intended past date if text is "
    "relative (e.g., 'last Friday').¥n"
    "4) Respect the timezone and 'today' provided.¥n"
    "5) If the text is a month/quarter/range, pick the most plausible "
    "single representative business day "
    "(e.g., the last calendar day of that period; if weekend/holiday "
    "ambiguity, pick the nearest prior weekday)."
)

few_shot_examples = (
    f"Today (timezone={self.timezone}) is {today_str} ({weekday}). "
    "Examples:¥n"
    "- 'last Friday' -> pick the Friday of the previous week "
    "in this timezone.¥n"
    "- '2024/6/1' or 'June 1, 2024' -> 2024-06-01.¥n"
    "- '2024-06' -> choose 2024-06-30 (or nearest prior weekday "
    "if rule 5 applies).¥n"
    "- 'end of last month' -> choose the last calendar day of "
    "the previous month.¥n"
    "- 'quarter 2 of 2024' -> choose 2024-06-30 "
    "(or prior weekday if needed).¥n"
    "- 'latest' -> latest.¥n"
    "Return via the function call ONLY."
)

user = (
    "Normalize this free-form date text for Frankfurter.¥n"
    f"Timezone: {self.timezone}¥n"
    f"Today: {today_str} ({weekday})¥n"
    f"Text: {date_raw}"
)

messages = [
    {"role": "system", "content": system},
    {"role": "system", "content": few_shot_examples},
    {"role": "user", "content": user},
]

resp = self.client.chat.completions.create(
    model=self.model,
    messages=messages,
    temperature=0,
    tools=tools,
    tool_choice={"type": "function",
                 "function": {"name": "resolve_date_for_frankfurter"}},
)

if not resp.choices or not resp.choices[0].message.tool_calls:
    # 一度は失敗し得るので、強制リトライ（latest 回避ルール明示）
    messages.insert(1, {"role": "system",
                        "content":
                        "Do NOT default to 'latest' "
                        "unless strictly required."})
    resp = self.client.chat.completions.create(
        model=self.model,
```

```
        messages=messages,
        temperature=0,
        tools=tools,
        tool_choice={"type": "function",
                     "function":
                     {"name": "resolve_date_for_frankfurter"}},
    )
    if not resp.choices or not resp.choices[0].message.tool_calls:
        return "latest", "LLM could not normalize the date; "
        "using 'latest'."

args_raw = resp.choices[0].message.tool_calls[0].function.arguments
try:
    args = json.loads(args_raw) ¥
        if isinstance(args_raw, str) else (args_raw or {})
except Exception:
    return "latest", "LLM returned invalid arguments; using 'latest'."

iso = (args.get("normalized_date") or "").strip()
if not iso:
    return "latest", "LLM did not provide a normalized date; "
    "using 'latest'."

# 軽い表層バリデーション (YYYY-MM-DD or 'latest')
if iso.lower() == "latest":
    # ユーザが 'latest' と言ってないのに latest を返した場合、もう一度だけ強制的に具体日を求める
    if date_raw.lower() != "latest":
        messages.insert(1, {"role": "system",
                            "content": "User did NOT say 'latest'. "
                            "Choose a concrete date."})
        resp2 = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            temperature=0,
            tools=tools,
            tool_choice={"type": "function",
                         "function":
                         {"name": "resolve_date_for_frankfurter"}},
        )
        if resp2.choices and resp2.choices[0].message.tool_calls:
            args2_raw = (
                resp2.choices[0].
                message.tool_calls[0].
                function.arguments
                )
            try:
                args2 = (
                    json.loads(args2_raw)
                    if isinstance(args2_raw, str)
                    else (args2_raw or {})
                )
                iso2 = (args2.get("normalized_date") or "").strip()
                if iso2 and iso2.lower() != "latest":
                    return iso2, None
            except Exception:
                pass
    # どうしても具体日にできないなら latest
    return "latest", None

# 'YYYY-MM-DD' の形式確認
import re as _re
if not _re.match(r"^¥d{4}-¥d{2}-¥d{2}$", iso):
    return "latest", "LLM returned a non-ISO date; using 'latest'."
    return iso, None

# =================
# ストリーミング応答
# =================
async def stream(self, query_in: str,
                 context_id: str) -> AsyncGenerator[dict[str, Any], None]:
    # 解析フェーズ
    yield {
        "is_task_complete": False,
        "require_user_input": False,
        "content": "入力を解析しています…",
    }

    # 簡易的なコンテキストの維持
    if context_id not in self.context_id2query:
        self.context_id2query[context_id] = ""
    self.context_id2query[context_id] += query_in + "¥n"
    query = self.context_id2query[context_id]

    c_from, c_to, date_raw, parse_err = self._extract_params(query)
    if parse_err:
        yield {
            "is_task_complete": False,
```

```
        "require_user_input": True,
        "content": f"入力解析に失敗しました: {parse_err}¥n"
                   "例: `USD EUR`"
                   "/ `usd jpy 2024-06-01` "
                   "/ `米ドルをユーロに 先週の金曜`",
    }
    return

missing = []
if not c_from:
    missing.append("通貨 (変換元, 3文字コード) ")
if not c_to:
    missing.append("通貨 (変換先, 3文字コード) ")

if missing:
    yield {
        "is_task_complete": False,
        "require_user_input": True,
        "content": (
            "為替レートを取得するために次の情報が必要です : "
            f"{' と '.join(missing)}。¥n"
            "例: `USD EUR` や `usd jpy 先週の金曜`"
        ),
    }
    return

if c_from == c_to:
    yield {
        "is_task_complete": True,
        "require_user_input": False,
        "content": f"{c_from} と {c_to} は同一通貨です。為替レートは常に 1.0 です。",
    }
    return

# 日付正規化 (モデル解釈)
yield {
    "is_task_complete": False,
    "require_user_input": False,
    "content": f"日付を解釈しています… (指定: {date_raw or 'latest'}) ",
}
date_norm, date_warn = self._normalize_date(date_raw or "latest")

# レート取得
yield {
    "is_task_complete": False,
    "require_user_input": False,
    "content": f"為替レートを取得中… ({c_from} → {c_to}, 日付: {date_norm}) ",
}

data = get_exchange_rate(c_from, c_to, date_norm)

# 整形
yield {
    "is_task_complete": False,
    "require_user_input": False,
    "content": "結果を整形しています…",
}

if "error" in data:
    msg = f"取得に失敗しました : {data['error']}¥n"
    if date_warn:
        msg += f"補足: {date_warn}¥n"
    msg += "通貨コード (例: USD EUR) と、必要なら自然言語で日付を指定して再実行してください。"
    yield {
        "is_task_complete": False,
        "require_user_input": True,
        "content": msg,
    }
    return

rate = data.get("rates", {}).get(c_to)
if rate is None:
    yield {
        "is_task_complete": False,
        "require_user_input": True,
        "content": "指定の通貨コードが不正か、レートが取得できませんでした。¥n"
                   "例: `USD EUR 2024-06-01` や `USD EUR last Friday`。",
    }
    return

date_used = data.get("date", date_norm)
inv = 1.0 / rate if rate else None

lines = [
    f"【為替レート】{c_from} → {c_to}",
    f"- 日付: {date_used}",
    f"- 1 {c_from} = {rate:.6f} {c_to}",
]
```

```
if inv:
    lines.append(f"- 1 {c_to} = {inv:.6f} {c_from}")
if date_warn:
    lines.append(f"¥n注: {date_warn}")

yield {
    "is_task_complete": True,
    "require_user_input": False,
    "content": "¥n".join(lines),
}
```