

```

001 import os
002 import json
003 from collections.abc import AsyncGenerator
004 from typing import Any, Optional
005 import httpx
006 from openai import OpenAI
007 from pydantic import BaseModel
008
009
010 # --- 外部API: Frankfurter -----
011 def get_exchange_rate(
012     currency_from: str = "USD",
013     currency_to: str = "EUR",
014     currency_date: str = "latest",
015 ):
016     """Frankfurter API から為替レートを取得"""
017     try:
018         resp = httpx.get(
019             f"https://api.frankfurter.app/{currency_date}",
020             params={"from": currency_from, "to": currency_to},
021             timeout=10.0,
022         )
023         resp.raise_for_status()
024         data = resp.json()
025         if "rates" not in data:
026             return {"error": "Invalid API response format."}
027         return data
028     except httpx.HTTPError as e:
029         return {"error": f"API request failed: {e}"}
030     except ValueError:
031         return {"error": "Invalid JSON response from API."}
032
033
034 # --- 互換のための型 (内部用) -----
035 class ResponseFormat(BaseModel):
036     status: str = "input_required"
037     message: str = ""
038
039
040 class CurrencyAgent:
041     """CurrencyAgent """
042
043     SYSTEM_INSTRUCTION = (
044         "You are a specialized assistant for currency conversions. "
045         "Only help with currency exchange rate questions. "
046         "Always extract parameters via the provided tools."
047     )
048
049     SUPPORTED_CONTENT_TYPES = ["text", "text/plain"]
050
051     def __init__(self):
052         self.client = OpenAI(
053             api_key=os.getenv("API_KEY", "EMPTY"),
054             base_url=os.getenv("TOOL_LLM_URL", None),
055         )
056         self.model = os.getenv("TOOL_LLM_NAME", "gpt-4o-mini")
057         self.timezone = os.getenv("A2A_TIMEZONE", "Asia/Tokyo")
058         self.context_id2query = {}
059
060     # =====
061     # LLM 1: パラメータ抽出
062     # =====
063     def _extract_params(self, query: str):
064         tools = [
065             {
066                 "type": "function",
067                 "function": {
068                     "name": "extract_currency_query",
069                     "description": (
070                         "Extract currency parameters from user input.¥n"
071                         "- currency_from: ISO-4217 3-letter uppercase "
072                         "(e.g., USD)¥n"
073                         "- currency_to: ISO-4217 3-letter uppercase "
074                         "(e.g., EUR)¥n"
075                         "- currency_date_raw: free-form date text or "
076                         "'latest'¥n"
077                         "If a field is missing, return an empty string."
078                     ),
079                     "parameters": {
080                         "type": "object",
081                         "properties": {
082                             "currency_from": {"type": "string"},
083                             "currency_to": {"type": "string"},
084                             "currency_date_raw": {
085                                 "type": "string",
086                                 "description":
087                                     "Free-form date expression or 'latest'."
088                             },
089                         },
090                     },
091                 },
092             },
093         ]

```

list3.txt

```
089         },
090         "required":
091         ["currency_from", "currency_to", "currency_date_raw"],
092     },
093     },
094 ],
095 ]
096
097 messages = [
098     {"role": "system", "content": self.SYSTEM_INSTRUCTION},
099     {"role": "user", "content": query.strip()},
100 ]
101
102 resp = self.client.chat.completions.create(
103     model=self.model,
104     messages=messages,
105     temperature=0,
106     tools=tools,
107     tool_choice={"type": "function",
108                 "function": {"name": "extract_currency_query"}},
109 )
110
111 if not resp.choices:
112     return None, None, None, "No response from LLM."
113
114 msg = resp.choices[0].message
115 if not msg.tool_calls:
116     return None, None, None, "LLM did not call the extraction tool."
117
118 args_raw = msg.tool_calls[0].function.arguments
119 try:
120     args = (
121         json.loads(args_raw)
122         if isinstance(args_raw, str) else (args_raw or {})
123     )
124 except Exception:
125     args = {}
126
127 c_from = (args.get("currency_from") or "").strip().upper() or None
128 c_to = (args.get("currency_to") or "").strip().upper() or None
129 date_raw = (args.get("currency_date_raw") or "").strip()
130 if not date_raw:
131     date_raw = "latest"
132
133 return c_from, c_to, date_raw, None
134
135 # =====
136 # LLM 2: 日付の自然言語 → Frankfurter形式
137 # =====
138 def _normalize_date(self, date_raw: str) -> tuple[str, Optional[str]]:
139     """
140     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。
141     - 入力が 'latest' → そのまま返す
142     - 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'）
143     """
144     import json
145     import zoneinfo
146     from datetime import datetime
147
148     if date_raw.lower() == "latest":
149         return "latest", None
150
151     # 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
152     tz = zoneinfo.ZoneInfo(self.timezone)
153     now = datetime.now(tz)
154     today_str = now.strftime("%Y-%m-%d")
155     weekday = ["Mon", "Tue", "Wed", "Thu",
156              "Fri", "Sat", "Sun"][now.weekday()]
157
158     # モデルが「出力」を引数として埋める function schema
159     tools = [
160         {
161             "type": "function",
162             "function": {
163                 "name": "resolve_date_for_frankfurter",
164                 "description": (
165                     "Choose a SINGLE calendar day for "
166                     "the Frankfurter API. "
167                     "Return it in ISO 'YYYY-MM-DD'. Only return 'latest' "
168                     "when the text explicitly means "
169                     "the most recent available rate or "
170                     "when the date truly cannot be determined. "
171                     "Prefer a concrete past calendar day when possible. "
172                     "Honor the provided timezone and 'today'."
173                 ),
174                 "parameters": {
175                     "type": "object",
176                     "properties": {
```

```

list3.txt
177     "normalized_date": {
178         "type": "string",
179         "description":
180         "The resolved date for Frankfurter "
181         "in 'YYYY-MM-DD' or 'latest'."
182     },
183     "confidence": {
184         "type": "number",
185         "description":
186         "0.0-1.0 subjective confidence of the mapping",
187         "minimum": 0.0, "maximum": 1.0
188     },
189     "policy_applied": {
190         "type": "string",
191         "description":
192         "A short note of the disambiguation rule "
193         "you applied."
194     }
195 },
196     "required": ["normalized_date"]
197 },
198 },
199 ],
200 ]
201
202 # 強い指示+少数ショット例で「latest」濫用を抑制
203 system = (
204     "You are a precise date normalizer for currency queries.¥n"
205     "Rules:¥n"
206     "1) Output exactly one calendar day in 'YYYY-MM-DD'.¥n"
207     "2) Only return 'latest' when the user literally means 'latest' "
208     "or the date cannot be determined.¥n"
209     "3) Prefer a concrete, most likely intended past date if text is "
210     "relative (e.g., 'last Friday').¥n"
211     "4) Respect the timezone and 'today' provided.¥n"
212     "5) If the text is a month/quarter/range, pick the most plausible "
213     "single representative business day "
214     "(e.g., the last calendar day of that period; if weekend/holiday "
215     "ambiguity, pick the nearest prior weekday).¥n"
216 )
217
218 few_shot_examples = (
219     f"Today (timezone={self.timezone}) is {today_str} ({weekday}). ¥n"
220     "Examples:¥n"
221     "- 'last Friday' -> pick the Friday of the previous week "
222     "in this timezone.¥n"
223     "- '2024/6/1' or 'June 1, 2024' -> 2024-06-01.¥n"
224     "- '2024-06' -> choose 2024-06-30 (or nearest prior weekday "
225     "if rule 5 applies).¥n"
226     "- 'end of last month' -> choose the last calendar day of "
227     "the previous month.¥n"
228     "- 'quarter 2 of 2024' -> choose 2024-06-30 "
229     "(or prior weekday if needed).¥n"
230     "- 'latest' -> latest.¥n"
231     "Return via the function call ONLY."
232 )
233
234 user = (
235     "Normalize this free-form date text for Frankfurter.¥n"
236     f"Timezone: {self.timezone}¥n"
237     f"Today: {today_str} ({weekday})¥n"
238     f"Text: {date_raw}"
239 )
240
241 messages = [
242     {"role": "system", "content": system},
243     {"role": "system", "content": few_shot_examples},
244     {"role": "user", "content": user},
245 ]
246
247 resp = self.client.chat.completions.create(
248     model=self.model,
249     messages=messages,
250     temperature=0,
251     tools=tools,
252     tool_choice={"type": "function",
253                 "function": {"name": "resolve_date_for_frankfurter"}},
254 )
255
256 if not resp.choices or not resp.choices[0].message.tool_calls:
257     # 一度は失敗し得るので、強制リトライ (latest 回避ルール明示)
258     messages.insert(1, {"role": "system",
259                         "content":
260                         "Do NOT default to 'latest' "
261                         "unless strictly required."})
262     resp = self.client.chat.completions.create(
263         model=self.model,
264         messages=messages,

```

list3.txt

```
265         temperature=0,
266         tools=tools,
267         tool_choice={"type": "function",
268                     "function":
269                       {"name": "resolve_date_for_frankfurter"}},
270     )
271     if not resp.choices or not resp.choices[0].message.tool_calls:
272         return "latest", "LLM could not normalize the date;"
273         "using 'latest'."
274
275     args_raw = resp.choices[0].message.tool_calls[0].function.arguments
276     try:
277         args = json.loads(args_raw) ¥
278         if isinstance(args_raw, str) else (args_raw or {})
279     except Exception:
280         return "latest", "LLM returned invalid arguments; using 'latest'."
281
282     iso = (args.get("normalized_date") or "").strip()
283     if not iso:
284         return "latest", "LLM did not provide a normalized date;"
285         "using 'latest'."
286
287     # 軽い表層バリデーション (YYYY-MM-DD or 'latest')
288     if iso.lower() == "latest":
289         # ユーザが 'latest' と言っていないのに latest を返した場合、もう一度だけ強制的に具体日を求める
290         if date_raw.lower() != "latest":
291             messages.insert(1, {"role": "system",
292                                 "content": "User did NOT say 'latest'. "
293                                 "Choose a concrete date."})
294         resp2 = self.client.chat.completions.create(
295             model=self.model,
296             messages=messages,
297             temperature=0,
298             tools=tools,
299             tool_choice={"type": "function",
300                         "function":
301                           {"name": "resolve_date_for_frankfurter"}},
302         )
303         if resp2.choices and resp2.choices[0].message.tool_calls:
304             args2_raw = (
305                 resp2.choices[0].
306                 message.tool_calls[0].
307                 function.arguments
308             )
309             try:
310                 args2 = (
311                     json.loads(args2_raw)
312                     if isinstance(args2_raw, str)
313                     else (args2_raw or {})
314                 )
315                 iso2 = (args2.get("normalized_date") or "").strip()
316                 if iso2 and iso2.lower() != "latest":
317                     return iso2, None
318             except Exception:
319                 pass
320             # どうしても具体日にできないなら latest
321             return "latest", None
322
323     # 'YYYY-MM-DD' の形式確認
324     import re as _re
325     if not _re.match(r"^\d{4}-\d{2}-\d{2}$", iso):
326         return "latest", "LLM returned a non-ISO date; using 'latest'."
327
328     return iso, None
329
330 # =====
331 # ストリーミング応答
332 # =====
333 async def stream(self, query_in: str,
334                 context_id: str) -> AsyncGenerator[dict[str, Any], None]:
335     # 解析フェーズ
336     yield {
337         "is_task_complete": False,
338         "require_user_input": False,
339         "content": "入力を解析しています...",
340     }
341
342     # 簡易的なコンテキストの維持
343     if context_id not in self.context_id2query:
344         self.context_id2query[context_id] = ""
345     self.context_id2query[context_id] += query_in + "\n"
346     query = self.context_id2query[context_id]
347
348     c_from, c_to, date_raw, parse_err = self._extract_params(query)
349     if parse_err:
350         yield {
351             "is_task_complete": False,
352             "require_user_input": True,
```

```

list3.txt
353     "content": f"入力解析に失敗しました: {parse_err}¥n"
354     "例: `USD EUR`",
355     "`/ `usd jpy 2024-06-01` ",
356     "`/ `米ドルをユーロに 先週の金曜`",
357     }
358     return
359
360 missing = []
361 if not c_from:
362     missing.append("通貨 (変換元, 3文字コード) ")
363 if not c_to:
364     missing.append("通貨 (変換先, 3文字コード) ")
365
366 if missing:
367     yield {
368         "is_task_complete": False,
369         "require_user_input": True,
370         "content": (
371             "為替レートを取得するために次の情報が必要です: "
372             f"{ 'と'.join(missing)}. ¥n"
373             "例: `USD EUR` や `usd jpy 先週の金曜`",
374         ),
375     }
376     return
377
378 if c_from == c_to:
379     yield {
380         "is_task_complete": True,
381         "require_user_input": False,
382         "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。",
383     }
384     return
385
386 # 日付正規化 (モデル解釈)
387 yield {
388     "is_task_complete": False,
389     "require_user_input": False,
390     "content": f"日付を解釈しています… (指定: {date_raw or 'latest'}) ",
391 }
392 date_norm, date_warn = self._normalize_date(date_raw or "latest")
393
394 # レート取得
395 yield {
396     "is_task_complete": False,
397     "require_user_input": False,
398     "content": f"為替レートを取得中… ({c_from} → {c_to}, 日付: {date_norm})",
399 }
400
401 data = get_exchange_rate(c_from, c_to, date_norm)
402
403 # 整形
404 yield {
405     "is_task_complete": False,
406     "require_user_input": False,
407     "content": "結果を整形しています…",
408 }
409
410 if "error" in data:
411     msg = f"取得に失敗しました: {data['error']}¥n"
412     if date_warn:
413         msg += f"補足: {date_warn}¥n"
414     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。"
415     yield {
416         "is_task_complete": False,
417         "require_user_input": True,
418         "content": msg,
419     }
420     return
421
422 rate = data.get("rates", []).get(c_to)
423 if rate is None:
424     yield {
425         "is_task_complete": False,
426         "require_user_input": True,
427         "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"
428             "例: `USD EUR 2024-06-01` や `USD EUR last Friday`。",
429     }
430     return
431
432 date_used = data.get("date", date_norm)
433 inv = 1.0 / rate if rate else None
434
435 lines = [
436     f"【為替レート】 {c_from} → {c_to}",
437     f"- 日付: {date_used}",
438     f"- 1 {c_from} = {rate:.6f} {c_to}",
439 ]
440 if inv:

```

```
list3.txt
441     lines.append(f"- 1 {c_to} = {inv:.6f} {c_from}")
442 if date_warn:
443     lines.append(f"注: {date_warn}")
444
445 yield {
446     "is_task_complete": True,
447     "require_user_input": False,
448     "content": "\n".join(lines),
449 }
```