ChatGPT

Kế hoạch tái cấu trúc (refactor) workflow bot du lịch

Mục tiêu chính là đảm bảo mỗi trường hợp sử dụng (GET_PLACE_INFO, GET_WEATHER, GET_DISTANCE, SEARCH_NEARBY, SEARCH_TOUR, SEARCH_HOTEL) được xử lý chính xác và hiệu quả, đồng thời tối ưu hóa hiệu năng, giảm thiểu số node dư thừa, chuẩn hóa luồng logic (ưu tiên dùng DB nội bộ trước khi gọi API bên ngoài), và tuân thủ đúng hợp đồng đầu ra/UI. Kế hoạch gồm **phân đoạn pipeline theo từng use case** với đề xuất chi tiết cho từng node (giữ, gộp, bỏ), cải tiến cache, logging, xử lý lỗi, và tuân thủ schema `poi_override_postgres.sql`. Cuối cùng có checklist chất lượng production.

GET_PLACE_INFO

- **Ý nghĩa:** Trả về thông tin một địa điểm (tên, mô tả, giá vé, giờ mở cửa...) theo từ khóa người dùng.
- **Luồng hiện tại:**
- **Fn_CheckRequiredSlots** (xác định intent = GET_PLACE_INFO và cần slot `place_name`), nếu thiếu thì qua **Respond_Clarify**.
- **Switch_Pipeline (placeName)** → **Fn_ResolvePlaceName** (tách tên địa điểm từ câu hỏi).
- **HTTP_SearchPlaceInfo** (gọi SerpAPI Google Search để lấy thông tin địa điểm).
- **Fn_FormatPlaceInfo** (định dạng kết quả: tên, mô tả, giá vé, giờ, địa chỉ).

- **Fn_BuildAIInput** → **AI_Response_Guard** → **Respond_Final** (tạo câu trả lời cho người dùng).

- **Vấn đề và cải tiến:**

- *DB ưu tiên:* Có thể tận dụng bảng `poi_override` để chuyển alias (VD: "hoan_kiem" → "Hồ Hoàn Kiếm") và cung cấp **canonical_name** (tên chính thức). Đề xuất thêm một bước **Fn_NormalizeAlias (giống GET_WEATHER)** trước **HTTP_SearchPlaceInfo** để tìm `normalized_alias`. Nếu DB có match (`DB_POI_Lookup`), ta lấy luôn `canonical_name` làm từ khóa tìm kiếm thay vì dùng chuỗi thô. Tuy nhiên, *lưu ý:* DB không chứa mô tả/gía vé, nên vẫn cần gọi API. Việc dùng DB chủ yếu tăng độ chính xác và tính ổn định đầu vào.
- *Giảm node:* Luồng hiện tại chỉ có 4 node chính (resolve, HTTP, format, buildAI) là hợp lý. Có thể **gộp** một số logic nhỏ vào code node cho gọn: ví dụ hợp nhất việc trích xuất giá và giờ vào Fn_FormatPlaceInfo (đã làm). Không nên tách nhỏ hơn nữa tránh tạo nhiều node xử lý chuỗi.
- *Xử lý lỗi:* Nếu **Fn_ResolvePlaceName** không tìm được tên (mặc định Fn_CheckRequiredSlots đã đảm bảo có khớp regex), thì nên trả lời rõ ràng (ví dụ "Xin lỗi, tôi không hiểu địa điểm bạn muốn xem thông tin"). Hiện đã có **IF_NeedClarification** trước đó để hỏi lại khi thiếu slot. Nên đảm bảo **HTTP_SearchPlaceInfo** khi trả về không kết quả thì gửi phản hồi nhẹ nhàng (thông qua AI, hoặc fallback message), không để lộ lỗi nội bộ.
- *Cache:* Thông tin địa điểm (mô tả, giờ mở, giá vé) thường ít thay đổi. Có thể cân nhắc caching kết quả API trong cơ sở dữ liệu hoặc bộ nhớ (Redis) theo **canonical_name**. Khi có cache hit, sẽ trả về trực tiếp mà không cần gọi API bên ngoài (theo nguyên tắc "lưu trữ thông tin tĩnh thay vì gọi lại nhiều lần" [1] [2]). Nếu thêm chức năng này, cần tạo bảng *place_info_cache* (JSON lưu kết quả) hoặc tính năng Redis.

- *Kết cấu output:* Đảm bảo `Fn_FormatPlaceInfo` tạo ra định dạng hợp đồng: một object chứa `name, description, ticket_price, opening_hours, address`. Các field này khớp với UI yêu cầu, không để dư field thừa hoặc thiếu trường.

Node	Chức năng	Đề xuất
Fn_ResolvePlaceName	Trích xuất tên địa điểm từ câu hỏi người dùng	**Giữ:** Có thể bổ sung normalize alias với DB (nếu alias không có trong regex).
HTTP_SearchPlaceInfo	Gọi SerpAPI (Google Search) lấy thông tin địa điểm	**Giữ:** Thêm kiểm tra lỗi/timeout, giữ cấu hình phù hợp.
Fn_FormatPlaceInfo	Xử lý kết quả, trích mô tả, giá vé, giờ mở cửa, địa chỉ	**Giữ:** Đảm bảo trích đúng trường, thêm logging kết quả.
Fn_BuildAIInput	Tạo input cho AI (kết hợp data intent + kết quả)	**Giữ:** Thực hiện cuối cùng chung cho tất cả pipeline.

GET_DISTANCE

- **Ý nghĩa:** Tính quãng đường (và/hoặc thời gian) từ điểm A đến điểm B.
- **Luồng hiện tại:**
- **Fn_CheckRequiredSlots** (intent=GET_DISTANCE, regex "từ ... đến ..." xác định origin/destination). Thiếu slot sẽ đưa ra **Respond_Clarify**.
- **Switch_Pipeline (distance)** → **Fn_ExtractDistanceEntities** (phân tích câu hỏi, xác định `distanceOrigin` và `distanceDestination`).
- **HTTP_GeocodeOrigin** (địa điểm gốc → lat/lng).
- **Fn_PrepareDestGeocode** (chuẩn bị dữ liệu đến).
- **HTTP_GeocodeDestination** (đích đến → lat/lng).
- **Fn_BuildDistanceORS** (tạo request OpenRouteService cho matrix hoặc route).
- **HTTP_ORS_DistanceCalc** (tính toán qua ORS).
- **Fn_FormatDistance** (định dạng kết quả (km, thời gian)).

- **Fn_BuildAIInput** → **Respond_Final**.

- **Vấn đề và cải tiến:**

- *DB ưu tiên:* Có thể thêm bước sử dụng **poi_override** cho origin/destination: trước khi gọi geocode, normalize origin/dest rồi truy vấn DB (intent "GET_DISTANCE"). Nếu tìm thấy (ví dụ: "tràng_an"), ta lấy kinh/vĩ độ từ DB, đánh dấu `geocoded: true` và bỏ qua HTTP_GeocodeOrigin/Gendpoint tương ứng (tương tự GET_WEATHER). Cần thêm node **Fn_NormalizeAlias (gốc, đích)** và **DB_POI_Lookup** cho từng địa điểm. Điều này tăng độ tin cậy và tốc độ (không phải gọi Google API nếu DB đã có).
- *Xử lý lỗi:* Hiện đã có IF cho geocode gốc (IF_GeocodeSuccess) nhưng chưa có cho geocode đích. Nên thêm IF tương tự cho **HTTP_GeocodeDestination**: nếu không lấy được tọa độ, trả lời người dùng lỗi rõ (ví dụ: "Không tìm thấy địa điểm đến, xin thử lại."). Đồng thời cần xác minh cả origin/dest hợp lệ trước khi gọi ORS.
- *Cache distance:* Để tăng hiệu năng, lưu cache các kết quả tính toán khoảng cách phức tạp. Ví dụ, tạo bảng **distance_cache(origin,destination, distance_km, duration, updated_at)**. Trước khi gọi ORS, kiểm tra cache (ví dụ key là "lat1,lng1 -> lat2,lng2"). Nếu có và còn tươi (ví dụ 1 tuần), trả về ngay. Điều này **lưu trữ kết quả tính đường thường dùng** để tránh gọi ORS nhiều lần [1] [2]. Cần câu truy vấn SQL nhanh (đảm bảo index trên cặp origin/dest).

- *Giảm node:* Các bước hiện tại rất rõ ràng. Có thể *gộp* một số hàm nhỏ: ví dụ, tích hợp **Fn_PrepareDestGeocode** vào **Fn_BuildDistanceORS** để bớt một node Code, hoặc tích hợp khâu tạo request ORS vào Format hoặc dùng chung một code node. Tuy nhiên, việc chia rõ ràng giúp bảo trì tốt hơn. Quan trọng là đảm bảo mỗi node code có nhiệm vụ rõ ràng.
- *Quan sát/log:* Ghi log đầu vào (origin, destination) và đầu ra (kết quả distance) trong các code node (console.log) để dễ debug. Đảm bảo lỗi mạng hoặc ORS timeout được ghi log và gửi đến cơ chế fallback (ví dụ cảnh báo qua email bằng workflow lỗi).
- *Đầu ra:* `Fn_FormatDistance` cần trả về chính xác `{ distance_km, duration }` (hoặc các trường tương ứng) theo hợp đồng UI.

Node	Chức năng	Đề xuất
Fn_ExtractDistanceEntities	Phân tích câu, tách điểm gốc và đích	**Giữ:** Bổ sung normalize alias & DB lookup cho origin/destination.
HTTP_GeocodeOrigin	Lấy lat/lng từ điểm gốc	**Giữ:** Thêm timeout, xử lý lại nếu coord không hợp lệ.
Fn_PrepareDestGeocode	Chuẩn bị thông tin/đầu vào để geocode đích	**Gộp:** Có thể hợp nhất vào Fn_BuildDistanceORS để bớt 1 node.
HTTP_GeocodeDestination	Lấy lat/lng từ điểm đích	**Giữ:** Thêm IF success/fail và log lỗi.
Fn_BuildDistanceORS	Tạo request cho ORS (tính đường)	**Giữ:** Thêm logic cache ORS (tương tác với bảng distance_cache).
HTTP_ORS_DistanceCalc	Gọi ORS trả về khoảng cách, thời gian	**Giữ:** Đảm bảo cấu hình API key, handle error.
Fn_FormatDistance	Định dạng kết quả (km, thời gian)	**Giữ:** Kiểm tra phép tính (chuyển từ m sang km, phút), log kết quả.

GET_WEATHER

- **Ý nghĩa:** Cung cấp thông tin thời tiết hiện tại hoặc dự báo (theo ngày mai, cuối tuần, số ngày) cho một địa điểm cụ thể.
- **Luồng hiện tại:**
- **Fn_CheckRequiredSlots** (intent=GET_WEATHER, yêu cầu `location`).
- **Switch_Pipeline (geocode)** → **Fn_ExtractLocation** → *DB hoặc Geocode* (qua Fn_NormalizeAlias → DB_POI_Lookup → HTTP_Geocode → Fn_ParseGeocode). Kết quả: tọa độ và `locationDisplay`.
- **Switch_RouteToAPI (intent=GET_WEATHER)** → **Fn_NormalizeWeatherTime** (xác định `weatherMode` hiện tại/forecast và `forecastDays`).
- **Switch_WeatherMode**: nếu forecast → **HTTP_GetWeatherForecast**, else **HTTP_GetWeather** (WeatherAPI).
- **Fn_FormatWeather** hoặc **Fn_FormatWeatherForecast** (định dạng kết quả: nhiệt độ, mô tả thời tiết...).

- **Fn_BuildAIInput** → **Respond_Final**.

- **Vấn đề và cải tiến:**

- *DB ưu tiên:* Quy trình `Fn_UseDBResult` cho GET_WEATHER đã có sẵn: nếu `normalized_alias` khớp DB (ví dụ "trang_an"), dùng kinh/vĩ độ DB và bypass Geocode (source='poi_override_db'). Giữ nguyên. Thêm index nếu cần trên `aliases` / `intent_support` đã có, đảm bảo truy vấn nhanh.
- *Tối ưu cache:* **Thời tiết** là ứng dụng lý tưởng cho caching. Tạo bảng **weather_cache** (ví dụ: key="lat,lng|date|mode", dữ liệu JSON API, timestamp). Trước khi gọi WeatherAPI, kiểm tra cache (nếu data còn mới, ví dụ trong 1 giờ). Nếu có, dùng dữ liệu cached, tránh gọi API nhiều lần [1] [3]. Khi lưu, thêm thời gian cập nhật để tuỳ chỉnh TTL. Cách này lưu thông tin thời tiết tĩnh ngắn hạn, giảm truy vấn mạng.
- *Giảm node:* Có thể gộp **Fn_NormalizeWeatherTime** với **Switch_WeatherMode** thành một node code để xác định loại yêu cầu và chọn URL. Tuy nhiên, tách ra giúp rõ ràng: Fn_NormalizeWeatherTime trả ra `weatherMode`, Switch_WeatherMode chỉ chọn output. Có thể giữ. Các bước format (Weather/Forecast) riêng biệt cũng hợp lý vì khác cấu trúc data. Không nên gộp chúng để tránh code quá phức tạp.
- *Xử lý lỗi:* Nếu WeatherAPI trả lỗi (ví dụ quá tải, hay không tìm ra địa điểm), cần bắt lỗi: thêm kiểm tra status và trả về thông báo thích hợp (ví dụ "Không lấy được dữ liệu thời tiết"). Không để lộ chi tiết lỗi. Có thể dùng cơ chế **AI_Response_Guard** để giữ an toàn nội dung.
- *Đầu ra:* `Fn_FormatWeather` và `Fn_FormatWeatherForecast` phải trả ra rõ ràng các trường như `current_temp`, `condition`, `forecast` (mảng các ngày) theo hợp đồng. Ví dụ `forecastday` từ API cần chuyển thành danh sách `{ date, maxtemp, mintemp, condition }`.
- *Performance:* Đảm bảo chỉ gọi forecast nếu người dùng yêu cầu (Fn_NormalizeWeatherTime quyết định). Thông thường, đếm ngày dự báo tối đa 7 hoặc 14 (theo API giới hạn). Tránh gửi yêu cầu ngoài phạm vi.

Node	Chức năng	Đề xuất
Fn_ExtractLocation	Lấy `location` từ input (đã normalize, db lookup)	**Giữ:** Bổ sung cache cho lat/lng nếu cần.
Fn_NormalizeWeatherTime	Xác định mode (current/ forecast) và số ngày dự báo	**Giữ:** Gộp với Switch_WeatherMode nếu phù hợp.
Switch_WeatherMode	Chuyển hướng đến current hoặc forecast	**Giữ:** Logic đơn giản, k cần thay đổi.
HTTP_GetWeather	Lấy dữ liệu thời tiết hiện tại (WeatherAPI)	**Giữ:** Thêm xử lý thất bại, dùng cache trước khi gọi.
HTTP_GetWeatherForecast	Lấy dữ liệu dự báo WeatherAPI	**Giữ:** Thêm xử lý tương tự, tùy mode/time.
Fn_FormatWeather	Định dạng dữ liệu hiện tại	**Giữ:** Chuyển kết quả sang đối tượng đúng.
Fn_FormatWeatherForecast	Định dạng dữ liệu dự báo	**Giữ:** Chia nhỏ dữ liệu, có thể lặp cho mỗi ngày.

SEARCH_NEARBY

- **Ý nghĩa:** Tìm các địa điểm gần vị trí người dùng (theo loại địa điểm cụ thể).
- **Luồng hiện tại:**

- **Fn_CheckRequiredSlots** (intent=SEARCH_NEARBY, yêu cầu `user_location`). Nếu không có vị trí người dùng (GPS), qua **Respond_Clarify**.
- **Switch_Pipeline (userLocation)** → **Fn_BuildNearbySearch** (dùng `userLocation` lat/lng, tạo query SerpAPI để tìm điểm lân cận).
- **HTTP_SearchNearby** (SerpAPI Google Search, engine=places, dùng query build sẵn).
- **Fn_FormatNearby** (lọc top-k kết quả, trích tọa độ từ `local_results`).
- **Fn_BuildORSRequest** (chuẩn bị request matrix cho các địa điểm đó từ user).
- **HTTP_ORS_Matrix** (tính khoảng cách/vận tốc đến mỗi địa điểm).
- **Fn_EnrichWithDistances** (gắn thêm khoảng cách/thời gian vào `nearby_raw`).

- **Fn_BuildAIInput** → **Respond_Final**.

- **Vấn đề và cải tiến:**

- *Đơn giản hóa:* Các bước hiện tại khá rõ. Có thể **gộp** một phần logic: ví dụ, **Fn_BuildORSRequest** chỉ thu thập lat/lng từ **Fn_FormatNearby**, sau đó gọi ORS. Ta có thể kết hợp `Fn_FormatNearby` và `Fn_BuildORSRequest` thành một node để giảm 1 node, song cần code phức tạp hơn. Cân nhắc gộp nếu workflow quá dài (mục tiêu giảm độ phức tạp node [4]).
- *Xử lý lỗi:* Nếu SerpAPI trả ít kết quả hoặc thất bại, cần kiểm tra trong **Fn_FormatNearby** (thử giải thích: ví dụ "Không tìm thấy địa điểm nào xung quanh bạn"). Cũng kiểm tra kết quả **ORS** trước khi enrich. Nếu thiếu lat/lng trong kết quả SerpAPI, không gọi ORS.
- *Cache:* Thông thường `SEARCH_NEARBY` phụ thuộc vào vị trí người dùng và thời điểm, ít tái sử dụng nhiều lần. Nếu cần cache, có thể đơn giản dùng bộ nhớ tạm thời trên phiên, hoặc caching bán động (cache lần tìm gần nhất trong vài phút). Tuy nhiên, ưu tiên cho các cache quan trọng hơn (weather/distance).
- *Quan sát/log:* Log `query` tìm kiếm và kết quả trả về để debug. Đảm bảo middleware của ORS sử dụng truy vấn batch, nếu mở rộng thêm người dùng song song có thể optimize thêm bằng split batch (nếu kết quả quá nhiều).
- *Đầu ra:* `Fn_EnrichWithDistances` phải trả về danh sách `nearby` gồm `{ name, address, distance_km, ... }` đúng hợp đồng.

Node	Chức năng	Đề xuất
Fn_BuildNearbySearch	Tạo query SerpAPI từ `userLocation`	**Giữ:** Đảm bảo fallback nếu địa điểm yêu cầu không rõ.
HTTP_SearchNearby	Gọi SerpAPI Google (engine=places)	**Giữ:** Thêm timeout/catch lỗi.
Fn_FormatNearby	Chọn kết quả, trích lat/lng	**Giữ/Gộp:** Có thể gộp với Fn_BuildORSRequest để giảm node.
Fn_BuildORSRequest	Tạo yêu cầu matrix khoảng cách	**Giữ:** Thêm cache nếu áp dụng cho ORS.
HTTP_ORS_Matrix	Gọi ORS tính khoảng cách	**Giữ:** Đảm bảo định danh điểm đúng đầu vào.
Fn_EnrichWithDistances	Gắn khoảng cách vào kết quả (distance_km)	**Giữ:** Định dạng số, đơn vị, log kết quả.

SEARCH_TOUR

- **Ý nghĩa:** Tìm tour du lịch phù hợp (theo số ngày, khu vực).
- **Luồng hiện tại:**
- **Fn_CheckRequiredSlots** (intent=SEARCH_TOUR, cần `duration`). Thiếu thì **Respond_Clarify**.
- **Switch_Pipeline (textFilter)** → **Fn_ParseTourDuration** (bóc tách `duration` (số ngày) và `tourLocation` từ câu).
- **HTTP_SearchTour** (SerpAPI Google Search, engine=tourism, dùng từ khóa, duration).
- **Fn_FormatTour** (lọc các tour, định dạng thông tin).

- **Fn_BuildAIInput** → **Respond_Final**.

- **Vấn đề và cải tiến:**

- *Đơn giản hóa:* Mục đích chính của `Fn_ParseTourDuration` là tách ngày và địa điểm. Có thể gộp một phần sang **Fn_FormatTour** nếu không phức tạp. Ví dụ, sau khi gọi HTTP_SearchTour, có thể trong Fn_FormatTour phân tích `duration` nếu cần. Nhưng tách riêng giúp rõ nghĩa.
- *DB ưu tiên:* Nếu `tourLocation` nhận được (ví dụ "Hội An") khớp alias trong `poi_override`, có thể dùng lat/lng DB để đặt vị trí trong query tìm tour (nếu API hỗ trợ lọc theo tọa độ) hoặc ít nhất chuẩn hoá tên địa điểm (ví dụ đưa canonical_name). Không quá quan trọng vì SerpAPI search theo text chính.
- *Cache:* Tour thường ít có tính thời gian, nhưng không cần thiết. Tuy nhiên, nếu tour gần như tĩnh, có thể cache kết quả tương tự như địa điểm (bảng *tour_cache*). Có thể bỏ qua do ít lợi ích.
- *Đầu ra:* `Fn_FormatTour` cần trả về danh sách các tour với thông tin chính (tên tour, địa điểm, số ngày, giá, liên kết). Điều chỉnh phù hợp hợp đồng UI.

Node	Chức năng	Đề xuất
Fn_ParseTourDuration	Trích số ngày, địa điểm từ câu hỏi tour	**Giữ:** Đã thực hiện tốt chức năng slot.
HTTP_SearchTour	Gọi SerpAPI (engine=tourism)	**Giữ:** Thêm xử lý lỗi, batch nếu cần.
Fn_FormatTour	Lọc và định dạng kết quả tour	**Giữ:** Xác nhận các trường như duration, price.
Fn_BuildAIInput	Tạo input AI cuối cùng	**Giữ:** Như các pipeline khác.

SEARCH_HOTEL

- **Ý nghĩa:** Tìm khách sạn ở khu vực yêu cầu.
- **Luồng hiện tại:**
- **Fn_CheckRequiredSlots** (intent=SEARCH_HOTEL, cần `location`).
- **Switch_Pipeline (geocode)** → **Fn_ExtractLocation** → *DB hoặc Geocode* (tương tự GET_WEATHER).
- **Switch_RouteToAPI (intent=SEARCH_HOTEL)** → **HTTP_SearchHotel** (SerpAPI Google Hotels, dùng lat/lng và khoảng ngày mặc định).
- **Fn_FormatHotel** (chọn top5, định dạng tên, điểm đánh giá, giá).

- **Fn_BuildAIInput** → **Respond_Final**.

- **Vấn đề và cải tiến:**

- *DB ưu tiên:* Quy trình geocode như đã có. Nếu vị trí đầu vào là alias trong `poi_override` (VD "Hoi_An"), sẽ lấy được toạ độ nhanh mà không cần Geocode. Giữ nguyên logic Db lookup hiện có (Fn_NormalizeAlias → DB_POI_Lookup → Fn_UseDBResult) trước HTTP_Geocode.
- *Xử lý lỗi:* Nếu SerpAPI Hotels trả ít kết quả, đảm bảo **Fn_FormatHotel** vẫn hoạt động (có thể trả kết quả ít hơn hoặc cho biết "không tìm thấy khách sạn phù hợp").
- *Cache:* Có thể cân nhắc caching kết quả "top hotels cho từng khu vực" trong thời gian ngắn (vài giờ) vì giá phòng thay đổi theo thời điểm; nếu muốn tối ưu có thể bỏ qua.
- *Đầu ra:* `Fn_FormatHotel` phải tạo ra danh sách khách sạn với các trường như `name, rating, price, address`. Kiểm tra tính chính xác (`$json.properties` tồn tại hay không, `$json.cars` nếu trả về từ serpapi).
- *Giảm node:* Luồng ngắn gọn, không cần gộp thêm. Có thể merge Fn_FormatHotel và Fn_BuildAIInput, nhưng cần giữ Fn_BuildAIInput chuẩn cho consistency.

Node	Chức năng	Đề xuất
HTTP_SearchHotel	Gọi SerpAPI Google Hotels (engine=google_hotels)	**Giữ:** Thêm giới hạn và xử lý lỗi (Google rate-limit).
Fn_FormatHotel	Lọc top5 khách sạn, trích `name, rating, price, address`	**Giữ:** Xác nhận format output hợp đồng.
Fn_BuildAIInput	Xây dựng dữ liệu đầu vào AI	**Giữ:** Cuối cùng chung cho tất cả pipeline.

Đề xuất cơ sở dữ liệu (Schema)

Bảng **poi_override** hiện tại đã đầy đủ cho lookup địa điểm và intent (kiểm tra `aliases`, `intent_support`) 【*/mnt/data/poi_override_postgres.sql*】. Đề xuất: - **Không thay đổi** cấu trúc `poi_override`. Nên thêm index đã được tạo sẵn (trong file SQL) để truy vấn theo alias và intent nhanh: `idx_poi_aliases`, `idx_poi_intent`.
- **Thêm bảng cache** (nếu quyết định cài):
- **weather_cache** (ví dụ `location_key VARCHAR PK, data JSONB, fetched_at TIMESTAMP`).
- **distance_cache** (`origin TEXT, destination TEXT, distance_km NUMERIC, duration_mins INT, updated_at TIMESTAMP`, cặp (origin,destination) PK).
- Nếu có caching khách sạn/gần, tương tự.
- Các bảng cache cần thêm trigger cập nhật hoặc TTL (có thể quản lý TTL qua query).

Tóm lại, tuân thủ schema `poi_override` như hiện có, chỉ bổ sung thêm bảng mới cho cache nếu cần.

Checklist chất lượng (Production/Staff-Engineer)

- **Performance:** Tận dụng *caching* cho các dữ liệu tĩnh (thời tiết, khoảng cách) [1] [2] ; kết hợp batch API và giảm bớt node thừa [4] . Tối giản truy vấn, tránh vòng lặp API (nên đưa data vào một node duy nhất khi có thể) [4] . Đảm bảo query DB có index phù hợp (theo schema SQL).
- **Quan sát (Observability) / Logging:** Ghi log chi tiết tại các bước quan trọng: thông tin đầu vào (intent, slot), kết quả từ API, lỗi nếu có. Ví dụ, thêm `console.log` trong code nodes và bật level debug khi test [5] . Thiết lập logging toàn bộ workflow (qua winston) để dễ dàng kiểm tra khi phát hành.
- **Xử lý fallback (Fallback Safety):** Luôn có kịch bản thay thế khi DB/API thất bại. Dùng `IF` để bắt lỗi geocode, weather, ORS, trả về câu trả lời an toàn (ví dụ yêu cầu thử lại, hoặc trả lời dựa

trên dữ liệu khác) thay vì crash. Cấu hình **Error Trigger** trong n8n để gửi cảnh báo/kịch bản xử lý khi workflow có exception [6]. Không để lộ chi tiết hệ thống ra người dùng.

- **Bao phủ trường hợp đặc biệt (Edge cases):** Đảm bảo mọi slot đều được kiểm tra (có phản hồi rõ ràng nếu thiếu thông tin). Xử lý các trường hợp không có kết quả (ví dụ, không tìm thấy địa điểm, không có tour phù hợp) bằng thông báo thân thiện. Kiểm thử các input biên (đường dài > ngưỡng ORS, thành phố lạ, ngày thời tiết quá xa, v.v.).
- **Scalability:** Thiết kế pipeline có thể mở rộng: sử dụng tính song song (parallel) ở các chỗ có thể, cân nhắc tách workflow nếu quá dài [7] [2]. Xem xét triển khai nhiều instance n8n (horizontal scaling) và cân bằng tải cho Webhook nếu lượng lớn. Tối ưu DB (index, partition nếu lớn) theo khuyến cáo của n8n docs [8].

Bằng cách tuân thủ các cải tiến trên và checklist chất lượng, workflow bot du lịch sẽ đáp ứng tốt 6 use case chính, vận hành ổn định, hiệu quả trên môi trường production. Các node được tinh gọn, logic rõ ràng và có ghi chú chú thích chức năng giúp dễ bảo trì về sau. Các đề xuất lớn về caching và fallback nhằm tối ưu performance và độ an toàn của hệ thống.

Nguồn tham khảo: Các phương pháp tối ưu n8n (trong đó có caching và giảm node dư thừa) được khuyến nghị [7] [4]; logging và error-handling cũng được nêu rõ trong tài liệu chính thức [5] [6].

---

[1] [3] [7] [8] Elevate Your Business with n8n Workflow Optimization | by Dejan Markovic, co-founder https://hypestudio.org/ | Medium
https://medium.com/@dejanmarkovic_53716/elevate-your-business-with-n8n-workflow-optimization-da8b6b28042c

[2] [4] I analysed 2,000+ n8n workflows and this is what I learned : r/n8n
https://www.reddit.com/r/n8n/comments/1l1f6n8/i_analysed_2000_n8n_workflows_and_this_is_what_i/

[5] Logging | n8n Docs
https://docs.n8n.io/hosting/logging-monitoring/logging/

[6] Error handling | n8n Docs
https://docs.n8n.io/flow-logic/error-handling/