Demo Presentation

Title: Demo Presentation (Screencast Plan and Script)
Author: [Your Name]
Date: [Date]
Video Link: [Add after recording]
Code Repository: [Add repo URL]

Goal and Audience
Demonstrate the agent, how it leverages multiple map servers, and key lessons learned.
Audience: instructors/peers evaluating functionality, design choices, and understanding.

Running Order (5â 7 Minutes)
0:00â 0:30 â Intro: Project goal and one-line pitch.
0:30â 1:30 â Architecture: Agent + provider modules overview.
1:30â 3:30 â Demo 1: Geocoding/POI search using two providers (contrast).
3:30â 5:00 â Demo 2: Routing or Overpass feature query.
5:00â 6:00 â Challenges and solutions (rate limits, normalization, caching).
6:00â 6:45 â Results: What works well; edge cases; performance.
6:45â 7:00 â Close: Repo link, how to run, next steps.

Script (Concise Talking Points)
Intro
â Hi, Iâ m [Name]. This is [Agent Name], which answers location-based queries by combining multiple map servers for better coverage and reliability.â

Architecture
â The agent normalizes responses from [Server A], [Server B], and [Server C]. It routes queries by typeâ geocoding, POI search, routingâ and reconciles conflicting results. Fallbacks and caching reduce latency and handle rate limits.â

Demo 1: Geocoding + POI Search
* Query: â Find coffee shops near [City/Neighborhood].â
* Show: Provider 1 (e.g., Nominatim) vs. Provider 2 (e.g., Mapbox/Google) results.
* Explain: Differences in counts, categories, and confidence; how the agent merges and ranks results.

Demo 2: Routing or Feature Query
* Option A: â Best walking route from [A] to [B]â via routing API.
* Option B: â List parks (amenity=park) within [bbox]â via Overpass.
* Show: Request, response snippet, and rendered map or a concise textual summary.
* Explain: Rate-limit handling, retries, and user-facing error messages.

Challenges and Solutions
* Data inconsistencies across providers â normalization, tie-breaking, and consensus.
* Quotas/rate limits â exponential backoff + caching of hot paths.
* Ambiguous addresses â multi-provider disambiguation and optional user prompts.

Demo Presentation

Results and Lessons
* Multi-provider approach improved coverage and recall (qualitatively/quantitatively).
* Vector tiles sped up map rendering; Overpass enabled structured feature extraction.
* Logging and observability helped tune retries and provider selection.

Close
â Repo: [URL]. Setup instructions are in the README. The recording link is in the submission. Next steps include adding isochrones, batch queries, and more tests.â

Live Demo Checklist
* Environment variables and API keys loaded; rate-limit headroom available.
* Demo scenarios rehearsed; fallback paths validated.
* Stable network; cache warmed for key calls.
* Terminal font large; map window ready; consistent theme.
* Copy-ready commands/queries in a notes doc.

On-Screen Flow (Example)
* Terminal: run ‘python demo.py --query "coffee shops near [City]"‘ and show logs (provider selection, retries, merged ranking).
* Browser/app: display a small map or show ranked textual results.
* Terminal: run ‘python demo.py --route --from "[A]" --to "[B]"‘ for routing.
* Editor: briefly show provider modules and agent selection logic.

Recording Tips
* 1080p video, 125â 150% UI scaling; clear mic; minimal background noise.
* Keep mouse movement deliberate; zoom windows to focus key moments.
* If a call might be slow, prewarm cache or show a recorded snippet.

Submission Notes
* Include: this PDF, code repository link, video link, and a short reflection (1â 2 paragraphs) on lessons learned and next steps.

Example: Distance Tripoli â Beirut
* Command (CLI):
* ‘python -m part2_implementation.demo_runner "Distance from Tripoli to Beirut"‘
* What happens under the hood:
* Agent detects a distance query â selects ‘ors_distance_places‘.
* Geocodes both places with OSM/Nominatim â gets ‘[lon, lat]‘ pairs.
* Calls ORS Distance/Route API with those coordinates.
* Returns a compact JSON with ‘distance_km‘ and echoed coordinates.
* Expected JSON shape (values vary by provider data):
* ‘{ "answer": "...", "tool_results": [{ "tool": "ors_distance_places", "content": { "distance_km": <float>, "origin": [<lon>,<lat>], "destination": [<lon>,<lat>] } }] }‘
* Optional environment for offline-style demo:

Demo Presentation

* Set ‘MAP_AGENT_PROVIDER=ollama‘ (if Ollama is running) or run without OpenAI keys to use heuristic fallback; output will show the tool used and JSON result directly.

Example: Hospitals in Tripoli
* Command (CLI):
* ‘python -m part2_implementation.demo_runner "Find 3 hospitals in Tripoli"‘
* What happens under the hood:
* Agent detects a POI query â selects ‘osm_search_poi‘.
* Uses OSM/Nominatim to search with ‘query="hospitals"‘, ‘city="Tripoli"‘, ‘max_count=3‘.
* Returns a list of up to 3 items with name and coordinates.
* Expected JSON shape (truncated):
* ‘{ "answer": "...", "tool_results": [{ "tool": "osm_search_poi", "content": [{ "name": "...", "lat": "...", "lon": "..." }, ...] }] }‘
* Notes:
* For best results, use a model provider (OpenAI or ‘MAP_AGENT_PROVIDER=gemini‘) so the tool args include the correct city from your prompt.
* If you run in offline/heuristic fallback, refine the prompt to be explicit (e.g., "Search POIs: hospitals, city: Tripoli, max=3") to avoid default city bias.

Components and File Roles (Part 2)
* part2_implementation/agent_sdk_app.py: Main agent. Defines tool schemas (osm_geocode, ors_route, etc.), dispatches calls to servers, caches geocodes, and routes across providers (OpenAI/Gemini/Ollama) with fallbacks.
* part2_implementation/servers/osm_server.py: OSM/Nominatim helper with async methods: geocode(place), reverse(lat, lon), search_poi(query, city, max_count). Adds optional country bias via OSM_COUNTRYCODES.
* part2_implementation/servers/ors_server.py: OpenRouteService helper with async methods: route(origin, destination, profile), distance(...), nearby(lat, lon). Requires ORS_API_KEY; parses variant response shapes; returns error dicts on failures.
* part2_implementation/mcp_base.py: Minimal MCP-style types (MCPCommand, MCPMapServer interface) used to describe server commands.
* part2_implementation/openai_client.py: Loads OPENAI_API_KEY and constructs the OpenAI client used in the OpenAI provider path.
* part2_implementation/gemini_provider.py: Direct Gemini calls and tool-calling bridge translating our tool schema to Gemini function declarations and stitching function responses.
* part2_implementation/demo_runner.py: Small CLI to run the agent with a prompt and print the JSON result.
* part2_implementation/litellm_agents_demo.py: Optional demo of OpenAI Agents SDK routed to Gemini via a local LiteLLM proxy.
* part2_implementation/map_agent.ipynb: Notebook variant of the demo (interactive exploration).

APIs and Providers Used
* OSM/Nominatim (no API key required; rate limits apply)

Demo Presentation

* Search (geocode): https://nominatim.openstreetmap.org/search?q=<query>&format=json
* Reverse geocode: https://nominatim.openstreetmap.org/reverse?lat=<lat>&lon=<lon>&format=json
* Headers: User-Agent: C5-MapAgent; Optional filter: OSM_COUNTRYCODES=lb,us (example)
* OpenRouteService (requires ORS_API_KEY in part2_implementation/.env)
* Directions: POST https://api.openrouteservice.org/v2/directions/{profile} body: {"coordinates": [[lon,lat],[lon,lat]]}
* POIs: POST https://api.openrouteservice.org/pois body includes bbox around [lon,lat]
* Headers: Authorization: <ORS_API_KEY>; Content-Type: application/json

Roles: OSM vs ORS
* OSMServer (Nominatim)
* Role: Place search, address lookup, reverse geocoding, and simple POI search by text. Great for entity names and address resolution; open and free.
* Used for: Translating user place names to coordinates; reverse-mapping coordinates to readable addresses; finding candidate POIs in a city.
* ORSServer (OpenRouteService)
* Role: Routing (turn-by-turn steps), distance/time summaries, and nearby POIs using a bounding box. Requires API key; robust routing profiles.
* Used for: Driving/walking/cycling routes; distances between coordinates or between place names (with OSM geocode pre-step); nearby discovery around a point.

Add to Live Demo (talking cues)
* â File roles: The agent is in part2_implementation/agent_sdk_app.py. It calls two server wrappers under part2_implementation/servers: osm_server.py for Nominatim (search/reverse/POI) and ors_server.py for OpenRouteService (route/distance/nearby). mcp_base.py defines simple command descriptors; openai_client.py and gemini_provider.py wire the model providers. demo_runner.py is the CLI entry.â
* â APIs: We use Nominatimâ s public endpoints for geocoding and reverse, and ORSâ s v2 Directions/POIs endpoints with an ORS_API_KEY. The agent normalizes outputs and handles errors without crashing.â
* â Flow: For â distance between A and Bâ , we auto-geocode A and B with OSM, then call ORS distance. For â route from A to Bâ , we call ORS route and show steps or a summary.â