MASIV 2025 Intern Test: "Urban Design 3D City Dashboard with LLM Querying"

Time Limit: Submit within 24 hours of receiving this brief.

Objective: Build and launch a working prototype that demonstrates your ability to independently create a robust, full-stack solution. The project will showcase your skills in backend development, data persistence, frontend visualization, and AI integration.

Task Description:

Create and launch a web-based dashboard. It must:

1. Fetch Calgary city map data for minimum **3-4 city blocks (or more)** from a public city open data API (e.g., building footprints, heights, zoning) and process it for display. An example resource can be the City of Calgary open data website.
2. Visualize the buildings in **3D** using Three.js (e.g., extruded shapes based on footprint and height data) to represent the full 3-4 block area.
3. Add interactivity: Clicking any building highlights it and shows a popup with its fetched data (e.g., address, height, zoning type, assessed property value, etc.). Data points must be available for all buildings.
4. Integrate an **LLM** to query the map:
   - ○ Add a text input field where users can type natural language queries (e.g., "highlight buildings over 100 feet" or "show commercial buildings" or “show buildings in RC-G zoning” or “show buildings less than $500,000 in value”).
   - ○ Use the LLM to interpret the query and filter the dataset, then highlight matching buildings in the 3D view.
5. Implement Project Persistence: Add a system for users to save and load their map analyses.
   - ○ User Identification: Allow a user to identify themselves. A simple username input field is sufficient; a full, secure authentication system is not required.
   - ○ Save Functionality: Create a UI element (e.g., a "Save Project" button) that allows a user to save the current set of active LLM-generated filters under a project name.
   - ○ Load Functionality: Display a list of the user's saved projects. Clicking on a project should load its filters and re-apply them to the 3D map.
6. Create a UML diagram (e.g., class diagram, sequence diagram, or both) documenting your solution's structure and flow. This should include the new user and project data models.

Requirements:

- Backend in **Python** (e.g., Flask) to fetch data and handle LLM integration.
- Frontend in **JavaScript**, **React**, and **Three.js**.
- Use a lightweight database (e.g., SQLite) to persist user and project data.
- Use a free LLM API (e.g., Hugging Face Inference API) to process queries.
- Implement **software engineering best practices**: clean code, modularity, error handling, and clear documentation in the code.
- Deliver a ZIP file with:

    - Source code.
    - A README.md with setup instructions (include how to get an LLM API key if needed).
    - A UML diagram (PDF, PNG, or hand-drawn and scanned—tool of your choice).
- **Host it using a free service and provide the link.**

Tools and Resources:

- Public APIs: OpenStreetMap, City of Calgary Open Data, or similar (use datasets with building footprints, heights, or zoning).
- LLM: Hugging Face Inference API (free tier, sign up for an API key at huggingface.co).
- Any open-source libraries you need (e.g., requests for Python API calls).
- ArcGIS can be used but not required—simulate analysis with Python processing.

Submission:

- Email the ZIP file.
- Provide a link to the publicly hosted website.
- Optional: Include a 2-3 minute video link (e.g., Loom) walking through your solution.

Note:

We are testing your ability to figure it out and deliver a well-engineered solution independently. Focus on functionality, best practices, and integrating the LLM to query the map—no “right” approach is provided.

How the LLM Integration Works as an ***Example***:

- **Example Workflow:**
    1. User enters: "highlight buildings over 100 feet."
    2. Flask sends the query to Hugging Face API with a prompt like:
       "Extract the filter from this query: {user_input}. Return a JSON object with 'attribute' (e.g., height), 'operator' (e.g., >), and 'value' (e.g., 100)."
    3. LLM returns: {"attribute": "height", "operator": ">", "value": "100"}.
    4. Backend filters the dataset (e.g., building.height > 100), sends results to frontend.
    5. Three.js highlights matching buildings (e.g., changes their color).
- **Time Feasibility:** Adding a text input and API call takes ~1-2 hours for someone familiar with Python and React.

- Fallback: If an alternative approach is taken, provide explanation and include it in UML.