**ASSIGNMENT 1**

**Each individual has to follow rules**

\*\*Objective:\*\*

Your task is to create a Python service using Flask that acts as a geometry engine. This service should be capable of performing various geometric operations on 3D Geometry.

\*\*Requirements:\*\*

1. \*\*Setup:\*\*

   - Initialize a new Python project.

   - Install Flask as a dependency.

   - Set up a virtual environment to manage dependencies.

   - Feel free to read any necessary documentation about computational geometry and algorithms.

2. \*\*Endpoints:\*\*

   - Create endpoints for the following geometric calculations:

* Given an array of 3D Points, calculate the Smallest Bounding Box that contains all the 3D Points.
* Given a 3D Mesh as an Input, rotate the mesh by X degrees along the specified axis.
  + e.g. Input: { mesh: [3D POINT], angle: 30, axis: Y}

                    Output: {mesh: [3D POINT]} which is now rotated.

* Given a 3D mesh as an input, Move the Mesh by a, b and c units along X, Y and Z axis respectively.
  + e.g. Input: { mesh: [3D POINT], x: 30, y: 20, z: 25}

                    Output: {mesh: [3D POINT]} which is now at a new location.

* Given a polygon in a 3D space represented by 3D Points, check whether the polygon is convex.
* Each endpoint should accept the required parameters via query parameters or JSON payload.
* Implement appropriate error handling for invalid input.
* In case of ambiguity, make your best assumptions and document those assumptions

3. \*\*Implementation:\*\*

   - Implement the geometric calculations using Python functions.

   - Ensure that the service returns the calculated values in the response.

   - Use appropriate HTTP methods for each endpoint (GET or POST).

   - Ensure that the service is RESTful and follows best practices.

   - Implement and reuse common data structures when they are used by multiple services

4. \*\*Documentation:\*\*

   - Include clear documentation for each endpoint, specifying the required parameters and expected output.

   - Provide instructions on how to run the service locally.

5. \*\*Testing:\*\*

   - Write unit tests to verify the correctness of the geometric calculations.

   - Test each endpoint with sample inputs to ensure proper functionality.

   - Ensure that the service handles edge cases gracefully.

6. \*\*after testing:\*\*

   - Implement additional geometric operations and make sure that the service has a configurable precision level.

   - Add authentication and authorization mechanisms if time permits.

   - Dockerize the application for easier deployment.

\*\*Submission:\*\*

- Provide the source code of your project along with any necessary configuration files.

- Create a github repository and share

- Include documentation explaining how to run the service and test the endpoints.

- Optionally, provide any additional notes or insights about your implementation choices.

\*\*Evaluation Criteria:\*\*

- Correctness of geometric calculations.

- Modular code with appropriate data structures

- DRY principles

- Clarity and completeness of documentation.

- Proper error handling and edge case management.

- Unit test coverage and effectiveness.

- Bonus points for additional features or enhancements.

\*\*Note:\*\*

- **DO NOT USE external geometry libraries that perform exactly these operations. IMPLEMENT YOUR OWN SOLUTION.**

- Focus on writing clean, maintainable, and well-documented code.

- You are encouraged to ask for clarification if any requirements are unclear.