```
                                  (unsigned int), indices, GL_STATIC_DRAW);
    // shader
    unsigned int shader = createShader(readFile("res/shaders/vertex.shader"),
readFile("res/shaders/fragment.shader"));
    glUseProgram(shader);
    int location1 = glGetUniformLocation(shader, "u_color");

    while(!glfwWindowShouldClose(window)) {
        // clear the buffer
        glClear(GL_COLOR_BUFFER_BIT);

        // sets the background color
        glClearColor(1.0f, 1.0f ,1.0f, 1.0f);

        // draw
        glUseProgram(shader);
        glUniform4f(location1, 85.0f*INV_255, 184.0f*INV_255, 237.0f*INV_255, 1.0f);
        glBindVertexArray(vertexArrayObj);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, NULL);

        // display bound buffer
```

USE CASE: ARRAY OPERATIONS

Use Case

Case Study: The client-side has limited computational power, requiring it to invoke array manipulation operations remotely on a more powerful server. The server exposes services that perform several array operations. The client sends arrays to the server to utilize the following services:

1. **Sorting an Array**: Sort the provided integer array in ascending order.
2. **Finding the Maximum Element**: Return the maximum value in the array.
3. **Finding the Minimum Element**: Return the minimum value in the array.
4. **Adding Two Arrays**: Compute the element-wise sum of two arrays.
5. **Multiplying Two Arrays**: Compute the element-wise product of two arrays.

P1 - Programming for Communication

a. **Provide the architecture of the client-server setup**

- Describe the architecture where a client with limited processing power relies on the server to perform computationally intensive array operations remotely. Consider how these operations are exposed and accessed by the client.

b. **Define the service contract**

- The protocol

c. **Implement the client-side code for the "findMax" service**

- Provide the implementation code for the client side to invoke the **findMax** service remotely. This client code should demonstrate how to call the service on the server, send an integer array as input, and handle the server's response to receive the maximum element.

P2 - Programming for Integration

Part 1: XML/SOAP Web Service Implementation

1. Server-Side Implementation:
   - Implement the ArrayManipulatorService in Java using JAX-WS.
   - The service should expose the following operations:
   - sortArray(int[] array): Sort an array of integers.
   - findMax(int[] array): Find the maximum element in an array.
   - findMin(int[] array): Find the minimum element in an array.
   - addArrays(int[] array1, int[] array2): Add two arrays element-wise.
   - multiplyArrays(int[] array1, int[] array2): Multiply two arrays element-wise.
2. WSDL Generation:
   - Generate the WSDL for your web service using wsgen.
3. Write Client Code:- Python: Write a Python client using the zeep module to interact with the SOAP
   service.
   - JavaScript: Write a JavaScript client using the soap module.
   - Java: Write a Java client by generating the stubs using wsimport.
4. Test the Clients:

- Run your webservice and test each client to ensure the array operations work correctly.

Part 2: Pseudo-RESTful Web Service Implementation

1. Server-Side Implementation:
    - Implement the ArrayManipulatorService using Spring Boot and expose the same operations as REST endpoints:
    - POST /sortArray
    - POST /findMax
    - POST /findMin
    - POST /addArrays
    - POST /multiplyArrays
2. OpenAPI/Swagger Generation:
    - Use Springdoc OpenAPI to generate the OpenAPI specification for your RESTful web service.
3. Python Client Generation:
    - Use Swagger tools to generate a Python client from the OpenAPI specification.
4. Test the Python Client:
    - Run your pseudo-RESTful service and test the generated Python client.