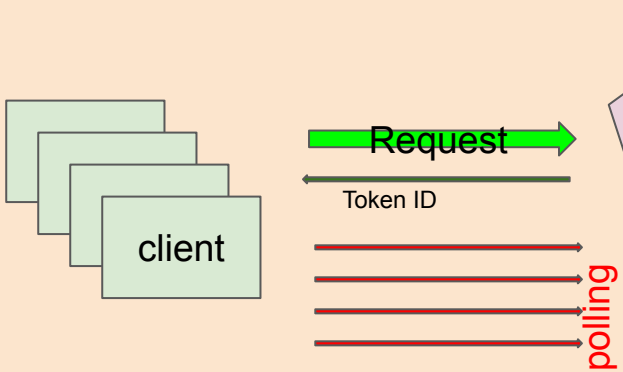
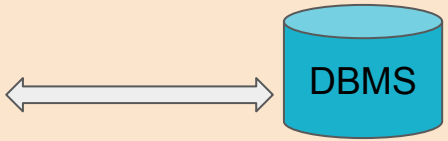
Autograder server

Preeti(22D0376), Shubham(22D0378)

The Architecture





Req_ID	Check_Status	Grading Status	compile error	runtime error	output
123	0/1	G/NG	0/1	0/1	file ptr
...	...	...	...	...	...

Functions that the server will implement

1. Request Handling
2. Request ID generation
3. Request queue
4. Request Status
5. Data storage
6. Handling ‘status’ Requests
7. Managing Request Lifecycle
8. Handling Client Timeouts
9. Logging and Audit
10. Graceful server Restart
11. Handling Concurrent status Requests
12. Client Communication

Pseudo code(we have used python but implementation is in C only)

1. Request Handling

```
while True:
    client_connection = accept_connection()
    request = receive_request(client_connection)

    if request.type == 'new':
        request_id = generate_unique_request_id()
        add_request_to_queue(request, request_id)
        send_response(client_connection, f'Your grading request ID
{request_id} has been accepted and is currently being processed.')
    elif request.type == 'status':
        request_id = request.request_id
        status = get_request_status(request_id)
        send_status_response(client_connection, status)
    else:
        send_invalid_request_response(client_connection)
```

2. Request ID generation

```
# Pseudo-code for generating a unique request ID
def generate_unique_request_id():
    # Generate a unique request ID, e.g., using UUID or timestamp +
counter
    request_id = generate_request_id()
    return request_id
```

3. Request Queue

```
# Pseudo-code for managing the request queue
request_queue = []

def add_request_to_queue(request, request_id):
    # Add the request to the queue with its status
    request.status = 'Accepted'
    request_queue.append((request, request_id))
```

4. Data Storage

```
# Pseudo-code for storing request status and results
def save_request_status(request_id, status):
    # Store the request status in a database or file
    store_in_database(request_id, status)

def get_request_status(request_id):
    # Retrieve request status from the storage
    status = retrieve_from_database(request_id)
    return status

# Similar functions for saving and retrieving grading results
```

4. Handling ‘status’ Requests

```
# Pseudo-code for handling 'status' requests
def send_status_response(client_connection, status):
    if status == 'Accepted':
        send_response(client_connection, f'Your grading request
ID {request_id} has been accepted. It is currently at position
{queue_position} in the queue.')
    elif status == 'In Process':
        send_progress_response(client_connection, progress)
    elif status == 'Completed':
        grading_results = get_grading_results(request_id)
        send_results_response(client_connection,
grading_results)
    elif status == 'Not Found':
        send_response(client_connection, 'Grading request not
found. Please check and resend your request ID or re-send your
original grading request.')
```

5. Managing Request Lifecycle

```
# Pseudo-code for updating request status
def update_request_status(request_id, new_status):
    # Update the request status in the storage
    save_request_status(request_id, new_status)
```

6. Handling Client Timeouts:

```
# Pseudo-code for handling client timeouts
def handle_timeout(request_id):
    # Check for client timeouts and update the request status
    current_status = get_request_status(request_id)
    if current_status == 'In Process' and timeout_expired:
        update_request_status(request_id, 'Not Found')
```

7. Logging and Audit

```
# Pseudo-code for logging request activities
def log_activity(activity):
    # Log the activity for debugging and auditing purposes
    log(activity)
```

8. Graceful Server Restart

```
# Pseudo-code for recovering request statuses and results after a
server restart
def recover_request_statuses():
    # Read request statuses from persistent storage and update
the request queue
    statuses = read_statuses_from_storage()
    for request_id, status in statuses:
        add_request_to_queue(status, request_id)
```

9. Handling Concurrent Status Requests:

```
#Ensure that the status update mechanism handles multiple
concurrent 'status' requests for the same request ID without
conflicts.
```

Client Communication:

The client-side program should be updated to send 'new' requests with filenames and 'status' requests with request IDs, as specified in the program usage.