High-Level Design (HLD)

System Overview

The camera system is designed to handle concurrent capture requests based on urgency. It processes requests by maintaining a priority queue and utilizes asynchronous workers to handle image capture tasks.

Components

1. **CameraController**
    - **Responsibility**: Handles HTTP requests to submit capture requests. Receives requests and passes them to the `CameraService` for processing.
    - **Endpoints**:
        - `POST /camera/capture`: Accepts a `CaptureRequest` object and submits it to the service.
2. **CameraService**
    - **Responsibility**: Manages the submission of capture requests to the `PriorityCaptureQueue`. It interfaces between the controller and the queue.
    - **Methods**:
        - `submitCaptureRequest(CaptureRequest request)`: Adds the capture request to the priority queue.
3. **PriorityCaptureQueue**
    - **Responsibility**: Maintains a priority-based queue of capture requests. Provides methods to enqueue and dequeue requests.
    - **Methods**:
        - `enqueue(CaptureRequest request)`: Adds a request to the queue based on its priority.
        - `dequeue()`: Retrieves and removes the highest-priority request from the queue.
        - `getSize()`: Returns the current size of the queue.
4. **CaptureWorker**
    - **Responsibility**: Continuously processes capture requests from the `PriorityCaptureQueue`. Simulates image capture and notifies the client

based on success or failure.

- **Methods**:
  - `start()`: Starts a background thread to process requests.
  - `processCaptureRequest(CaptureRequest request)`: Simulates the image capture process and updates the callback with the result.

Use-Case Diagram

Description: The use-case diagram should depict how the camera system handles concurrent requests with different urgency levels.

- **Actors**:
  - Client (User submitting capture requests)
- **Use Cases**:
  - Submit Capture Request
  - Process Capture Request
- **Logical Flow**:
  1. **Client submits capture requests**: Requests are sent to `CameraController`.
  2. **Controller passes requests to `CameraService`**: `CameraService` enqueues requests in `PriorityCaptureQueue`.
  3. **CaptureWorker processes requests**: The worker processes requests based on their priority, simulating capture and notifying the client.

Low-Level Design (LLD)

Classes and Interfaces

1. **CameraController**
   - **Responsibilities**:
     - Receive HTTP requests.
     - Forward requests to `CameraService`.
   - **Methods**:
     - `captureImage(CaptureRequest request)`: Endpoint to handle `POST /camera/capture`.

```
package com.nymble.shubham.camera_system.controller;
```

```
import com.nymble.shubham.camera_system.model.CaptureRequest;
import com.nymble.shubham.camera_system.service.CameraService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;


@RestController
@RequestMapping("/camera")
public class CameraController {

    @Autowired
  private CameraService cameraService;

  @PostMapping("/capture")
    public String captureImage(@RequestBody CaptureRequest request) {
        cameraService.submitCaptureRequest(request);
 return "Capture request submitted";
  }
}
```

2. **CameraService**

   - **Responsibilities**:
     - Manage request submission.
   - **Methods**:
     - `submitCaptureRequest(CaptureRequest request)` : Adds the request to

```
package com.nymble.shubham.camera_system.service;


import com.nymble.shubham.camera_system.model.CaptureRequest;
import com.nymble.shubham.camera_system.queue.PriorityCaptureQueue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class CameraService {

   private Logger logger = LoggerFactory.getLogger(this.getClass());
 @Autowired
 private PriorityCaptureQueue priorityCaptureQueue;
```

```
public void submitCaptureRequest(CaptureRequest request) {
       priorityCaptureQueue.enqueue(request);
 logger.info("Queue size is: {}",priorityCaptureQueue.getSize());

 }
}
```

3. **PriorityCaptureQueue**

   - **Responsibilities**:
     - Handle the priority queue for capture requests.
   - **Methods**:
     - `enqueue(CaptureRequest request)` : Add request to queue.
     - `dequeue()` : Retrieve and remove the highest-priority request.
     - `getSize()` : Get the current size of the queue.

```
package com.nymble.shubham.camera_system.queue;


import com.nymble.shubham.camera_system.model.CaptureRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import java.util.concurrent.PriorityBlockingQueue;

@Component
public class PriorityCaptureQueue {

    private Logger logger = LoggerFactory.getLogger(this.getClass());
 private PriorityBlockingQueue<CaptureRequest> queue;

 public PriorityCaptureQueue() {
       queue = new PriorityBlockingQueue<>();
  }

    public void enqueue(CaptureRequest request) {
        logger.info("Enqueuing capture request with urgency: {}", requ
  queue.put(request);
  logger.info("Queue size after enqueuing: {}", queue.size());
  }

    public CaptureRequest dequeue() throws InterruptedException {
        return queue.take();
```

```
  }

    public int getSize(){
        return queue.size();
  }
}
```

4. **CaptureWorker**

   - **Responsibilities**:
     - Process capture requests in a background thread.
   - **Methods**:
     - `start()`: Start processing requests.
     - `processCaptureRequest(CaptureRequest request)`: Simulate capture process and handle callbacks.

```
package com.nymble.shubham.camera_system.worker;


import com.nymble.shubham.camera_system.model.CaptureRequest;
import com.nymble.shubham.camera_system.model.CaptureResult;
import com.nymble.shubham.camera_system.queue.PriorityCaptureQueue;
import jakarta.annotation.PostConstruct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;

public class CaptureWorker {

private Logger logger = LoggerFactory.getLogger(this.getClass());
  @Autowired
  private PriorityCaptureQueue priorityCaptureQueue;

 public CaptureWorker(PriorityCaptureQueue queue) {
        this.priorityCaptureQueue = queue;
  }

    @PostConstruct
  public void start() {
        Thread workerThread = new Thread(() -> {
            while (true) {
                try {
                    CaptureRequest request = priorityCaptureQueue.dequ
  processCaptureRequest(request);
```

```
  } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
  }
            }
        });
  workerThread.setDaemon(true);
  workerThread.start();
  }

    private void processCaptureRequest(CaptureRequest request) {
        // Simulate image capturing logic
  boolean success = Math.random() > 0.4; // 60% success rate
  logger.info("Received capture request: {} with success status: {}",r
  CaptureResult result;
 if (success) {
            result = new CaptureResult("CapturedImage", null);
  logger.info("notifying client for success");
  logger.info("callback url: {}",request.getSuccessCallback().getOnSuc
  logger.info("message: {}",result.getCapturedImage());
  request.getSuccessCallback().setSuccessFromCallback(result);
  } else {
            result = new CaptureResult(null, "Capture failed");
  logger.info("notifying client for failure");
  logger.info("callback url: {}",request.getFailureCallback().getOnFai
  logger.info("error: {}",result.getErrorMessage());
  request.getFailureCallback().setOnFailureFromCallback(result);
  }
    }
}
```

Use-Case Diagram

Description: The use-case diagram for LLD should detail the interactions between components when handling capture requests with varying urgency levels.

- **Actors**:
  - **Client**: Submits requests.
  - **CameraController**: Receives and forwards requests.
  - **CameraService**: Manages request submission.
  - **PriorityCaptureQueue**: Stores and retrieves requests.
  - **CaptureWorker**: Processes requests.

Logical Flow:

1. **Client submits requests**: Requests are sent to `CameraController`.
2. **Controller to `CameraService`**: Forwards requests for processing.
3. **Service to `PriorityCaptureQueue`**: Adds requests to the queue.
4. **Worker processing**: `CaptureWorker` processes requests from the queue and handles callbacks.

Running Application

- Using Maven: `mvn spring-boot:run`
- Using JAR: `java -jar target/camera-system-<version>.jar`

Swagger URL

- open http://localhost:8080/swagger-ui/index.html

Endpoint: `POST /camera/capture`

Example Payload:

```
{
    "urgency": 5,
    "successCallback": {
        "onSuccess": "http://example.com/success"
    },
    "failureCallback": {
        "onFailure": "http://example.com/failure"
    }
}
```

Github Link : https://github.com/Jaiswal-Shubham/camera.git