

1. User/Client - Every client will make request to camera module with the image to be captured and the client will have data members such as client’s id, name, state etc..
2. Camera Module - Camera module will forward the request of the client with the priority of the client, to manage the urgent requirements of the different clients. It will hold the capture logic to handle multiple requests made to the camera system and will direct the requests to be handled based on their priority.
3. Capture Module - This module will get the request that needs to be processed at the given instant time frame based on its priority. It will be handling one request at a time.
4. Callback Handler - Callback handler will handle the callback requests and send the response back in the camera module with the response as the captured image or descriptive error message based on the success or error of the response message

**Client**- clientId, clientName, clientState //getter and setter, submitRequest()

**Request** - requestId, requestPriority, assignRequestPriority()

**RequestHandler** - client, Priority\_Queue<request> addRequestToQueue(), sendRequest(), registerCaptureCallback(), processCallback(), removeRequestFromQueue()

**Response** - responseId, responseMessage, responseStatus

**ResponseHandler** - response, Image, sendResponse(), getCallbackResponse()

**CaptureModule** - captureImage() singleton class, only one instance will be present at time

**CameraModuleMangager** -List<Client> clientList, requestHanlder, responseHandler, addClient(), submitCaptureRequest(), captureImage()

**Image** - imageId, imageMetaData, List<Pixel> pixels

**Pixel** - xLocation, yLocation

**Priority** - LOW, MEDIUM, HIGH -> Enum

**ClientState** - READY, WAIT -> keep volatile so change in other thread will be visible

**ResponseStatus** - FAILURE, SUCCESS, TIMEOUT

1. We can use **Strategy Design Pattern** to implement capture strategies for different type of cameras
2. We can use **Obsever Design Pattern** to implement callback response
3. **Facade Design Pattern** can be used to construct camera functioning
4. Concurrency handling will be done using executor service with n size of thread pool. Synchronise block will be placed while implementing capture method in CaptureModule and to make client wait until response is not received use mutux on Client state. So Before capture request proceed, put client into WAIT state and once capture response is received change client state into READY state.