

1. Client- The system is capable of handling concurrent requests from multiple clients, each potentially initiating operations simultaneously.
2. Load Balancer - All incoming requests from the client are directed to the load balancer, which will manage the distribution based on the system's capacity. The load balancer will ensure that the traffic is evenly distributed among available resources, utilizing configuration-based rate limiting to control the flow of requests. This will help to prevent system overload and will maintain optimal performance.
3. Message Queue - As the system needs to handle the urgency/priority of some services/requests using the module, we will use the message queue to store the remaining requests which can be handled at a later time, when the system has available resources and it will dequeue these requests from the queue and begin processing them. This approach ensures that no requests are lost and maintain an efficient flow of operations.
4. Camera System - This system will have the logic to process each incoming request. Upon successful processing, it will return the captured image and in the backend it will also store the image in persistent database with proper logging mechanism for future reference for quick debugging in case of any issue. If the request fail for any reason, the system will return a failure message to the client, ensuring that the user is promptly informed of the issue.
5. Config Service - This has two components for secure and efficient access
   1. Rate-limiter - This component will control the number of requests made by the client in a specified time frame. This will prevent abuse, fair distribution of resources among clients and system stability.
   2. Authentication - This component will ensure only authorized users can access the camera system, reducing the risk of misuse and unauthorized access

6. Database Store - The system will use blob database to store captured images. We will also implement a caching layer that stores indexing information of the database, which will help in faster access to the image.

7. Handle Concurrency - we’ll use optimistic lock to handle concurrency request and to synchronise all the requests, use synchronised block as well as volatile atomic variables. Mutex is also helpful here to avoid deadlock situation.