Concurrent Capture System Design

High level design :

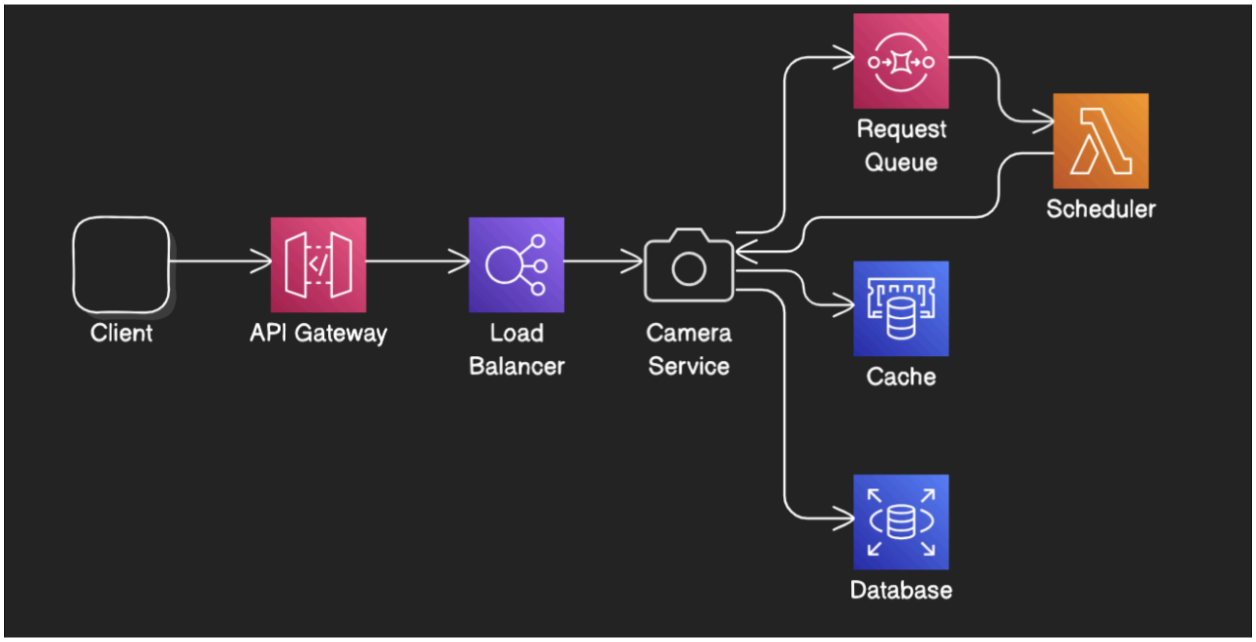
Components:

1. API Gateway and Load Balancer
2. Camera service
3. Request queue
4. Scheduler
5. Cache
6. Database

Components and Responsibilities:

1. **API Gateway and Load Balancer**
    - Acts as entry point for all client requests
    - Distributes incoming requests across multiple instances of the camera system services
2. **Camera service**
    - Interface to the actual camera hardware and overall management of the capture process.
3. **Request queue**
    - Manages incoming capture requests and prioritizes them based on urgency.
4. **Scheduler**
    - Manages the prioritization and dispatching of capture requests
5. **Cache**
    - Temporarily stores recently captured images and frequently accessed data to reduce latency
6. **Database:**
    - Stores persistent data related to capture requests and results.

HLD diagram:



Flow explanation:

- Client submits a capture request to the API Gateway.
- API Gateway routes the request to an appropriate Scheduler instance using load balancing.
- Scheduler enqueues the request in the RequestQueue based on its priority.
- Scheduler checks the cache for a recently captured image that matches request.
- If the image is found in the cache, the Scheduler retrieves the cached image.
- Scheduler returns cached result to API Gateway.
- API Gateway sends the cached capture result back to the client.
- If the image is not found in the cache, Scheduler proceeds with processing the request.
- Scheduler dequeues the highest priority request and initiates the capture process with CameraSystem.
- CameraSystem captures the image and returns the result.
- Scheduler stores the capture result in cache for future requests.
- Scheduler saves the capture result in the database for persistent storage.
- Scheduler returns the capture result to API Gateway.
- API Gateway sends the capture result back to the client.

DB choice:

- For the camera system, a **NoSQL database is recommended** over a relational database. NoSQL databases offer horizontal scalability, flexible schema, and high write/read throughput, making them ideal for handling large volumes of data and high write loads, especially for unstructured data like images.
- Relational databases, while providing ACID compliance and good support for complex queries, face challenges with horizontal scalability and performance issues with high write loads and large binaries, making them less suitable for this use case.

Why cache is required:

1. **Reduce Latency and improved throughput :** Serve cached results quickly, minimizing wait times.
2. **Reduce Load on Hardware:** Decrease direct access to camera hardware, extending its lifespan.

Example:

- Without cache both low and high-urgency clients directly access the camera, causing delays.
- With Cache
    - **Journey Logging Client:** Receives a quick response from cache, reducing hardware load.
    - **Steering Decision Client:** Gets prioritized, direct access to the camera for the most current image.