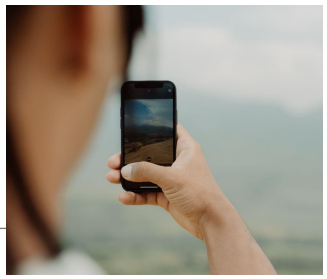
Logs: What would you log?

A photo-upload API stores a photo, then asks a separate service to create a thumbnail. It returns success only after both steps finish.



Andrew IDs:

Design the logs

Add **up to four logging calls across both services**. For each, specify its location, event, level (`debug`, `info`, `warn`, or `error`), and fields. Show where a correlation ID is created, passed, and attached to records. Write **one example emitted record**.

Number your chosen logging locations in the code, then complete the design sheet on page 2. Use short notes or pseudocode; you may add supporting code and error handling. Working code is unnecessary.

Code

```
// Service: upload-api
async function uploadPhoto(request) {
  const photo = request.file;
  if (!supportedType(photo.type)) {
    return response(415);
  }

  const stored = await storage.put(photo);
  const thumbnail = await thumbnailService.create(stored.id);
  return response(201, { photoId: stored.id, thumbnail });
}

// Service: thumbnail-service
async function createThumbnail(request) {
  const existing = await thumbnails.findByPhotoId(request.body.photoId);
  if (existing) {
    return response(200, { thumbnailId: existing.id });
  }

  const photo = await storage.get(request.body.photoId);
  const thumbnail = await resize(photo);
  const stored = await thumbnails.put(request.body.photoId, thumbnail);
  return response(201, { thumbnailId: stored.id });
}
```

Assumptions

- Requests run concurrently. Each service's JSON logger adds timestamp and service name.
- `thumbnailService.create` calls `createThumbnail` over HTTP, sending `photoId` in the body.
- Identical photos share a storage ID. Thumbnail lookup returns an existing record or `null`.
- Storage and processing failures throw; the HTTP client also throws on timeout or non-2xx responses. Unhandled exceptions produce HTTP 500. There are no retries.
- The uploaded file is available before `uploadPhoto` starts.

Your logging design

Match the numbers below to locations you mark in the code on page 1. For calls in new error handlers, mark which operations the handler surrounds.

Logging calls

For each call, write its **event, level, and fields**. You may use fewer than four.

1.

2.

3.

4.

Correlation ID

Sketch or describe where the ID is created, how it crosses HTTP, and how it reaches each record.

One example emitted record

Include the fields the logger adds automatically. JSON or clearly labeled fields are enough.

LogQL cheatsheet

Select a service, parse the JSON record, then filter its fields. Replace the example names and values with your own.

Purpose	Example
Select a service	`{service="example-api"}`
Select either service	`{service=~"example-api|example-worker"}`
Parse JSON fields	`| json`
Match a field	`| operation="example_operation"`
Compare a numeric field	`| item_count > 10`
Match a correlation ID	`| correlation_id="abc123"`
Search message text	`|= "example text"`

Combine the pieces

```
{service="example-api"} | json
  | operation="example_operation"
  | item_count > 10
```

Each additional filter narrows the results. This query selects records from `example-api` with the given operation and an item count greater than ten.

For this exercise

- `service` is a **Loki label**, with values `upload-api` and `thumbnail-service`. Other fields are inside each JSON record.
- Select the time range in Grafana; it is not part of these queries.
- Quote strings. Compare numeric fields using their recorded units.

Test your logging design

Wait for the instructor to reveal the problem. Use your own field names and units, with the cheatsheet on page 3. Use a placeholder correlation ID for a hypothetical matching record.

Query 1

Query 2

Revise if needed

If your records lack evidence needed for the queries, revise a logging call here. Otherwise, explain which fields make your queries possible.