Hamsini Neelakantan
42110417

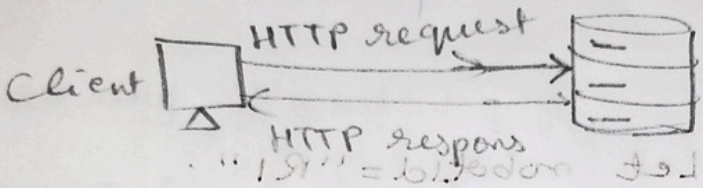
PART A [Backend + Frontend]

Q.1

- Explain request life cycle in ~~Fastapi~~ / Express

↳ Basic communication on web is based on Request and response in cycle.

↳



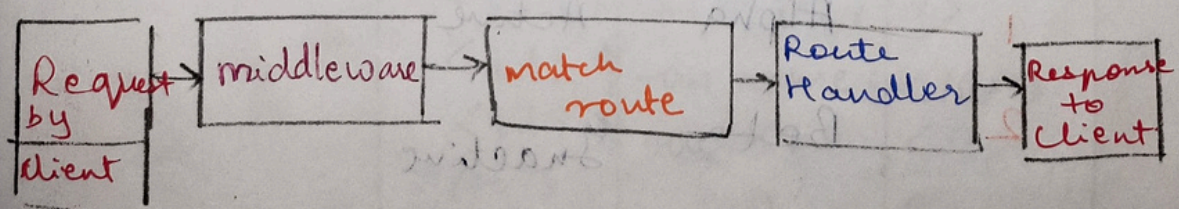
↳ for example, If I visit a website like "https://www.youtube.com/user", I have sent an HTTP request to server to get me user page. (GET request.)

↳ Now Express.js processes the request and sends request through middleware where authentication or logging takes place.

↳ Express finds route handler that matches request URL and HTTP method. and a response object is prepared.

↳ which is fetching of user data.

↳ Response is sent to client as plain text or JSON or Status code.

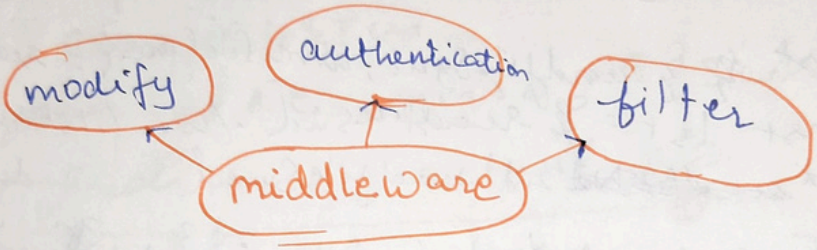


- What is Middleware? Give examples.

↳ Middleware is a function that sits between client request request and server response.

↳ It has access to:

- the request object (req)
- response object (res)
- next function (next())



I used ~~Ngj~~ Node.js + Express in my Smart traffic management system where the purpose of project was to create green corridor and all traffic lights become green from red.

So Express provides routes and pathways for every device to talk to server.

↳ so this decides things like:

① Vehicle volume
② Override red lights
③ A dashboard alert.

① So whenever I get data ~~from~~ ESP8266 on ~~traffic~~, ambulance GPS ; data ~~is stored~~ goes in JSON parser and ~~a middle~~ is processed.
This is middleware 1.

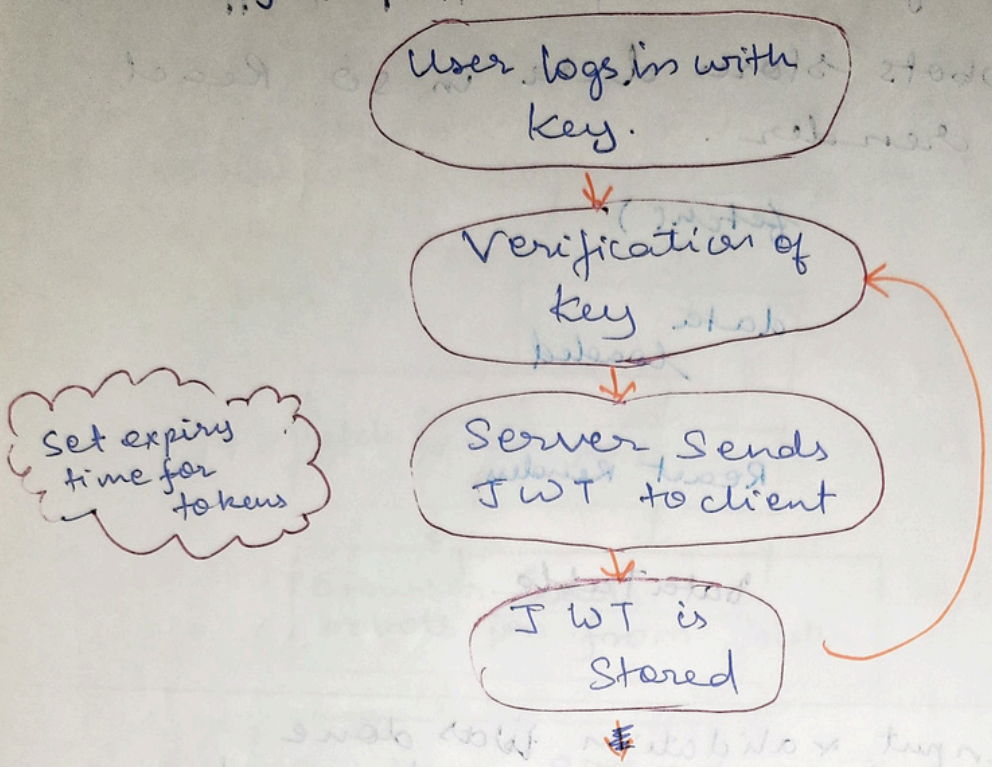
② Then request is logged for authentication middleware 2.

(3) Input validation was done to check for coordinates and how to fill null values.
- middleware 3.

- How do you manage authentication JWT / OAuth2 ?

- JWT - Json Web Token
  - ↳ works when server creates token after login/sign in then gives to client.
  - ↳ Then client has to give back on each request
  - ↳ In this token authentication there are 3 parts
    (1) Header (2) Payload (3) Signature

Working -



In OAuth 2.0

↳ it is an authorization framework that enables app to access some user accounts on HTTP service.

Steps -

Step1: Register App with OAuth provider.

Step2: Install package like "passport-google-oauth20"

Step3: Configure Passport strategies

Step4: Set routes for starting OAuth and handle callbacks.

Step5: DO create Or Update.

— Async vs Sync in Node.js.

Sync.

↳ Its fileSystem method blocks the event loop until operation completes.

eg.

```
const fs = require("fs");

const data = fs.readFileSync("App.txt", "utf8")
console.log("Done");
```

Asynchronous.

↳ Its fileSystem module method which doesn't block event loop and handles multiple operations simultaneously.

eg.

```
const fs = require("fs");
fs.readFile("App.txt", "utf8", (err, data) => {
console.log("File read");
});
console.log("Next work");
```

So basically lets say I order one Tea, All I can do is stand still, wait thats it, till I receive my order

↳ This is Sync.

But if I can answer a call, do my assignment & and scroll social media till my Tea arrives

↳ This is Async.

— Structure a Scalable backend project folder —

↳ Scaling depends on the purpose so, I need

1. Routes
2. Controllers
3. services.
4. Models

Structure will look like —

```
project/src/
routes/ → ~~end~~ API endpoint
controllers/ → logic of route
services/ → algorithms.
models/ → database
  ⋮ and so on.
```

I will use middleware/ utils/ config/ app.js / ↳ test.js.

— Idempotency in POST/PUT.

↳ means to repeat same request and must get same output every time API is called.

POST is not idempotent inheritently.

Ways to make POST idempotent

1. Using ddempotency key : 123abc. server stores key + result.
2. using checks with if statements.

PUT is idempotent.
so it maintains overwrite status.

- Rate limiting

↳ It is for crowd-control for backend

↳ It restricts on how many requests ~~req~~ an user, IP or token can make in time window.

Used / Applied in:

- API spam
- Brute force attacks.
- ~~DDO~~ on Public APIs.

Q.2. Github link. github link

Q-3.

PROPS	STATE
• Data is passed from component to another	• Data is passed within the component.
• It is immutable	• Mutable.
• Used with functional and state component.	• Used with State component only.

I have used in Smart traffic Mngmt Sys.
If i have to change or set colour, then it is Prop.

```
function Robo ({ id, battery, online }) {
  return (
    <div className="max-w-xs mx-auto bg-white p-4">
      <h2 className="font-bold">Robo ID : {id}</h2>
      <p className="mb-1">battery : {battery}%</p>
      <p className={`font-semi bold
        ${online ? 'text-green' : 'text-red'}`}>
        {online ? 'Online' : 'Offline'}
      </p>
    </div>
  );
}
```

O/p

eg. id="R-102" battery={25} online={T}

Q.7. Pagination.

↳ It means to request some data insted of fetching all at once.

↳ ① Server Side
② Client side. } types.

↳ function - Set Total Pages, fetch.

Catching

Catching prevents getting same data again and again ⇒ speed up app.

① Brower Cache
② Local state Cache } types.

We use function like setcache, fetch

PART B [DevOps]

Q.9. (1) Lets say we have.

```
1 From: node:18
2 Work dir /app
3 Copy package*.json ./
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD ["node", "index.js &"]
```

1. It is the base of img.
2. It is working directory
3. Copying package.json to host
4. Installing node.js dependencies in the container.
5. Copy files.
6. Listen on port 3000
7. Command to run when container starts.

Q.10

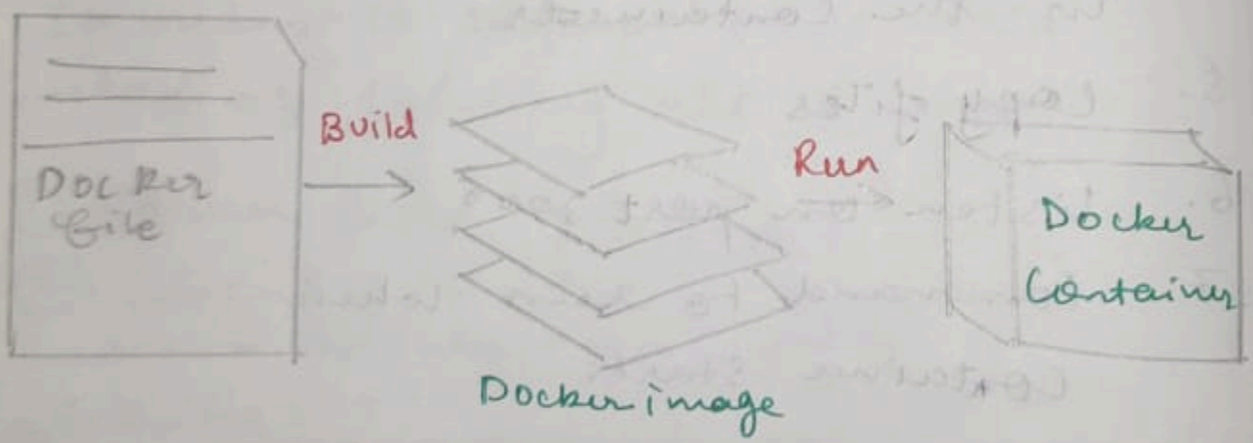
Docker Image	Docker Container.
• Blueprint of container • Created only once • It has everything an needs like all the packages, code, library. • It is Static	• Instance of Image • Can be created many times. • It is the instance of image uses CPU, memory, networking. • We can make in dynamic

in Simple words of Analogy.

Docker Image is a cake recipe, I can share it, store it but I still have not made a full cake.

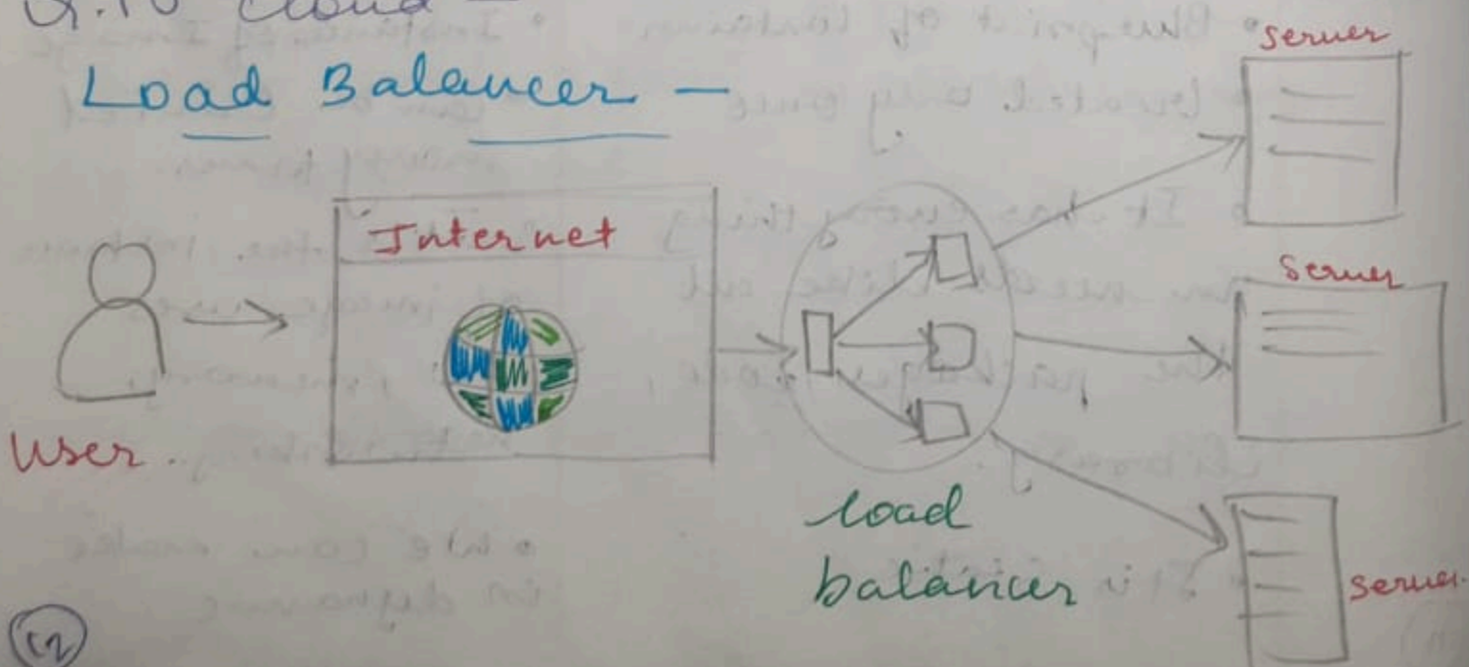
Docker Container is when a followed all the steps in recipe and bake a Cake

So cake is edible, readable, usable and real. So I slice and eat or ~~~~ work with it.



Q.10 cloud —

Load Balancer —



Load balancer basically distributes incoming network traffic accross multiple servers. so that server doesn't get crashed

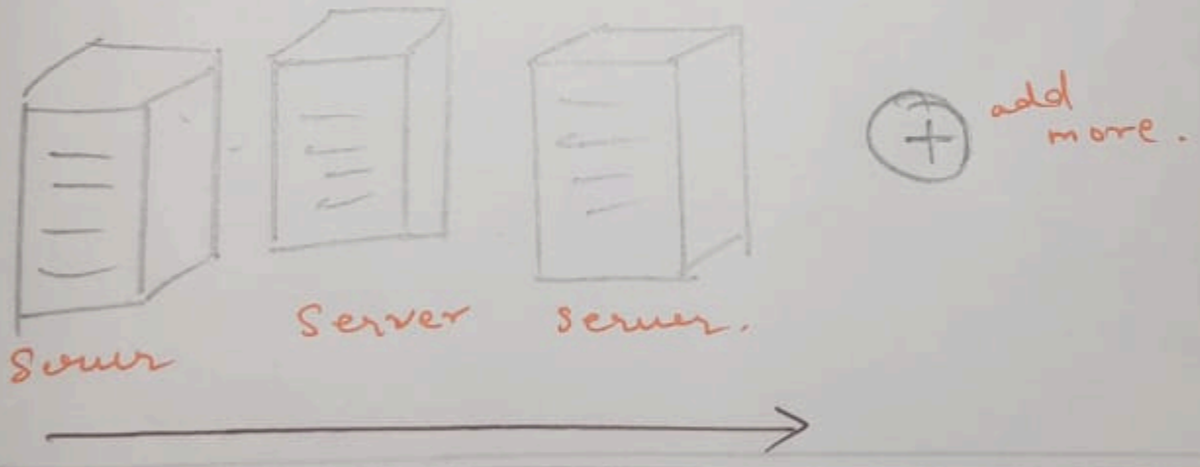
↳ Prevents downtime == More performance

↳ It handles each request.

eg. Round Robin, Hashing.

Horizontal Scaling — (HS)

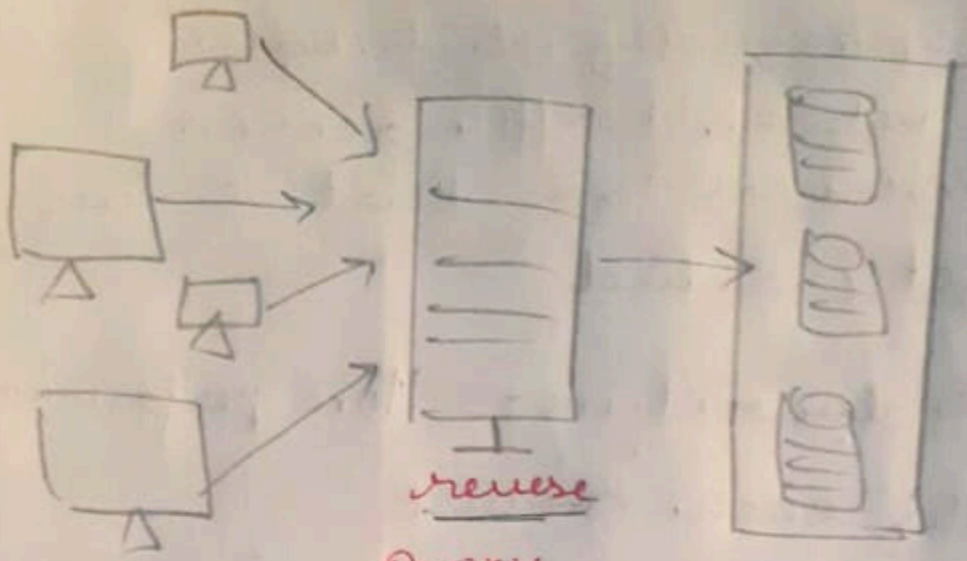
* Adding more servers to handle high load is Horizontal Scaling.
* If we keep making one server Stronger at one point of time server may get crashed.
* So HS. handles traffic efficiently.



— Reverse Proxy

A reverse Proxy sits in front of backend servers and forwards client request to them.

* It does:
  - load balancer.
  - caching
  - SSL stopping.




```
Server {
    listen 80;
    server_name xyz-loon;
    location / {
        proxy_pass https://localhost:3000;
        proxy_http_version 1.1;
        try_files $uri $uri/ =404
    }
}
```

PART - C ROS

Q.15. Kafka

Message Queues (MQ) is a system that allows different parts of an app to communicate asynchronously by sending messages to a queue.

- Producer - Sends msg.
- Consumer - Reads msg.

* MQ can be used when there is spike in traffic and improves reliability.

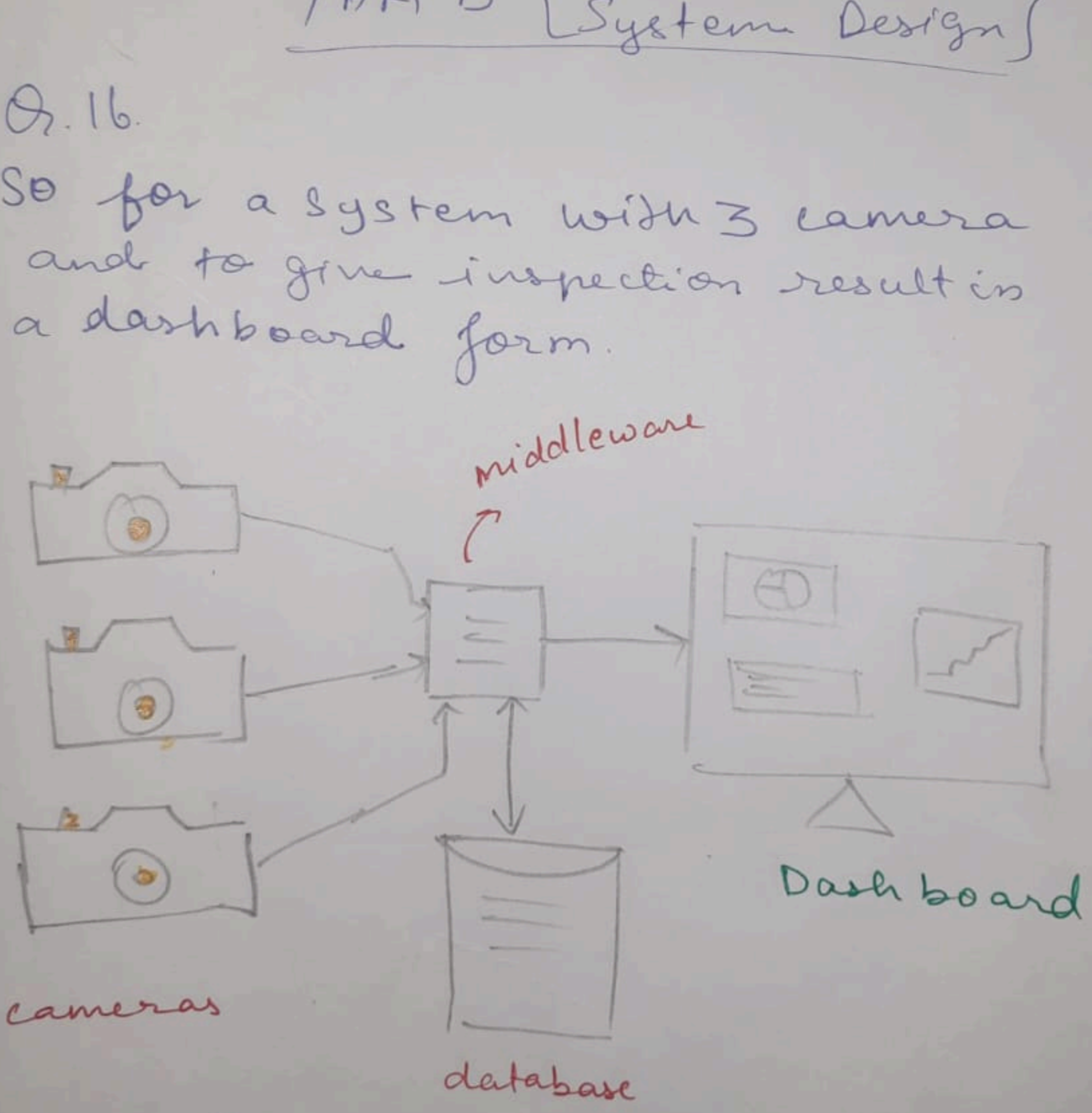
So Since Kafka is a event Streaming platform, it is used as MQ for large real time data Streaming.

So producer sends, consumer reads and Topics gives feed from stored data.

PART D [System Design]

Q. 16.

So for a system with 3 camera and to give inspection result in a dashboard form.



Requirement Analysis

↳ data from camera
↳ Display on dashboard
↳ Alert.

PART E [Coding Round]

Q. 19. Medium level

1. MySql

```
Select * ~~robot_logs~~ from robot_logs
where robot_id = ?
order by timestamp desc
limit 20;
```

Mongo

```
Let robotid = "R1".
db.robot_logs.find({ robot_id: "R1" })
             .sort({ timestamp : -1 })
             .limit(20);
```

2. Lets say we have a JSON like -

[{ "id": 1, name : Alpha, Status : Active"},
{ id : 2, name : Beta, Status : "Inactive" }]

So we have 3 columns id, Name, Status

We wrap everything in tags:

① <table> ② <thead> ③ <tbody>

Output should look like -

ID	NAME	STATUS
1	Alpha	Active
2	Beta	Inactive

There are 2 cases

① Importing JSON from a file

② Fetching JSON from an API

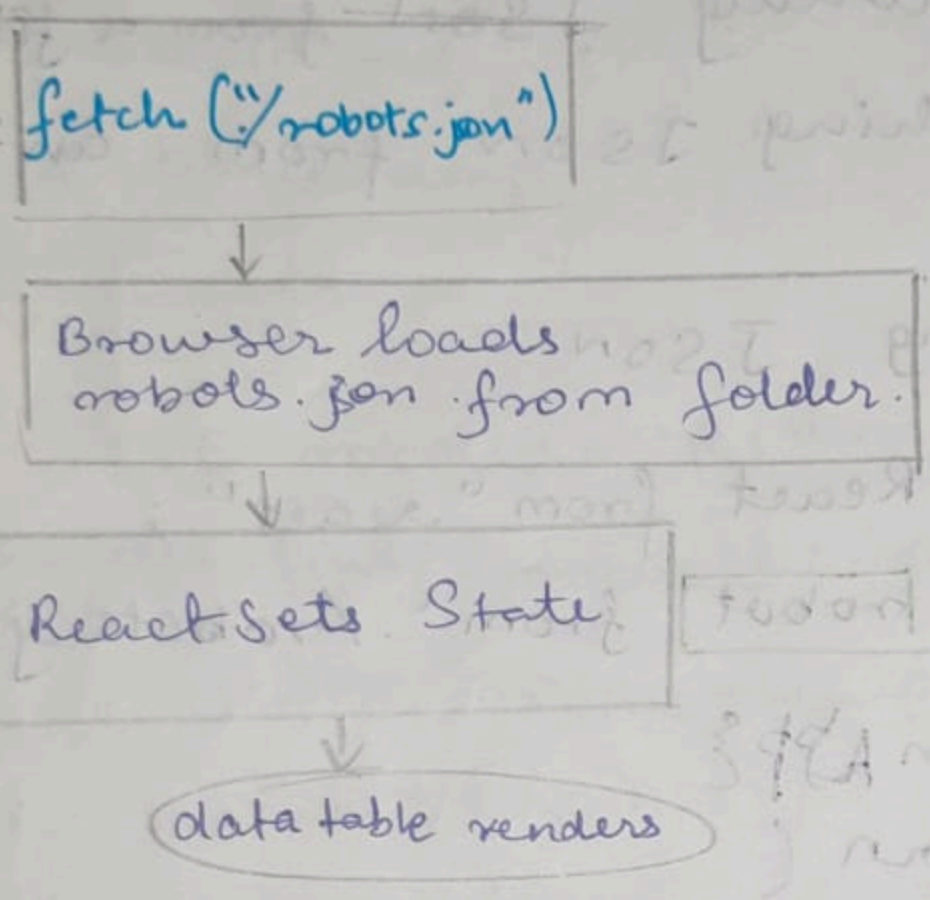
① Importing JSON

```
import React from "react";
import robot from './robots.json';

function App(){
  return (
    <div>
      <h2> Robot Data table </h2>
      <table border="1" cellPadding="8">
      <thead>
        <tr>
          <th> ID </th>
          <th> Name </th>
          <th> Status </th>
        </tr>
      </thead>
      <tbody>
        {robot.map((row) => (
          <tr key={row.id}>
            <td>{row.id}</td>
            <td>{row.name}</td>
            <td>{row.status}</td>
          </tr>
        ))}
```

```
</tbody>
</table>
</div>
);
}
```

So here —



(2) fetching JSON from API

```
import React, { useEffect, useState } from 'react';

function App() {
  const [robot, setData] = useState([]);

  useEffect(() => {
    fetch('https://App.com/robots')
      .then((res) => res.json())
      .then((json) => setData(json))
      .catch((err) => console.error
        ('Error', err));
  }, []);
  return ( )
```

Same as previous one.

- fetch gets JSON from https://App.com/robot
- Set Robots stores data in SO React can render.