**Microservices Architecture**

Design a microservices-based concert ticket booking system with user-service, booking-service, and payment-service. Describe inter-service communication and folder structure per service.  
  
  
1. User-Service is using for authenticates(identified) the user before access booking and payment sevices and maintenance the user session.

2. Booking service receives a request to book a ticket:

* Calls user-service via HTTP to validate the user.
* Saves booking info in its DB.
* Sends a message or HTTP call to payment-service to initiate payment.

3. Payment-Service processes the payment If successful, updates status and notifies booking-service.

app.js used to handle initializion Start Express, load routes, connect DB

routes used to handle Mapping HTTP paths to controller functions

controllers used to write Business logic (register user, create booking , payment)

model used to Define database schema or ORM model

.env used to handle Configurable settings PORT,DB, and other configuration

package.json used to handle Dependencies that required per service

**SQL & Index Optimization**

Given a table 'orders(order\_id, user\_id, total\_amount, created\_at)' with 10 million rows, write an SQL query to get total spending per user in the last 30 days. Optimize with indexes.  
  
**Query**  
SELECT user\_id, SUM(total\_amount) AS total\_spending FROM orders WHERE created\_at >= NOW() - INTERVAL 30 DAY GROUP BY user\_id;

**Index**  
CREATE INDEX idx\_orders\_covering ON orders (created\_at, user\_id, total\_amount);

**Frontend Optimization**

Your dashboard displays users, transactions, and notifications. What optimizations would you apply to improve load time, interactivity, and code-splitting?   
1. Using lazy loading that make view only load users, transactions, or notifications when ready to visible.  
2. Using pagination to make batch on view  
3. Load from CDN to reduce bundle size  
4. Splitting code like to be small component or file css and js so you just can use and render what component or file that you needed

**CI/CD Deployment Strategy**

Explain a CI/CD workflow from Git commit to production deployment for a fullstack app.  
  
to make CI/CD to make Automatically integration  and deploys   
1. git init  
2. git remote add origin < https://github.com/Robin/fullstack.git>  
3. git status  
4. git add .  
5. git commit -m "fullstack app message"  
6. Make branch for dev and production  
7. git push -u origin dev  
8. Login to cpanel and get you SSH Accecs

9. Generate a New Key and download Key

10. go to github to ‘fullstack’ repository

11. to menu setting and deploy key  
12. Paste the public key in there  
13. Click clone and then ssh copy  
14. Back to git version control in cpanel  
15. Click create and paste the url ssh in there   
16. Go to gitlab repository   
17. Add Public Key SSH in menu SSH Keys   
18. Open repository in git lab , click clone , copy with ssh  
19 back to **Git Version Control** and create and copy ssh in there  
20. Done

**Code Review & Debugging**

Fix the errors in the following Express.js code that fetches a user by ID. Identify and explain at least 3 issues.  
  
Below are the three issues I identified in the provided Express.js code.

1. Incorrect use of res.send(404).json(...) (line 5)

- Problem: res.send() is used to send the body of the response, not set the status code.

- Fix: Use res.status(404).json(...) to set the HTTP status code and send JSON.

2. Missing Return Statement After Sending a Response (line 5)

- Problem: Even after res.send(404)..., the function continues and calls res.send(user);—this causes an error like “Cannot set headers after they are sent.”

- Fix: Add return to stop function execution after sending a response.

3. Assuming db.findById is synchronous

- Problem: Most DB access methods (e.g., MongoDB, Sequelize, etc.) are asynchronous and return a Promise.

- Fix: Use async/await if db.findById is async. Otherwise, you're working with a pending promise, not the actual user.  
  
**error code**   
  
app.get('/user/:id', (req, res) => {

const id = req.params.id;

const user = db.findById(id);

if (!user) {

res.send(404).json({ error: "User not found" });

}

res.send(user);

});  
  
**correct code**  
**app.get('/user/:id', async (req, res) => {**

**try {**

**const id = req.params.id;**

**const user = await db.findById(id); // Await the asynchronous DB call**

**if (!user) {**

**return res.status(404).json({ error: "User not found" }); // Correct status + return**

**}**

**res.json(user); // Return the user object as JSON**

**} catch (err) {**

**console.error(err);**

**res.status(500).json({ error: "Internal server error" });**

**}**

**});**

**API Security**

List at least 5 steps to secure a public POST /api/users endpoint from spam and abuse.  
  
1. Add limit data   
2. Using CAPTCHA or code  
3. Add validation input  
4. Auth verification  
5. Whitelist IP

**Efficient Server Hit Handling**

Your /api/products/popular endpoint gets 5000 hits/min. How would you reduce database load and ensure freshness?  
  
using Database Optimization Add indexing to make query data optimization or CDN Caching

**Note: Waktu pengerjaan berdurasi 12 jam. "code di upload ke repo github / gitlab"**