QUESTION:

1. Build a simple health check service that monitors 3 endpoints and exposes metrics via /health. Then write a brief analysis of what happens when one service goes down and how you'd detect it in production.

Part 1: Health check service

```
const http = require('http');
const https = require('https');
const url = require('url');

const services = [
  { name: 'Users API', url: 'https://salak-codes.github.io/Simon-Game/'},
  { name: 'Posts API', url: 'https://pokeapi.co/api/v2/pokemon/pikachu'},
  { name: 'Invalid API', url:  'https://api.adviceslip.com/advice'}
];

const checkIntervalMs = 30000;
const timeoutMs = 5000;
let healthStatus = {};

function httpRequest(targetUrl) {
  return new Promise((resolve, reject) => {
    const parsed = url.parse(targetUrl);
    const protocol = parsed.protocol === 'https:' ? https : http;

    const options = {
      hostname: parsed.hostname,
      path: parsed.pathname + (parsed.search || ''),
      port: parsed.port || (parsed.protocol === 'https:' ? 443 : 80),
      method: 'GET'
    };

    const req = protocol.request(options, res => {
      res.resume();
      resolve(res.statusCode);
    });

    req.setTimeout(timeoutMs, () => {
      req.destroy(new Error('request-timeout'));
    });
    req.on('error', err => reject(err));

    req.end();
  });
}

async function checkService(service) {
  const start = Date.now();
  try {
    const statusCode = await httpRequest(service.url);
    const responseTime = Date.now() - start;
    healthStatus[service.name] = {
      status: statusCode === 200 ? 'UP' : 'DOWN',
      statusCode,
      responseTime: `${responseTime}ms`,
      lastChecked: new Date().toISOString()
    };
  } catch (err) {
    healthStatus[service.name] = {
      status: 'DOWN',
      error: err.message,
      lastChecked: new Date().toISOString()
    };
  }
}

async function runChecks() {
  await Promise.all(services.map(svc => checkService(svc)));
  console.log(`[${new Date().toISOString()}] Health check complete`);
  console.table(healthStatus);
}

setInterval(runChecks, checkIntervalMs);
runChecks();

http.createServer((req, res) => {
  if (req.url === '/health') {
    res.writeHead(200, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify(healthStatus, null, 2));
  } else {
    res.writeHead(404, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ error: 'Not found' }));
  }
}).listen(3000, () => {
  console.log('Health check server running on port 3000');
```

```
});
```

Part 2: Service Failure and Detection

This is a brief analysis of what happens in my Health Check Service when one monitored service goes down, and how I would detect it in production.

What Happens in The Code;

1. In checkService():

- If status code is exactly 200, status is "UP".
- Anything else (timeout, network error, or status not 200) → status "DOWN" and error field is set.
- lastChecked is updated to show the exact timestamp of the failure.
- If it's "UP", it also records the statusCode and responseTime.

2. In /health endpoint:

- That service's entry will show "status": "DOWN".
- If the error was network-related, "error" will contain something like "request-timeout".
- Other services still "UP" will remain unaffected.

3. In console:

- Every 30 seconds (checkIntervalMs) you'll see a console table with the updated health status of all services.
- The failing service will be marked **DOWN** in the table.

The /health API will show that service as up: false. If at least one service is down but others are fine, overall status becomes 'degraded'. If all services are down, overall status becomes 'down'.

How I'd Detect It in Production;

My practical plan for detection of failures in production includes:

1. Browser/manual check**:**
Open http://<server-ip>:3000/health in a browser — the JSON will clearly show "status": "DOWN" for the failing service, with the last checked timestamp.

2. Log monitoring**:**
If you have logs sent to something like CloudWatch, Logstash, or Grafana Loki, you can filter for "status": "DOWN".

3. Add automated alerts:

- Convert the healthStatus into metrics.
- Create alerts that fire if status == "DOWN" for more than 2 consecutive checks.
- Cam integrate with email for notifications.

4. External probes:
Use an uptime monitoring tool (UptimeRobot, Pingdom, or Grafana Cloud Synthetic Monitoring) to check your /health endpoint from multiple regions.