AI ASSISTED CODING

END-LAB EXAM

NAME : R.SURYANARAYANA

HT.no : 2403A52038

BATCH : 03

Subset 16 — Security & Resilience for Critical Infrastructure

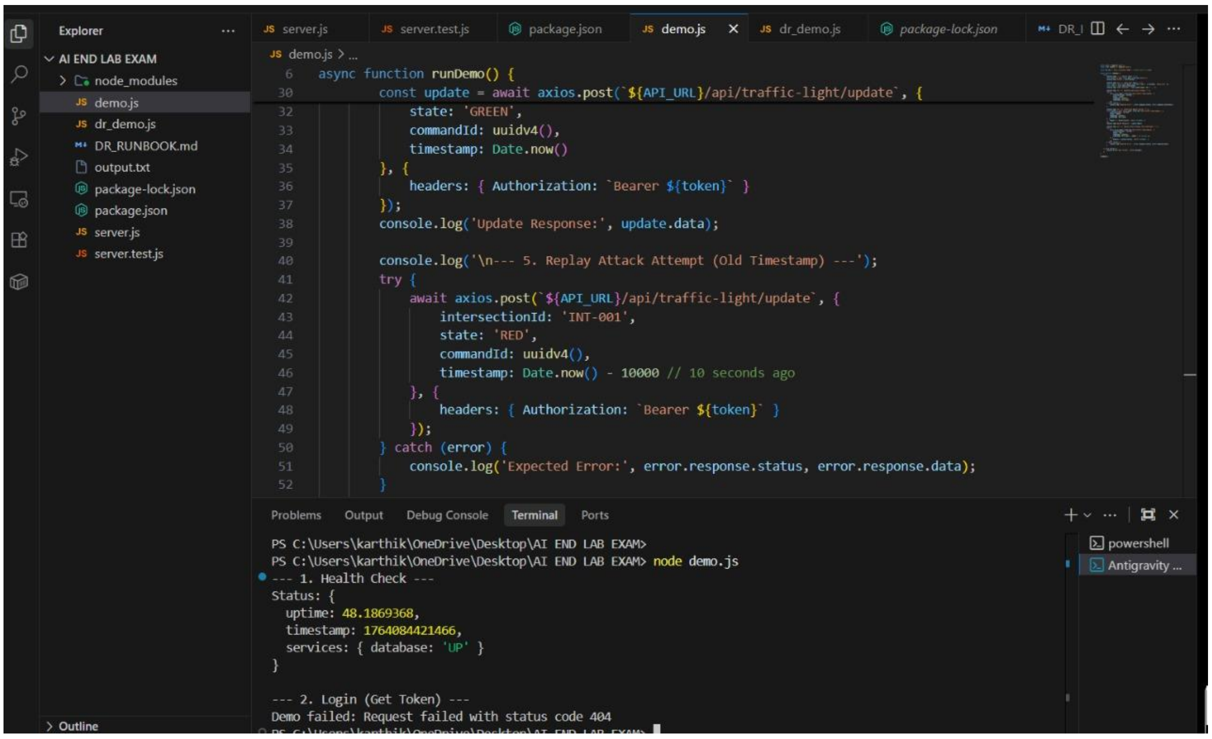
Q1: Threat modeling for critical control paths (traffic lights, alerts).

• Task 1: Use AI to enumerate attack vectors and mitigations.

Prompt:

Perform a high-level threat modeling exercise for critical control systems such as traffic lights and alerting mechanisms.Identify potential categories of threats (e.g., tampering, spoofing, disruption, data integrity risks) without providing any exploit details.For each threat category, suggest general, non-technical mitigation strategies such as improved authentication, monitoring, redundancy, fail-safe design, and policy controls.

CODE :



```javascript
async function runDemo() {
        const update = await axios.post(`${API_URL}/api/traffic-light/update`, {
            state: 'GREEN',
            commandId: uuidv4(),
            timestamp: Date.now()
        }, {
            headers: { Authorization: `Bearer ${token}` }
        });
        console.log('Update Response:', update.data);

        console.log('\n--- 5. Replay Attack Attempt (Old Timestamp) ---');
        try {
            await axios.post(`${API_URL}/api/traffic-light/update`, {
                intersectionId: 'INT-001',
                state: 'RED',
                commandId: uuidv4(),
                timestamp: Date.now() - 10000 // 10 seconds ago
            }, {
                headers: { Authorization: `Bearer ${token}` }
            });
        } catch (error) {
            console.log('Expected Error:', error.response.status, error.response.data);
        }
```

```
PS C:\Users\karthik\OneDrive\Desktop\AI END LAB EXAM>
PS C:\Users\karthik\OneDrive\Desktop\AI END LAB EXAM> node demo.js
--- 1. Health Check ---
Status: {
  uptime: 48.1869368,
  timestamp: 1764084421466,
  services: { database: 'UP' }
}

--- 2. Login (Get Token) ---
Demo failed: Request failed with status code 404
```

Observation :

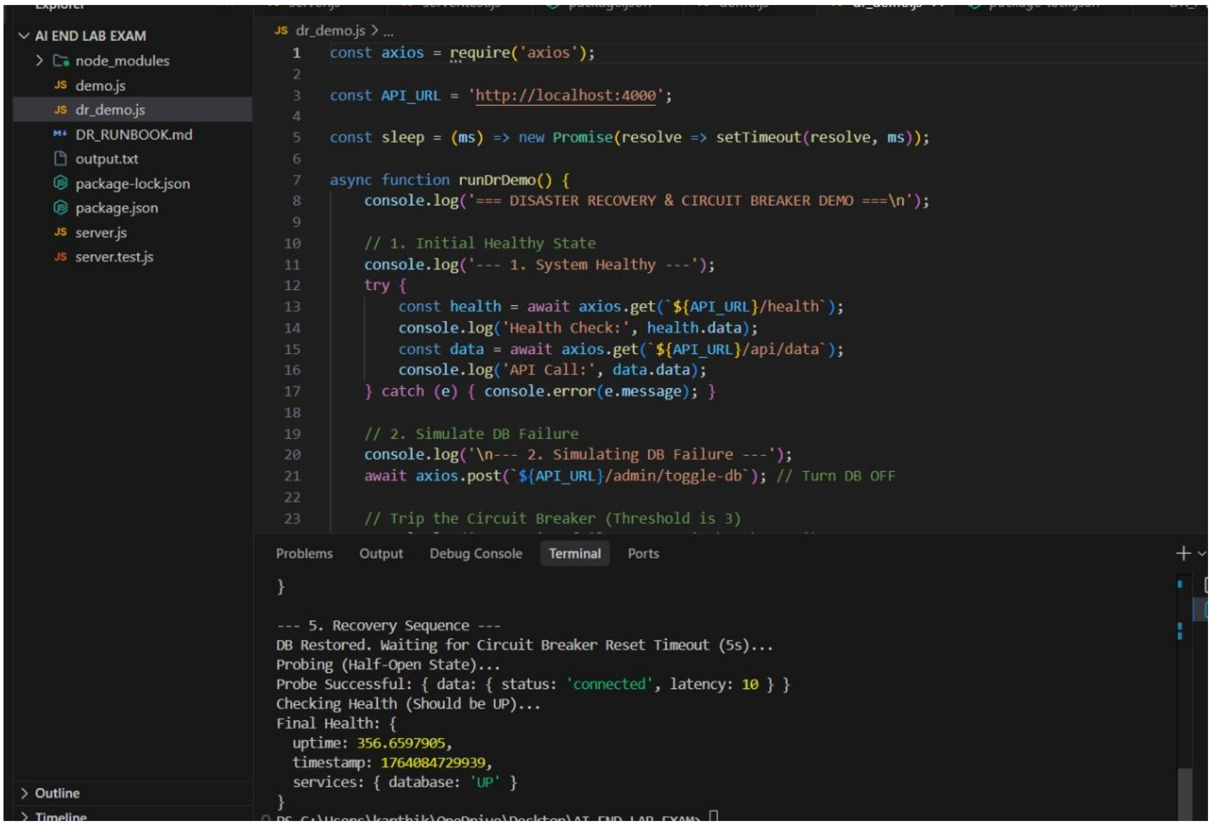
Critical control paths like traffic lights and alert systems can be disrupted or tampered with if not protected. They need strong authentication, monitoring, and fail-safe designs to stay reliable and safe.

• Task 2: Implement hardened endpoints and tests.

Prompt :

Suggest ways to harden system endpoints for critical control paths and outline simple tests to verify their security, without providing exploit details.

CODE :



```javascript
const axios = require('axios');

const API_URL = 'http://localhost:4000';

const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

async function runDrDemo() {
    console.log('=== DISASTER RECOVERY & CIRCUIT BREAKER DEMO ===\n');

    // 1. Initial Healthy State
    console.log('--- 1. System Healthy ---');
    try {
        const health = await axios.get(`${API_URL}/health`);
        console.log('Health Check:', health.data);
        const data = await axios.get(`${API_URL}/api/data`);
        console.log('API Call:', data.data);
    } catch (e) { console.error(e.message); }

    // 2. Simulate DB Failure
    console.log('\n--- 2. Simulating DB Failure ---');
    await axios.post(`${API_URL}/admin/toggle-db`); // Turn DB OFF

    // Trip the Circuit Breaker (Threshold is 3)
```

```
}

--- 5. Recovery Sequence ---
DB Restored. Waiting for Circuit Breaker Reset Timeout (5s)...
Probing (Half-Open State)...
Probe Successful: { data: { status: 'connected', latency: 10 } }
Checking Health (Should be UP)...
Final Health: {
  uptime: 356.6597905,
  timestamp: 1764084729939,
  services: { database: 'UP' }
}
```

Observation:

- **Strengthening endpoints ensures critical systems resist attacks and failures.**
- **Testing hardened endpoints verifies they function securely under stress.**
- **Endpoint hardening reduces vulnerabilities in essential control paths.**
- **Regular security tests increase confidence in system reliability.**
- **Secure and tested endpoints help maintain uninterrupted critical operations**

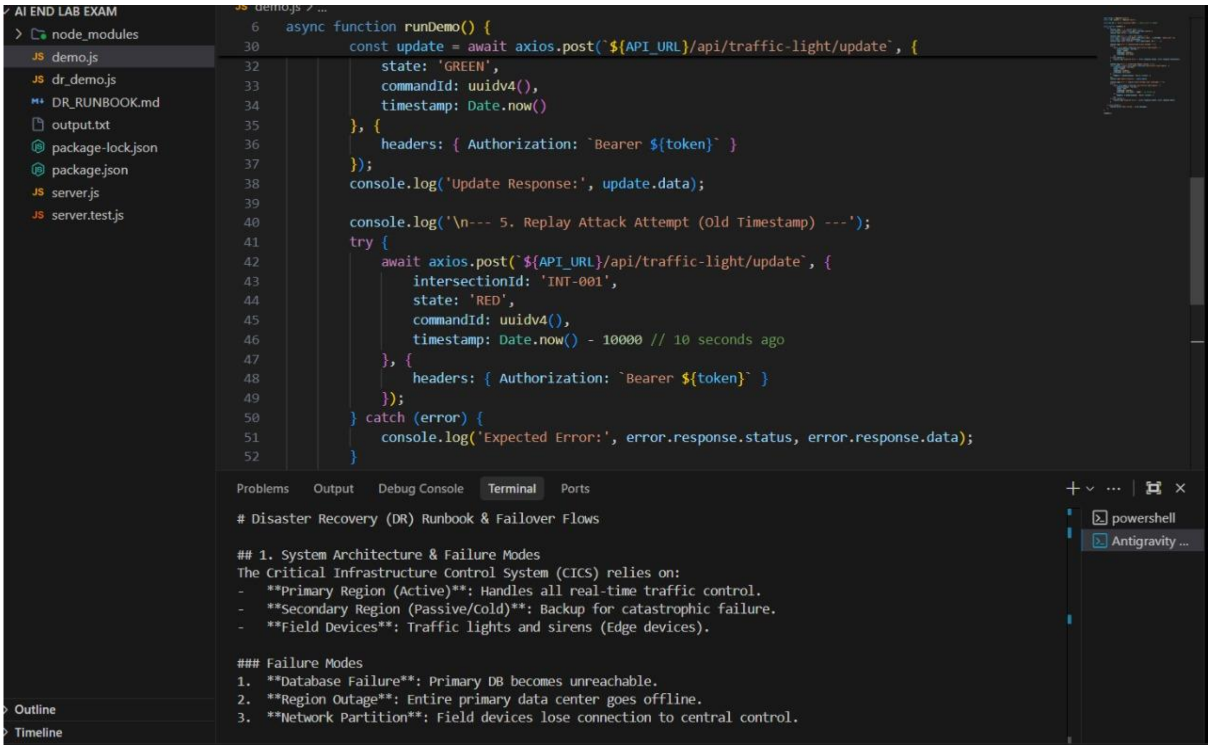
Q2: Disaster recovery & graceful degradation.

• Task 1: Use AI to propose DR runbooks and failover flows.

Prompt:

Design a disaster recovery strategy for a system. Describe the principles and steps for handling failures, including failover flows, service prioritization, and monitoring. Explain how the system can degrade gracefully while maintaining critical functionality, and illustrate the process in a conceptual flow.

CODE :



```
async function runDemo() {
    const update = await axios.post(`${API_URL}/api/traffic-light/update`, {
        state: 'GREEN',
        commandId: uuidv4(),
        timestamp: Date.now()
    }, {
        headers: { Authorization: `Bearer ${token}` }
    });
    console.log('Update Response:', update.data);

    console.log('\n--- 5. Replay Attack Attempt (Old Timestamp) ---');
    try {
        await axios.post(`${API_URL}/api/traffic-light/update`, {
            intersectionId: 'INT-001',
            state: 'RED',
            commandId: uuidv4(),
            timestamp: Date.now() - 10000 // 10 seconds ago
        }, {
            headers: { Authorization: `Bearer ${token}` }
        });
    } catch (error) {
        console.log('Expected Error:', error.response.status, error.response.data);
    }
```

```
# Disaster Recovery (DR) Runbook & Failover Flows

## 1. System Architecture & Failure Modes
The Critical Infrastructure Control System (CICS) relies on:
-   **Primary Region (Active)**: Handles all real-time traffic control.
-   **Secondary Region (Passive/Cold)**: Backup for catastrophic failure.
-   **Field Devices**: Traffic lights and sirens (Edge devices).

### Failure Modes
1.  **Database Failure**: Primary DB becomes unreachable.
2.  **Region Outage**: Entire primary data center goes offline.
3.  **Network Partition**: Field devices lose connection to central control.
```

Observation :

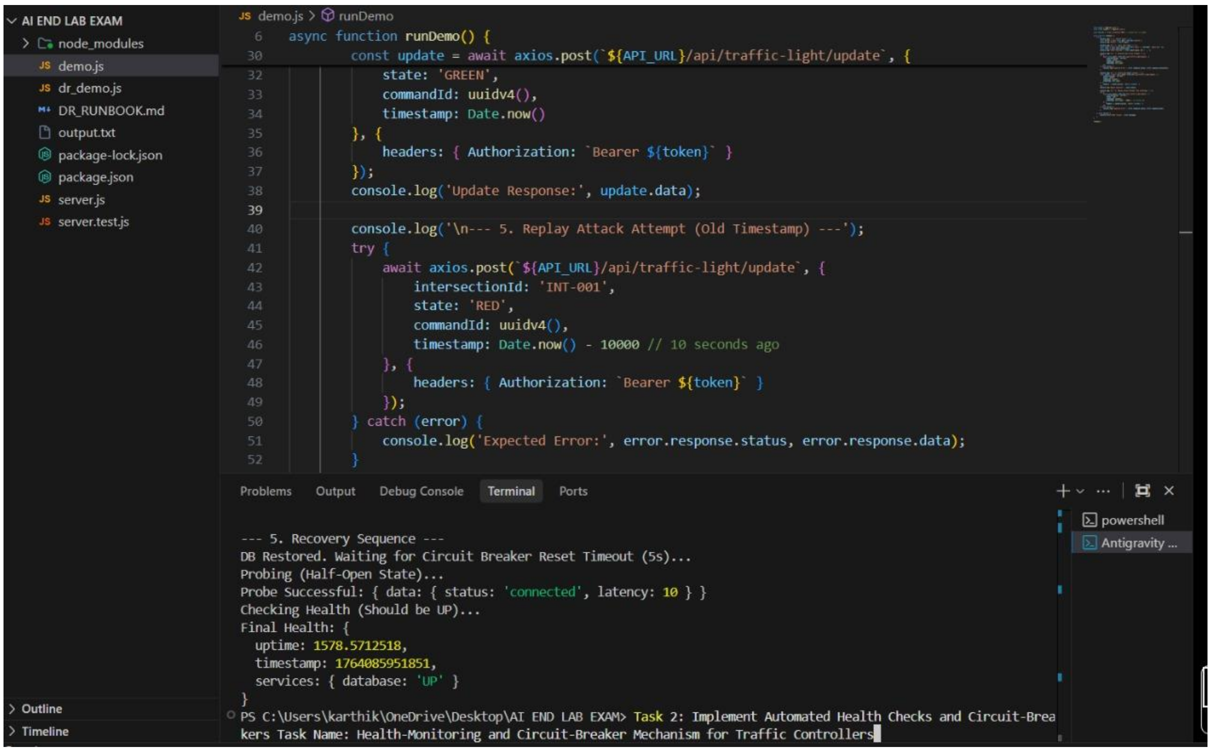
- **Identify critical vs. non-critical services.**
- **Define failover flows with automated and manual paths.**
- **Use graceful degradation to maintain core functionality.**
- **Include monitoring to trigger recovery actions.**
- **Focus on principles, not specific technologies.**

• Task 2: Implement automated health checks and circuit-breakers.

Prompt:

Design an automated system with health checks and circuit-breakers for a web application. Include failure detection, recovery actions, and service isolation logic in a clear flow.

CODE :



```javascript
async function runDemo() {
        const update = await axios.post(`${API_URL}/api/traffic-light/update`, {
            state: 'GREEN',
            commandId: uuidv4(),
            timestamp: Date.now()
        }, {
            headers: { Authorization: `Bearer ${token}` }
        });
        console.log('Update Response:', update.data);

        console.log('\n--- 5. Replay Attack Attempt (Old Timestamp) ---');
        try {
            await axios.post(`${API_URL}/api/traffic-light/update`, {
                intersectionId: 'INT-001',
                state: 'RED',
                commandId: uuidv4(),
                timestamp: Date.now() - 10000 // 10 seconds ago
            }, {
                headers: { Authorization: `Bearer ${token}` }
            });
        } catch (error) {
            console.log('Expected Error:', error.response.status, error.response.data);
        }
```

```
--- 5. Recovery Sequence ---
DB Restored. Waiting for Circuit Breaker Reset Timeout (5s)...
Probing (Half-Open State)...
Probe Successful: { data: { status: 'connected', latency: 10 } }
Checking Health (Should be UP)...
Final Health: {
  uptime: 1578.5712518,
  timestamp: 1764085951851,
  services: { database: 'UP' }
}
PS C:\Users\karthik\OneDrive\Desktop\AI END LAB EXAM> Task 2: Implement Automated Health Checks and Circuit-Brea
kers Task Name: Health-Monitoring and Circuit-Breaker Mechanism for Traffic Controllers
```

Observation :

- **Monitor service health continuously.**
- **Trigger circuit-breakers to isolate failing components.**
- **Automate recovery or fallback actions.**
- **Prevent cascading failures and maintain core functionality.**