Cloud Computing

Jackfruit Problem

Virtual Labs

Infrastructure Management

Jash S Bhalodia – PES1UG22AM075

Kirti S – PES1UG22AM084

Krithik Raman – PES1UG22AM086

Kritin Thakur – PES1UG22AM087

Source Code:

nginx.conf:

```
worker_processes 1;

events {
    worker_connections 1024;
}

http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;
    keepalive_timeout 65;

    server {
        listen 80;
        server_name localhost;

        location / {
            proxy_pass http://localhost:3000;
        }

        location /create-lab {
            proxy_pass http://localhost:3001;
        }

        location /delete-lab {
            proxy_pass http://localhost:3002;
        }
```

```
        location /monitor {
            proxy_pass http://localhost:3003;
        }

        location /performance {
            proxy_pass http://localhost:3004;
        }

        location /resource-allocation {
            proxy_pass http://localhost:3005;
        }
    }
}
```

Backend Folder:

infra/backend/create-lab-service/server.js:

```
const express = require('express');
const mysql = require('mysql2');
const cors = require('cors');
require('dotenv').config();

const app = express();

app.use(cors());
app.use(express.json());

const pool = mysql.createPool({
  host: 'localhost',
  user: process.env.DB_USER,
```

```
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
});

function findServersForLab(cpu, memory, disk, callback) {
  pool.query('SELECT * FROM servers', (err, servers) => {
    if (err) {
      return callback(err);
    }

    const suitable = servers.filter(server =>
      (server.max_cpu - server.cpu_usage) >= cpu &&
      (server.max_memory - server.memory_usage) >= memory &&
      (server.max_disk - server.disk_usage) >= disk
    );
    callback(null, suitable);
  });
}

app.post('/create-lab/create', (req, res) => {
  const name = req.body.name;
  const estimated_users = parseInt(req.body.estimated_users);
  const estimated_cpu = parseInt(req.body.estimated_cpu);
  const estimated_memory = parseInt(req.body.estimated_memory);
  const estimated_disk = parseInt(req.body.estimated_disk);

  if (
    !name ||
    isNaN(estimated_users) ||
    isNaN(estimated_cpu) ||
```

```
  isNaN(estimated_memory) ||
  isNaN(estimated_disk)
 ) {
  return res.status(400).send('Missing or invalid required fields');
 }

 const total_cpu = estimated_cpu * estimated_users;
 const total_memory = estimated_memory * estimated_users;
 const total_disk = estimated_disk * estimated_users;

 pool.query(
  'INSERT INTO labs (name, estimated_users, estimated_cpu, estimated_memory,
estimated_disk) VALUES (?, ?, ?, ?, ?)',
  [name, estimated_users, estimated_cpu, estimated_memory, estimated_disk],
  (err, result) => {
   if (err) {
    console.error('Error inserting lab:', err);
    return res.status(500).send({ message: 'Error creating lab' });
   }

   const labId = result.insertId;
   console.log(`Lab created with ID: ${labId}`);

   pool.query('SELECT * FROM servers', (err, servers) => {
    if (err) {
     console.error('Error fetching servers:', err);
     return res.status(500).send('Error fetching servers');
    }

    let cpuLeft = total_cpu;
    let memLeft = total_memory;
```

```
let diskLeft = total_disk;
const allocs = [];

for (let s of servers) {
  if (cpuLeft <= 0 && memLeft <= 0 && diskLeft <= 0) break;

  const availableCPU = Math.max(0, s.max_cpu - s.cpu_usage);
  const availableMem = Math.max(0, s.max_memory - s.memory_usage);
  const availableDisk = Math.max(0, s.max_disk - s.disk_usage);

  const cpuToAlloc = Math.min(cpuLeft, availableCPU);
  const memToAlloc = Math.min(memLeft, availableMem);
  const diskToAlloc = Math.min(diskLeft, availableDisk);

  if (cpuToAlloc > 0 || memToAlloc > 0 || diskToAlloc > 0) {
    allocs.push({
      server_id: s.id,
      cpu: cpuToAlloc,
      memory: memToAlloc,
      disk: diskToAlloc
    });

    cpuLeft -= cpuToAlloc;
    memLeft -= memToAlloc;
    diskLeft -= diskToAlloc;
  }
}

if (cpuLeft > 0 || memLeft > 0 || diskLeft > 0) {
  console.error('Not enough server capacity for new lab');
```

```
      return res.status(400).send('Not enough server capacity for new lab');
    }

    function performAllocations(index = 0) {
      if (index >= allocs.length) {
        return res.send({
          message: 'Lab created and resources allocated',
          labId,
          allocations: allocs
        });
      }

      const alloc = allocs[index];
      pool.query(
        'INSERT INTO lab_server (lab_id, server_id, cpu_allocated, memory_allocated, disk_allocated) VALUES (?, ?, ?, ?, ?)',
        [labId, alloc.server_id, alloc.cpu, alloc.memory, alloc.disk],
        (err) => {
          if (err) return res.status(500).send('Error inserting lab_server row');

          pool.query(
            'UPDATE servers SET cpu_usage = cpu_usage + ?, memory_usage = memory_usage + ?, disk_usage = disk_usage + ? WHERE id = ?',
            [alloc.cpu, alloc.memory, alloc.disk, alloc.server_id],
            (err) => {
              if (err) return res.status(500).send('Error updating server usage');
              performAllocations(index + 1);
            }
          );
        }
      );
```

```
        }

        performAllocations();
      });
    }
  );
});

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Create Lab Service is running on port ${PORT}`);
});
```

infra/backend/delete-lab-service/server.js:

```
const express = require('express');
const mysql = require('mysql2');
const cors = require('cors');
require('dotenv').config();

const app = express();
app.use(cors());
app.use(express.json());

const pool = mysql.createPool({
  host: 'localhost',
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
```

```
});

app.get('/delete-lab', async (req, res) => {
  try {
    const [labs] = await pool.promise().query('SELECT id, name FROM labs');
    res.json(labs);
  } catch (err) {
    console.error('Error fetching labs:', err);
    res.status(500).send('Failed to fetch labs');
  }
});

app.delete('/delete/:labId', async (req, res) => {
  const labId = parseInt(req.params.labId);
  const conn = await pool.promise().getConnection();

  try {
    await conn.beginTransaction();

    const [labResults] = await conn.query('SELECT * FROM labs WHERE id = ?', [labId]);
    if (labResults.length === 0) {
      conn.release();
      return res.status(404).send('Lab not found');
    }

    const [allocations] = await conn.query(
      'SELECT server_id, cpu_allocated, memory_allocated, disk_allocated FROM lab_server
WHERE lab_id = ?',
      [labId]
    );
```

```
    for (const alloc of allocations) {
      const [serverRows] = await conn.query('SELECT * FROM servers WHERE id = ?', [alloc.server_id]);
      const server = serverRows[0];

      const newCpu = Math.max(server.cpu_usage - alloc.cpu_allocated, 0);
      const newMemory = Math.max(server.memory_usage - alloc.memory_allocated, 0);
      const newDisk = Math.max(server.disk_usage - alloc.disk_allocated, 0);

      console.log(`Server ${server.id} usage BEFORE: CPU ${server.cpu_usage}, MEM ${server.memory_usage}, DISK ${server.disk_usage}`);
      console.log(`Deallocating: CPU ${alloc.cpu_allocated}, MEM ${alloc.memory_allocated}, DISK ${alloc.disk_allocated}`);
      console.log(`Server ${server.id} usage AFTER:  CPU ${newCpu}, MEM ${newMemory}, DISK ${newDisk}`);

      await conn.query(
        `UPDATE servers SET
          cpu_usage = ?,
          memory_usage = ?,
          disk_usage = ?
        WHERE id = ?`,
        [newCpu, newMemory, newDisk, alloc.server_id]
      );
    }

    await conn.query('DELETE FROM lab_server WHERE lab_id = ?', [labId]);
    await conn.query(
      'UPDATE lab_access_log SET is_active = FALSE WHERE lab_id = ?',
      [labId]
    );
    await conn.query('DELETE FROM labs WHERE id = ?', [labId]);
```

```
    await conn.commit();
    res.send({ message: 'Lab deleted and server resources freed' });
  } catch (err) {
    await conn.rollback();
    console.error('Error deleting lab:', err);
    res.status(500).send('Internal server error');
  } finally {
    conn.release();
  }
});

const PORT = 3002;
app.listen(PORT, () => {
  console.log(`Delete lab service running on port ${PORT}`);
});
```

infra/backend/lab-monitoring-service/server.js:

```
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2');
require('dotenv').config();

const app = express();
const PORT = 3003;

app.use(cors());
app.use(express.json());
```

```
const pool = mysql.createPool({
  host: 'localhost',
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
});

app.get('/monitor/labs', async (req, res) => {
  try {
    const [labs] = await pool.promise().query('SELECT id, name FROM labs');
    res.json(labs);
  } catch (err) {
    console.error('Error fetching labs:', err);
    res.status(500).send('Failed to fetch labs');
  }
});

app.post('/simulate/:labId', (req, res) => {
  const labId = parseInt(req.params.labId);
  const { active_users, duration_minutes } = req.body;

  pool.query('SELECT * FROM labs WHERE id = ?', [labId], (err, labs) => {
    if (err || labs.length === 0) return res.status(404).send('Lab not found');

    const lab = labs[0];

    const timeFactor = duration_minutes / 10;
    const factor = (active_users * timeFactor) / lab.estimated_users;

    const cpu = lab.estimated_cpu * factor;
```

```
    const memory = lab.estimated_memory * factor;
    const disk = lab.estimated_disk * factor;

    // Insert lab usage log
    pool.query(
      `INSERT INTO lab_access_log (lab_id, accessed_at, duration_minutes, user_count)
      VALUES (?, NOW(), ?, ?)`,
      [labId, duration_minutes, active_users],
      insertErr => {
        if (insertErr) {
          console.error('Error inserting lab access log:', insertErr);
          return res.status(500).send(insertErr);
        }

        console.log('Usage simulated for lab', labId);

        // Update average utilization
        pool.query(
          `UPDATE labs
          SET avg_users = (SELECT AVG(user_count) FROM lab_access_log WHERE lab_id =
?),
              avg_time = (SELECT AVG(duration_minutes) FROM lab_access_log WHERE
lab_id = ?)
          WHERE id = ?`,
          [labId, labId, labId],
          updateErr => {
            if (updateErr) {
              console.error('Error updating lab utilization:', updateErr);
              return res.status(500).send(updateErr);
            }
            console.log('Lab utilization updated for lab', labId);
```

```
        }
      );

      // Update popular metrics
      pool.query(
        `UPDATE labs
        SET total_sessions = (SELECT COUNT(*) FROM lab_access_log WHERE lab_id =
?),
            total_user_minutes = (SELECT SUM(duration_minutes * user_count) FROM
lab_access_log WHERE lab_id = ?)
        WHERE id = ?`,
        [labId, labId, labId],
        popularLabsErr => {
          if (popularLabsErr) {
            console.error('Error updating popular labs metrics:', popularLabsErr);
            return res.status(500).send(popularLabsErr);
          }
          console.log('Popular labs updated for lab', labId);

          res.send({
            message: 'Usage simulated and metrics updated',
            usage: {
              cpu: cpu.toFixed(2),
              memory: memory.toFixed(2),
              disk: disk.toFixed(2)
            }
          });
        }
      );
    }
  );
```

```
  });
});

// Get average usage stats for a lab
app.get('/usage/:labId', (req, res) => {
  const labId = parseInt(req.params.labId);

  pool.query(
    `SELECT AVG(user_count) as avg_users, AVG(duration_minutes) as avg_time
     FROM lab_access_log WHERE lab_id = ?`,
    [labId],
    (err, stats) => {
      if (err) return res.status(500).send(err);
      res.json(stats[0]);
    }
  );
});

app.get('/monitor/labs/over-under-utilized', (req, res) => {
  pool.query(`
    SELECT
      l.id,
      l.name,
      l.estimated_users,
      COALESCE(AVG(a.user_count), 0) AS avg_users,
      COALESCE(AVG(a.duration_minutes), 0) AS avg_time
    FROM labs l
    LEFT JOIN lab_access_log a ON l.id = a.lab_id
    GROUP BY l.id, l.name, l.estimated_users
  `, (err, results) => {
```

```
if (err) return res.status(500).send(err);

const thresholds = { upper: 1.25, lower: 0.75 };

const analysis = results.map(row => {
  const avgUsers = Number(row.avg_users);
  const avgTime = Number(row.avg_time);
  const estimatedUsers = row.estimated_users;

  // Normalize time-aware usage to 10-minute basis
  const normalizedUsage = avgUsers * (avgTime / 10);
  const expectedUsage = estimatedUsers; // assumed per 10 minutes

  const ratio = normalizedUsage / expectedUsage;

  let status = 'normal';
  if (ratio > thresholds.upper) status = 'over-utilized';
  else if (ratio < thresholds.lower) status = 'under-utilized';

  return {
    lab_id: row.id,
    name: row.name,
    estimated_users: estimatedUsers,
    avg_users: Number.isFinite(avgUsers) ? parseFloat(avgUsers.toFixed(2)) : null,
    avg_time: Number.isFinite(avgTime) ? parseFloat(avgTime.toFixed(2)) : null,
    normalized_usage: parseFloat(normalizedUsage.toFixed(2)),
    ratio: parseFloat(ratio.toFixed(2)),
    status
  };
});
```

```
    res.json(analysis);
  });
});

// GET /monitor/labs/popular
app.get('/monitor/labs/popular', (req, res) => {
  pool.query(`
    SELECT l.id, l.name,
           COUNT(a.id) AS total_sessions,
           COALESCE(SUM(a.duration_minutes * a.user_count), 0) AS total_user_minutes
    FROM labs l
    LEFT JOIN lab_access_log a ON l.id = a.lab_id
    GROUP BY l.id, l.name
    ORDER BY total_user_minutes DESC
  `, (err, results) => {
    if (err) return res.status(500).send(err);

    res.json(results.map(row => ({
      lab_id: row.id,
      name: row.name,
      total_sessions: row.total_sessions,
      total_user_minutes: row.total_user_minutes || 0
    })));
  });
});

app.listen(PORT, () => {
  console.log(`Lab Monitoring Service running at http://localhost:${PORT}`);
});
```

infra/backend/performance-service/server.js:

```
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2');
const dotenv = require('dotenv');
const moment = require('moment');

// Load environment variables from .env file
dotenv.config();

// Initialize express app
const app = express();

// CORS configuration
app.use(cors());

const pool = mysql.createPool({
  host: 'localhost',
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
});

// Helper to simulate load based on access logs
function simulateLoad(estimatedCpu, estimatedMemory, estimatedDisk, estimatedUsers, userCount, duration) {
  const timeFactor = duration / 10;
  const rawFactor = (userCount * timeFactor) / (estimatedUsers || 1); // Avoid divide by 0
  const factor = Math.min(rawFactor, 3); // Cap at 3x load
```

```
  const variance = 1 + (Math.random() * 0.1 - 0.05); // ±5%

  return {
    cpu: estimatedCpu * factor * variance,
    memory: estimatedMemory * factor * variance,
    disk: estimatedDisk * factor * variance
  };
}

// Function to update stats based on access logs
function updateStats() {
  pool.query('SELECT * FROM servers', (err, servers) => {
    if (err) return console.error(err);

    servers.forEach(server => {
      // Get the labs associated with this server
      pool.query(`
        SELECT l.*, lab_server.server_id
        FROM labs l
        JOIN lab_server ON l.id = lab_server.lab_id
        WHERE lab_server.server_id = ?
      `, [server.id], (err, labs) => {
        if (err) return console.error(err);

        let totalCpu = 0, totalMem = 0, totalDisk = 0;

        labs.forEach(lab => {
          // Get the lab access logs for this lab
          pool.query(`
            SELECT * FROM lab_access_log
```

```
  WHERE lab_id = ?
  ORDER BY accessed_at
`, [lab.id], (err, accessLogs) => {
  if (err) return console.error(err);

  let cpuUsage = 0, memoryUsage = 0, diskUsage = 0;

  accessLogs.forEach(log => {
    const { user_count, duration_minutes } = log;
    const usage = simulateLoad(
      lab.estimated_cpu,
      lab.estimated_memory,
      lab.estimated_disk,
      lab.estimated_users,
      user_count,
      duration_minutes
    );

    cpuUsage += usage.cpu;
    memoryUsage += usage.memory;
    diskUsage += usage.disk;
  });

  // Ensure usage doesn't exceed server's max capacity
  totalCpu += Math.min(cpuUsage, server.max_cpu);
  totalMem += Math.min(memoryUsage, server.max_memory);
  totalDisk += Math.min(diskUsage, server.max_disk);

  // Update server stats and insert into server_stats
  pool.query(`
```

```
            UPDATE servers
            SET cpu_usage = ?, memory_usage = ?, disk_usage = ?, last_checked = NOW()
            WHERE id = ?
          `, [totalCpu, totalMem, totalDisk, server.id]);

          pool.query(`
            INSERT INTO server_stats (server_id, cpu_usage, memory_usage, disk_usage, recorded_at)
            VALUES (?, ?, ?, ?, NOW())
          `, [server.id, totalCpu, totalMem, totalDisk]);
        });
      });
    });
  });
}

// Update stats every 10 minutes
setInterval(updateStats, 10 * 60 * 1000); // every 10 mins
updateStats();

// Route to get stats for all servers
app.get('/performance/servers/stats', (req, res) => {

  pool.query('SELECT * FROM server_stats', (err, results) => {
    if (err) return res.status(500).send(err);
    res.json(results);
  });
});

// Route to get server stats for a specific server
```

```
app.get('/performance/servers/:id/stats', (req, res) => {

  const { id } = req.params;

  pool.query('SELECT * FROM server_stats WHERE server_id = ?', [id], (err, results) => {
    if (err) return res.status(500).send(err);
    res.json(results);
  });
});

app.get('/performance/servers/:id/peaks', (req, res) => {
  const { id } = req.params;
  pool.query(`
    SELECT accessed_at, SUM(user_count) AS total_users
    FROM lab_access_log
    JOIN labs ON labs.id = lab_access_log.lab_id
    JOIN lab_server ON labs.id = lab_server.lab_id
    WHERE lab_server.server_id = ?
    GROUP BY accessed_at
    ORDER BY accessed_at
  `, [id], (err, results) => {
    if (err) return res.status(500).send(err);

    const peakTimes = results.map(log => ({
      time: moment(log.accessed_at).format('YYYY-MM-DD HH:mm:ss'),
      users: log.total_users
    }));

    res.json(peakTimes);
  });
```

```
});

const PORT = process.env.PORT || 3004;
app.listen(PORT, () => {
  console.log(`Performance Service is running at http://localhost:${PORT}`);
});
```

infra/backend/resourceAllocation/server.js:

```
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2');
require('dotenv').config();

const app = express();
const PORT = 3005;

app.use(cors());
app.use(express.json());

const pool = mysql.createPool({
  host: 'localhost',
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
});

function getAvailable(server) {
  return {
    cpu: server.max_cpu - server.cpu_usage,
```

```
    memory: server.max_memory - server.memory_usage,
    disk: server.max_disk - server.disk_usage
  };
}

app.get('/labs', (req, res) => {
  pool.query('SELECT * FROM labs', (err, results) => {
    if (err) {
      console.error('Error fetching labs:', err);
      return res.status(500).send(err);
    }
    res.send(results);
  });
});

app.get('/servers', (req, res) => {
  pool.query('SELECT * FROM servers', (err, results) => {
    if (err) {
      console.error('Error fetching servers:', err);
      return res.status(500).send(err);
    }
    res.send(results);
  });
});

app.get('/allocations', (req, res) => {
  const sql = `SELECT ls.id, l.name AS lab_name, s.id AS server_id
               FROM lab_server ls
               JOIN labs l ON ls.lab_id = l.id
               JOIN servers s ON ls.server_id = s.id`;
```

```
  pool.query(sql, (err, results) => {
    if (err) {
      console.error('Error fetching allocations:', err);
      return res.status(500).send(err);
    }
    res.send(results);
  });
});

app.post('/servers', (req, res) => {
  const { ip_address, max_cpu, max_memory, max_disk } = req.body;
  console.log('Adding new server:', req.body);

  if (!ip_address || !max_cpu || !max_memory || !max_disk) {
    return res.status(400).send('Missing required fields');
  }

  pool.query(
    'INSERT INTO servers (ip_address, max_cpu, memory_usage, max_memory, disk_usage,
max_disk, cpu_usage) VALUES (?, ?, 0, ?, 0, ?, 0)',
    [ip_address, max_cpu, max_memory, max_disk],
    (err, result) => {
      if (err) {
        console.error('Error adding server:', err);
        return res.status(500).send(err);
      }
      res.send({ message: 'Server added', serverId: result.insertId });
    }
  );
});
```

```
app.patch('/labs/:id', (req, res) => {
  const labId = req.params.id;
  const { per_user_cpu, per_user_memory, per_user_disk, estimated_users } = req.body;

  pool.query('SELECT * FROM labs WHERE id = ?', [labId], (err, labResults) => {
    if (err || labResults.length === 0) return res.status(500).send('Lab not found');

    pool.query(
      'UPDATE labs SET estimated_cpu=?, estimated_memory=?, estimated_disk=?,
estimated_users=? WHERE id=?',
      [per_user_cpu, per_user_memory, per_user_disk, estimated_users, labId],
      (err) => {
        if (err) return res.status(500).send('Error updating lab');

        pool.query('SELECT * FROM lab_server WHERE lab_id = ?', [labId], (err, allocations)
=> {
          if (err) return res.status(500).send('Error fetching old allocations');

          const revertTasks = allocations.map(alloc => {
            return new Promise((resolve, reject) => {
              pool.query(
                'UPDATE servers SET cpu_usage = cpu_usage - ?, memory_usage =
memory_usage - ?, disk_usage = disk_usage - ? WHERE id = ?',
                [alloc.cpu_allocated, alloc.memory_allocated, alloc.disk_allocated,
alloc.server_id],
                (err) => err ? reject(err) : resolve()
              );
            });
          });

          Promise.all(revertTasks).then(() => {
            pool.query('DELETE FROM lab_server WHERE lab_id = ?', [labId], (err) => {
```

```
        if (err) return res.status(500).send('Error deleting old lab_server rows');

        pool.query('SELECT * FROM servers', (err, servers) => {
          if (err) return res.status(500).send('Error fetching servers');

          let cpuLeft = estimated_users * per_user_cpu;
          let memLeft = estimated_users * per_user_memory;
          let diskLeft = estimated_users * per_user_disk;

          const allocs = [];

          for (let s of servers) {
            if (cpuLeft <= 0 && memLeft <= 0 && diskLeft <= 0) break;

            const availCPU = Math.max(0, s.max_cpu - s.cpu_usage);
            const availMem = Math.max(0, s.max_memory - s.memory_usage);
            const availDisk = Math.max(0, s.max_disk - s.disk_usage);

            const cpuToAlloc = Math.floor(Math.min(cpuLeft, availCPU));
            const memToAlloc = Math.floor(Math.min(memLeft, availMem));
            const diskToAlloc = Math.floor(Math.min(diskLeft, availDisk));

            if (cpuToAlloc > 0 || memToAlloc > 0 || diskToAlloc > 0) {
              allocs.push({ server_id: s.id, cpu: cpuToAlloc, memory: memToAlloc, disk:
diskToAlloc });
              cpuLeft -= cpuToAlloc;
              memLeft -= memToAlloc;
              diskLeft -= diskToAlloc;
            }
          }
```

```
            if (cpuLeft > 0 || memLeft > 0 || diskLeft > 0) {
              return res.status(400).send('Not enough server capacity for scaling');
            }

            const applyTasks = allocs.map(a => {
              return new Promise((resolve, reject) => {
                pool.query(
                  'INSERT INTO lab_server (lab_id, server_id, cpu_allocated,
memory_allocated, disk_allocated) VALUES (?, ?, ?, ?, ?)',
                  [labId, a.server_id, a.cpu, a.memory, a.disk],
                  (err) => {
                    if (err) return reject(err);
                    pool.query(
                      'UPDATE servers SET cpu_usage = cpu_usage + ?, memory_usage =
memory_usage + ?, disk_usage = disk_usage + ? WHERE id = ?',
                      [a.cpu, a.memory, a.disk, a.server_id],
                      (err) => err ? reject(err) : resolve()
                    );
                  }
                );
              });
            });

            Promise.all(applyTasks)
              .then(() => res.send({ message: `Lab ${labId} scaled and updated successfully.`
}))
              .catch(err => {
                console.error('Error applying new allocations:', err);
                res.status(500).send('Failed during allocation');
              });
          });
```

```
          });
        }).catch(err => {
          console.error('Error reverting previous allocations:', err);
          res.status(500).send('Failed during revert');
        });
      });
    }
  );
 });
});

app.get('/fix-lab-server-duplicates', (req, res) => {
 const query = `
  DELETE ls1 FROM lab_server ls1
  JOIN lab_server ls2
  ON ls1.lab_id = ls2.lab_id AND ls1.server_id = ls2.server_id AND ls1.id > ls2.id;
 `;

 pool.query(query, (err) => {
  if (err) {
   console.error(err);
   return res.status(500).send('Error removing duplicates');
  }

  pool.query('ALTER TABLE lab_server ADD UNIQUE KEY unique_lab_server (lab_id,
server_id)', (err) => {
   if (err) {
    console.error(err);
    return res.status(500).send('Could not enforce unique constraint');
   }
```

```
      res.send('Duplicates removed and unique constraint added.');
    });
  });
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

app.listen(PORT, () => console.log(`Resource Allocation Service running on port ${PORT}`));
```

Frontend Folder:

infra/frontend/src/App.js:

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Navbar from './components/Navbar';
import HomePage from './pages/HomePage';
import CreateLabPage from './pages/CreateLabPage';
import DeleteLabPage from './pages/DeleteLabPage';
import MonitorPage from './pages/MonitorPage';
import PerformancePage from './pages/PerformancePage';
import ResourceAllocationPage from './pages/ResourceAllocationPage';

function App() {
  return (
    <Router>
```

```
      <Navbar />
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/lab-creation" element={<CreateLabPage />} />
        <Route path="/lab-deletion" element={<DeleteLabPage />} />
        <Route path="/lab-monitoring" element={<MonitorPage />} />
        <Route path="/server-performance" element={<PerformancePage />} />
        <Route path="/resource-allocation" element={<ResourceAllocationPage />} />
      </Routes>
    </Router>
  );
}

export default App;
```

infra/frontend/src/pages/HomePage.js:

```
import React from 'react';
import { Link } from 'react-router-dom';

const HomePage = () => {
  const containerStyle = {
    minHeight: '100vh',
    backgroundColor: '#f3f4f6',
    padding: '2rem'
  };

  const titleStyle = {
    fontSize: '2.5rem',
    fontWeight: 'bold',
```

```
  marginBottom: '2rem',
  textAlign: 'center'

};

const gridStyle = {
  display: 'grid',
  gridTemplateColumns: 'repeat(auto-fit, minmax(280px, 1fr))',
  gap: '1.5rem',
  maxWidth: '800px',
  margin: '0 auto'
};

const cardStyle = (bgColor, hoverColor) => ({
  backgroundColor: bgColor,
  color: '#fff',
  padding: '1.25rem 1.5rem',
  borderRadius: '8px',
  textDecoration: 'none',
  boxShadow: '0 4px 10px rgba(0, 0, 0, 0.1)',
  textAlign: 'center',
  transition: 'background-color 0.3s',
  fontSize: '1.1rem',
  fontWeight: '500'
});

return (
  <div style={containerStyle}>
    <h1 style={titleStyle}>Lab Infrastructure Management</h1>
```

```
      <div style={gridStyle}>
        <Link to="/lab-creation" style={cardStyle('#444242')}>
          ➕ Create Lab
        </Link>

        <Link to="/lab-deletion" style={cardStyle('#444242')}>
          🗑️ Delete Lab
        </Link>

        <Link to="/lab-monitoring" style={cardStyle('#444242')}>
          📊 Monitor Labs
        </Link>

        <Link to="/server-performance" style={cardStyle('#444242')}>
          🖥️ Server Performance
        </Link>

        <Link to="/resource-allocation" style={cardStyle('#444242')}>
          ⚙️ Resource Allocation
        </Link>
      </div>
    </div>
  );
};

export default HomePage;
```

infra/frontend/src/pages/CreateLabPage.js:

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

const CreateLabPage = () => {
  const [form, setForm] = useState({
    name: '',
    estimated_users: 10,
    estimated_cpu: 10,
    estimated_memory: 20,
    estimated_disk: 5
  });

  const handleChange = e => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async e => {
    e.preventDefault();
    try {
      await axios.post('http://localhost:3001/create-lab/create', form);
      alert('Lab created and resources allocated!');
    } catch (err) {
      console.error(err);
      alert(err?.response?.data?.message || 'Error creating lab');
    }
  };

  return (
    <div style={{ maxWidth: '500px', margin: '40px auto', padding: '20px' }}>
```

```
    <form
      onSubmit={handleSubmit}
      style={{
        padding: '30px',
        border: '1px solid #ccc',
        borderRadius: '8px',
        backgroundColor: '#fff',
        boxShadow: '0 2px 8px rgba(0, 0, 0, 0.1)'
      }}
    >
      <h2 style={{ fontSize: '24px', fontWeight: 'bold', marginBottom: '30px', textAlign: 'center' }}>
        Create New Lab
      </h2>

      {['name', 'estimated_users', 'estimated_cpu', 'estimated_memory', 'estimated_disk'].map(field => (
        <div key={field} style={{ marginBottom: '20px' }}>
          <label
            htmlFor={field}
            style={{ display: 'block', marginBottom: '8px', fontWeight: '500', color: '#333' }}
          >
            {field.replace(/_/g, ' ').toUpperCase()}
          </label>
          <input
            type={field === 'name' ? 'text' : 'number'}
            name={field}
            value={form[field]}
            onChange={handleChange}
            style={{
              width: '100%',
```

```
            padding: '10px',
            border: '1px solid #ccc',
            borderRadius: '4px',
            fontSize: '14px',
            outline: 'none',
            boxSizing: 'border-box'
          }}
        />
      </div>
    ))}

    <button
      type="submit"
      style={{
        width: '100%',
        marginTop: '20px',
        backgroundColor: '#2563eb',
        color: '#fff',
        padding: '12px',
        fontSize: '16px',
        borderRadius: '6px',
        border: 'none',
        cursor: 'pointer'
      }}
    >
      Create Lab
    </button>
  </form>
 </div>
);
```

```
};

export default CreateLabPage;
```

infra/frontend/src/pages/DeleteLabPage.js:

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

const DeleteLabPage = () => {
  const [labs, setLabs] = useState([]);
  const [labId, setLabId] = useState('');

  useEffect(() => {
    axios.get('http://localhost:3002/delete-lab')
      .then(res => setLabs(res.data))
      .catch(err => console.error("Failed to fetch labs:", err));
  }, []);

  const handleDelete = async () => {
    if (!labId) return alert('Please select a lab to delete');
    try {
      await axios.delete(`http://localhost:3002/delete/${labId}`);
      alert('Lab deleted');
      setLabs(labs.filter(lab => lab.id !== labId)); // remove from dropdown
      setLabId('');
    } catch (err) {
      console.error(err);
      alert('Failed to delete lab');
    }
```

```
};

return (
  <div style={{ padding: '2rem', maxWidth: '500px', margin: '0 auto' }}>
    <h2 style={{ fontSize: '22px', marginBottom: '1rem' }}>Delete Lab</h2>

    <div style={{ marginBottom: '1rem' }}>
      <select
        value={labId}
        onChange={e => setLabId(e.target.value)}
        style={{
          width: '100%',
          padding: '10px',
          borderRadius: '4px',
          border: '1px solid #ccc'
        }}
      >
        <option value="">Select a lab</option>
        {labs.map(lab => (
          <option key={lab.id} value={lab.id}>{lab.name}</option>
        ))}
      </select>
    </div>

    <button
      onClick={handleDelete}
      style={{
        backgroundColor: '#dc2626',
        color: 'white',
        padding: '10px 16px',
```

```
          border: 'none',
          borderRadius: '4px',
          cursor: 'pointer'
        }}
      >
        Delete Lab
      </button>
    </div>
  );
};

export default DeleteLabPage;
```

infra/frontend/src/pages/MonitorPage.js:

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

const MonitorPage = () => {
  const [labs, setLabs] = useState([]);
  const [labId, setLabId] = useState('');
  const [activeUsers, setActiveUsers] = useState('');
  const [duration, setDuration] = useState('');
  const [simulationResult, setSimulationResult] = useState(null);

  const [utilizationData, setUtilizationData] = useState([]);
  const [popularLabs, setPopularLabs] = useState([]);

  useEffect(() => {
    axios.get('http://localhost:3003/monitor/labs')
```

```
    .then(res => setLabs(res.data))
    .catch(err => console.error("Error fetching labs:", err));

  fetchMonitoringData();
}, []);

const fetchMonitoringData = () => {
  axios.get('http://localhost:3003/monitor/labs/over-under-utilized')
    .then(res => setUtilizationData(res.data))
    .catch(err => console.error("Error fetching utilization data:", err));

  axios.get('http://localhost:3003/monitor/labs/popular')
    .then(res => setPopularLabs(res.data))
    .catch(err => console.error("Error fetching popular labs:", err));
};

const handleSimulate = async (e) => {
  e.preventDefault();
  try {
    const res = await axios.post(`http://localhost:3003/simulate/${labId}`, {
      active_users: Number(activeUsers),
      duration_minutes: Number(duration)
    });

    setSimulationResult(res.data.usage);
    fetchMonitoringData();
  } catch (err) {
    console.error("Simulation failed:", err);
  }
};
```

```
  const tableStyle = {
    width: '100%',
    borderCollapse: 'collapse',
    marginTop: '10px',
    marginBottom: '20px',
    boxShadow: '0 0 5px rgba(0,0,0,0.1)',
    backgroundColor: '#ffffff'
  };

  const thTdStyle = {
    border: '1px solid #ccc',
    padding: '12px',
    textAlign: 'left'
  };

  const sectionStyle = {
    marginBottom: '2rem',
    padding: '1.5rem',
    border: '1px solid #ddd',
    borderRadius: '6px',
    backgroundColor: '#f9f9f9'
  };

  return (
    <div style={{ padding: '2rem', maxWidth: '900px', margin: '0 auto' }}>
      <h2 style={{ fontSize: '24px', marginBottom: '1rem' }}>Lab Monitoring Dashboard</h2>

      <form onSubmit={handleSimulate} style={{ ...sectionStyle, padding: '20px', border: '1px
solid #ddd', borderRadius: '6px' }}>
        <h3 style={{ fontSize: '20px', marginBottom: '16px' }}>Simulate Lab Usage</h3>
```

```
      <div style={{ marginBottom: '12px' }}>
        <label style={{ display: 'block', marginBottom: '4px' }}>Select Lab:</label>
        <select value={labId} onChange={e => setLabId(e.target.value)} required style={{
width: '100%', padding: '8px' }}>
          <option value="">-- Choose a lab --</option>
          {labs.map(lab => (
            <option key={lab.id} value={lab.id}>{lab.name}</option>
          ))}
        </select>
      </div>

      <div style={{ marginBottom: '12px' }}>
        <label style={{ display: 'block', marginBottom: '4px' }}>Active Users:</label>
        <input
          type="number"
          value={activeUsers}
          onChange={e => setActiveUsers(e.target.value)}
          required
          style={{ width: '98%', padding: '8px' }}
        />
      </div>

      <div style={{ marginBottom: '12px' }}>
        <label style={{ display: 'block', marginBottom: '4px' }}>Duration (minutes):</label>
        <input
          type="number"
          value={duration}
          onChange={e => setDuration(e.target.value)}
          required
          style={{ width: '98%', padding: '8px' }}
```

```
          />
        </div>

        <button type="submit" style={{ padding: '10px 16px', backgroundColor: '#2563eb', color: '#fff', border: 'none', borderRadius: '4px', cursor: 'pointer' }}>
          Simulate
        </button>
      </form>

      {simulationResult && (
        <div style={sectionStyle}>
          <h4 style={{ fontSize: '18px', marginBottom: '8px' }}>Simulated Resource Usage</h4>
          <ul style={{ paddingLeft: '20px' }}>
            <li>CPU: {simulationResult.cpu}</li>
            <li>Memory: {simulationResult.memory}</li>
            <li>Disk: {simulationResult.disk}</li>
          </ul>
        </div>
      )}

      <div style={sectionStyle}>
        <h3 style={{ fontSize: '20px' }}>Lab Utilization</h3>
        <table style={tableStyle}>
          <thead>
            <tr>
              <th style={thTdStyle}>Lab</th>
              <th style={thTdStyle}>Estimated Users</th>
              <th style={thTdStyle}>Average Users</th>
              <th style={thTdStyle}>Status</th>
            </tr>
```

```
    </thead>
    <tbody>
      {utilizationData.map(lab => (
        <tr key={lab.lab_id}>
          <td style={thTdStyle}>{lab.name}</td>
          <td style={thTdStyle}>{lab.estimated_users}</td>
          <td style={thTdStyle}>{lab.avg_users}</td>
          <td style={thTdStyle}>{lab.status}</td>
        </tr>
      ))}
    </tbody>
  </table>
</div>

<div style={sectionStyle}>
  <h3 style={{ fontSize: '20px' }}>Most Popular Labs</h3>
  <table style={tableStyle}>
    <thead>
      <tr>
        <th style={thTdStyle}>Lab</th>
        <th style={thTdStyle}>Total Sessions</th>
        <th style={thTdStyle}>Total User Minutes</th>
      </tr>
    </thead>
    <tbody>
      {popularLabs.map(lab => (
        <tr key={lab.lab_id}>
          <td style={thTdStyle}>{lab.name}</td>
          <td style={thTdStyle}>{lab.total_sessions}</td>
          <td style={thTdStyle}>{lab.total_user_minutes}</td>
```

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

export default MonitorPage;
```

infra/frontend/src/pages/PerformancePage.js:

```
import React, { useEffect, useState } from 'react';
import { Line } from 'react-chartjs-2';
import {
 Chart as ChartJS,
 CategoryScale,
 LinearScale,
 PointElement,
 LineElement,
 Title,
 Tooltip,
 Legend
} from 'chart.js';
import axios from 'axios';

ChartJS.register(CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip,
Legend);
```

```
const PerformancePage = () => {
  const [serverStats, setServerStats] = useState([]);
  const [peakTimes, setPeakTimes] = useState({});
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    axios.get(`http://localhost:3004/performance/servers/stats`)
      .then(async (response) => {
        const stats = response.data;
        setServerStats(stats);
        const serverIds = [...new Set(stats.map(stat => stat.server_id))];
        const peakData = {};
        for (const id of serverIds) {
          try {
            const res = await axios.get(`http://localhost:3004/performance/servers/${id}/peaks`);
            peakData[id] = res.data;
          } catch (err) {
            console.error(`Error fetching peaks for server ${id}`, err);
            peakData[id] = [];
          }
        }
        setPeakTimes(peakData);
        setLoading(false);
      })
      .catch(err => {
        console.error('Error fetching server stats:', err);
        setLoading(false);
      });
  }, []);
```

```
const formatStatsForChart = (stats) => {
  const sorted = [...stats].sort((a, b) => new Date(a.recorded_at) - new Date(b.recorded_at));
  const labels = sorted.map(stat => {
    const date = new Date(stat.recorded_at);
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0');
    const hours = String(date.getHours()).padStart(2, '0');
    const minutes = String(date.getMinutes()).padStart(2, '0');
    return `${month}/${day} ${hours}:${minutes}`;
  });

  const cpuData = sorted.map(stat => stat.cpu_usage);
  const memoryData = sorted.map(stat => stat.memory_usage);
  const diskData = sorted.map(stat => stat.disk_usage);

  return {
    labels,
    datasets: [
      {
        label: 'CPU Usage',
        data: cpuData,
        borderColor: '#f87171',
        backgroundColor: 'rgba(248, 113, 113, 0.1)',
        fill: true,
      },
      {
        label: 'Memory Usage',
        data: memoryData,
        borderColor: '#60a5fa',
        backgroundColor: 'rgba(96, 165, 250, 0.1)',
```

```
        fill: true,
      },
      {
        label: 'Disk Usage',
        data: diskData,
        borderColor: '#34d399',
        backgroundColor: 'rgba(52, 211, 153, 0.1)',
        fill: true,
      }
    ]
  };
};

  const groupStatsByServer = (stats) => {
    return stats.reduce((acc, stat) => {
      if (!acc[stat.server_id]) acc[stat.server_id] = [];
      acc[stat.server_id].push(stat);
      return acc;
    }, {});
  };

  const groupedStats = groupStatsByServer(serverStats);

  return (
    <div style={{ padding: '32px', maxWidth: '1200px', margin: '0 auto' }}>
      <h1 style={{ fontSize: '28px', fontWeight: 'bold', marginBottom: '32px', textAlign:
'center' }}>
        All Server Performance
      </h1>

      {loading ? (
```

```
      <p style={{ fontSize: '16px', textAlign: 'center' }}>Loading performance data...</p>
    ) : (
      Object.entries(groupedStats).map(([serverId, stats]) => (
        <div key={serverId} style={{ marginBottom: '64px', padding: '24px', border: '1px solid #ddd', borderRadius: '8px' }}>
          <h2 style={{ fontSize: '22px', fontWeight: '600', marginBottom: '16px' }}>
            Server #{serverId}
          </h2>

          <Line data={formatStatsForChart(stats)} />

          <div style={{ marginTop: '28px' }}>
            <h3 style={{ fontSize: '18px', marginBottom: '12px' }}>Peak Access Times</h3>

            {peakTimes[serverId] && peakTimes[serverId].length > 0 ? (
              <table style={{ width: '100%', borderCollapse: 'collapse' }}>
                <thead>
                  <tr style={{ backgroundColor: '#f9fafb' }}>
                    <th style={{ border: '1px solid #ccc', padding: '10px', textAlign: 'left' }}>Time</th>
                    <th style={{ border: '1px solid #ccc', padding: '10px', textAlign: 'left' }}>Total Users</th>
                  </tr>
                </thead>
                <tbody>
                  {peakTimes[serverId].map((peak, index) => (
                    <tr key={index}>
                      <td style={{ border: '1px solid #eee', padding: '10px' }}>{peak.time}</td>
                      <td style={{ border: '1px solid #eee', padding: '10px' }}>{peak.users}</td>
                    </tr>
                  ))}
```

```
              </tbody>
            </table>
          ) : (
            <p>No peak times available.</p>
          )}
        </div>
      </div>
    ))
  )}
 </div>
);
};

export default PerformancePage;
```

infra/frontend/src/pages/ResourceAllocationPage.js:

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

function ResourceAllocationPage() {
  const [labs, setLabs] = useState([]);
  const [servers, setServers] = useState([]);
  const [allocations, setAllocations] = useState([]);
  const [newServer, setNewServer] = useState({ ip_address: '', max_cpu: '', max_memory: '', max_disk: '' });
  const [selectedLabId, setSelectedLabId] = useState(null);
  const [labScaling, setLabScaling] = useState({
    estimated_users: '',
    per_user_cpu: '',
```

```
  per_user_memory: '',
  per_user_disk: ''
});

useEffect(() => {
  fetchData();
}, []);

const fetchData = async () => {
  const [labsRes, serversRes, allocationsRes] = await Promise.all([
    axios.get('http://localhost:3005/labs'),
    axios.get('http://localhost:3005/servers'),
    axios.get('http://localhost:3005/allocations')
  ]);
  setLabs(labsRes.data);
  setServers(serversRes.data);
  setAllocations(allocationsRes.data);
};

const handleAddServer = async () => {
  try {
    await axios.post('http://localhost:3005/servers', {
      ip_address: newServer.ip_address,
      max_cpu: Number(newServer.max_cpu),
      max_memory: Number(newServer.max_memory),
      max_disk: Number(newServer.max_disk),
    });
    setNewServer({ ip_address: '', max_cpu: '', max_memory: '', max_disk: '' });
    fetchData();
  } catch (err) {
```

```
    console.error('Failed to add server', err);
  }
};

const handleAllocate = async () => {
  fetchData();
};

const handleScaleLab = async () => {
  if (!selectedLabId) return;

  const {
    per_user_cpu,
    per_user_memory,
    per_user_disk,
    estimated_users
  } = labScaling;

  try {
    await axios.patch(`http://localhost:3005/labs/${selectedLabId}`, {
      per_user_cpu: Number(per_user_cpu),
      per_user_memory: Number(per_user_memory),
      per_user_disk: Number(per_user_disk),
      estimated_users: Number(estimated_users),
    });

    await handleAllocate();
    await fetchData();
  } catch (err) {
    console.error('Failed to scale lab and reallocate', err);
```

```
  }
};

const sectionStyle = {
  marginBottom: '2rem',
  padding: '1.5rem',
  border: '1px solid #ddd',
  borderRadius: '6px',
  backgroundColor: '#f9f9f9'
};

const inputStyle = {
  padding: '10px',
  borderRadius: '4px',
  border: '1px solid #ccc',
  width: '100%',
  boxSizing: 'border-box'
};

const buttonStyle = {
  padding: '10px 20px',
  backgroundColor: '#2563eb',
  color: '#fff',
  border: 'none',
  borderRadius: '4px',
  cursor: 'pointer'
};

const tableCell = {
  border: '1px solid #ccc',
```

```
    padding: '8px',
    backgroundColor: "#ffffff"
  };

  return (
    <div style={{ padding: '2rem', maxWidth: '1200px', margin: '0 auto' }}>
      <h1 style={{ fontSize: '28px', fontWeight: 'bold', marginBottom: '2rem', textAlign: 'center' }}>
        Lab Infrastructure Manager
      </h1>

      {/* Add Server */}
      <section style={sectionStyle}>
        <h2 style={{ fontSize: '20px', marginBottom: '1rem' }}>Add New Server</h2>
        <div style={{ display: 'flex', gap: '1rem', flexWrap: 'wrap', marginBottom: '1rem' }}>
          {['ip_address', 'max_cpu', 'max_memory', 'max_disk'].map(key => (
            <input
              key={key}
              type={key === 'ip_address' ? 'text' : 'number'}
              placeholder={key.replace(/_/g, ' ').toUpperCase()}
              value={newServer[key]}
              onChange={e => setNewServer({ ...newServer, [key]: e.target.value })}
              style={{ ...inputStyle, flex: '1 0 200px' }}
            />
          ))}
        </div>
        <button onClick={handleAddServer} style={buttonStyle}>Add Server</button>
      </section>

      {/* Select Lab */}
      <section style={sectionStyle}>
```

```
      <h2 style={{ fontSize: '20px', marginBottom: '1rem' }}>Select a Lab to Scale</h2>
      <select
        onChange={e => setSelectedLabId(e.target.value)}
        value={selectedLabId || ''}
        style={{ ...inputStyle, maxWidth: '300px' }}
      >
        <option value="">-- Select a Lab --</option>
        {labs.map(lab => (
          <option key={lab.id} value={lab.id}>{lab.name}</option>
        ))}
      </select>
    </section>

    {/* Scale Lab */}
    {selectedLabId && (
      <section style={{ ...sectionStyle, backgroundColor: '#fffbea' }}>
        <h2 style={{ fontSize: '20px', marginBottom: '1rem' }}>Scale Lab Resources</h2>
        <div style={{ display: 'flex', gap: '1rem', flexWrap: 'wrap', marginBottom: '1rem' }}>
          {['estimated_users', 'per_user_cpu', 'per_user_memory', 'per_user_disk'].map(key => (
            <div key={key} style={{ flex: '1 0 200px' }}>
              <label style={{ display: 'block', marginBottom: '4px' }}>{key.replace(/_/g, ' ')}</label>
              <input
                type="number"
                value={labScaling[key] || ''}
                onChange={e => setLabScaling({ ...labScaling, [key]: e.target.value })}
                style={inputStyle}
              />
            </div>
          ))}
        </div>
```

```
        <button onClick={handleScaleLab} style={{ ...buttonStyle, backgroundColor:
'#ca8a04' }}>
          Scale & Reallocate
        </button>
      </section>
    )}

    {/* Labs Overview */}
    <section style={sectionStyle}>
      <h2 style={{ fontSize: '20px', marginBottom: '1rem' }}>Current Labs</h2>
      <table style={{ width: '100%', borderCollapse: 'collapse' }}>
        <thead style={{ backgroundColor: '#f7f7f7' }}>
          <tr>
            <th style={tableCell}>Lab Name</th>
            <th style={tableCell}>Estimated Users</th>
            <th style={tableCell}>CPU/user</th>
            <th style={tableCell}>Memory/user</th>
            <th style={tableCell}>Disk/user</th>
          </tr>
        </thead>
        <tbody>
          {labs.map(lab => (
            <tr key={lab.id}>
              <td style={tableCell}>{lab.name}</td>
              <td style={tableCell}>{lab.estimated_users}</td>
              <td style={tableCell}>{lab.estimated_cpu}</td>
              <td style={tableCell}>{lab.estimated_memory}</td>
              <td style={tableCell}>{lab.estimated_disk}</td>
            </tr>
          ))}
        </tbody>
```

```
      </table>
    </section>

    {/* Allocations */}
    <section style={sectionStyle}>
      <h2 style={{ fontSize: '20px', marginBottom: '1rem' }}>Lab-Server Allocations</h2>
      <ul style={{ paddingLeft: '1rem' }}>
        {allocations.map(alloc => (
          <li key={alloc.id} style={{ marginBottom: '4px' }}>
            Lab <strong>{alloc.lab_name}</strong> → Server <strong>{alloc.server_id}</strong>
          </li>
        ))}
      </ul>
    </section>

    {/* Server Load */}
    <section style={sectionStyle}>
      <h2 style={{ fontSize: '20px', marginBottom: '1rem' }}>Current Server Load</h2>
      <table style={{ width: '100%', borderCollapse: 'collapse' }}>
        <thead style={{ backgroundColor: '#f0f0f0' }}>
          <tr>
            <th style={tableCell}>Server ID</th>
            <th style={tableCell}>IP Address</th>
            <th style={tableCell}>CPU Usage</th>
            <th style={tableCell}>Memory Usage</th>
            <th style={tableCell}>Disk Usage</th>
          </tr>
        </thead>
        <tbody>
          {servers.map(server => (
```

```
            <tr key={server.id}>
              <td style={tableCell}>{server.id}</td>
              <td style={tableCell}>{server.ip_address}</td>
              <td style={tableCell}>{server.cpu_usage} / {server.max_cpu}</td>
              <td style={tableCell}>{server.memory_usage} / {server.max_memory}</td>
              <td style={tableCell}>{server.disk_usage} / {server.max_disk}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </section>
  </div>
 );
}

export default ResourceAllocationPage;
```

infra/frontend/src/components/Navbar.js:

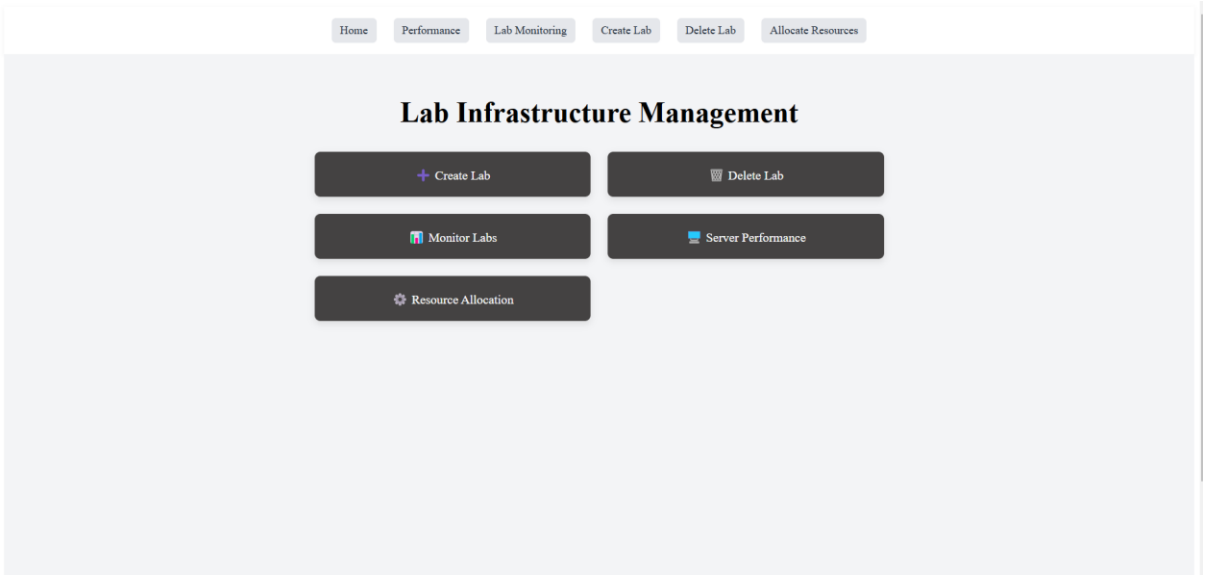
```
import React from 'react';
import { Link } from 'react-router-dom';

const Navbar = () => {
  const navStyle = {
    padding: '1rem 2rem',
    backgroundColor: '#ffffff',
    boxShadow: '0 2px 6px rgba(0, 0, 0, 0.05)',
    display: 'flex',
    justifyContent: 'center',
    gap: '1.5rem',
```

```
  fontSize: '1rem',
  fontWeight: '500'
};

const linkStyle = {
  textDecoration: 'none',
  color: '#1f2937', // neutral dark gray
  padding: '0.5rem 0.75rem',
  borderRadius: '6px',
  transition: 'background 0.2s'
};

const hoverStyle = {
  backgroundColor: '#e5e7eb' // soft hover bg
};

return (
  <nav style={navStyle}>
    {[
      { to: '/', label: 'Home' },
      { to: '/server-performance', label: 'Performance' },
      { to: '/lab-monitoring', label: 'Lab Monitoring' },
      { to: '/lab-creation', label: 'Create Lab' },
      { to: '/lab-deletion', label: 'Delete Lab' },
      { to: '/resource-allocation', label: 'Allocate Resources' }
    ].map(({ to, label }) => (
      <Link
        key={to}
        to={to}
        style={linkStyle}
```

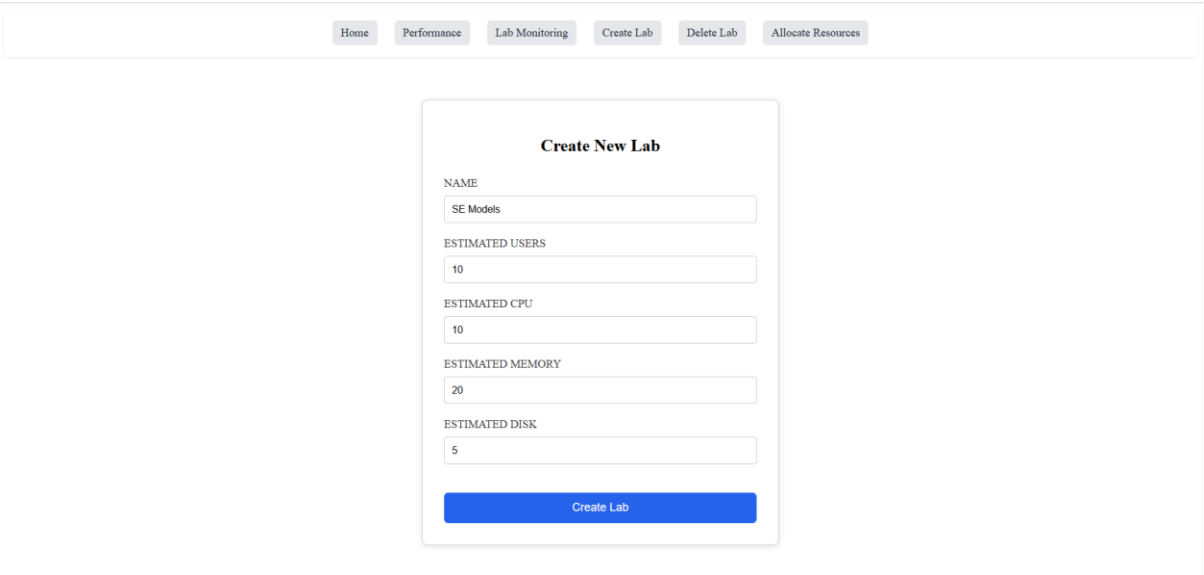
```
          onMouseEnter={e => Object.assign(e.target.style, hoverStyle)}
          onMouseLeave={e => Object.assign(e.target.style, linkStyle)}
        >
          {label}
        </Link>
      ))}
    </nav>
  );
};

export default Navbar;
```

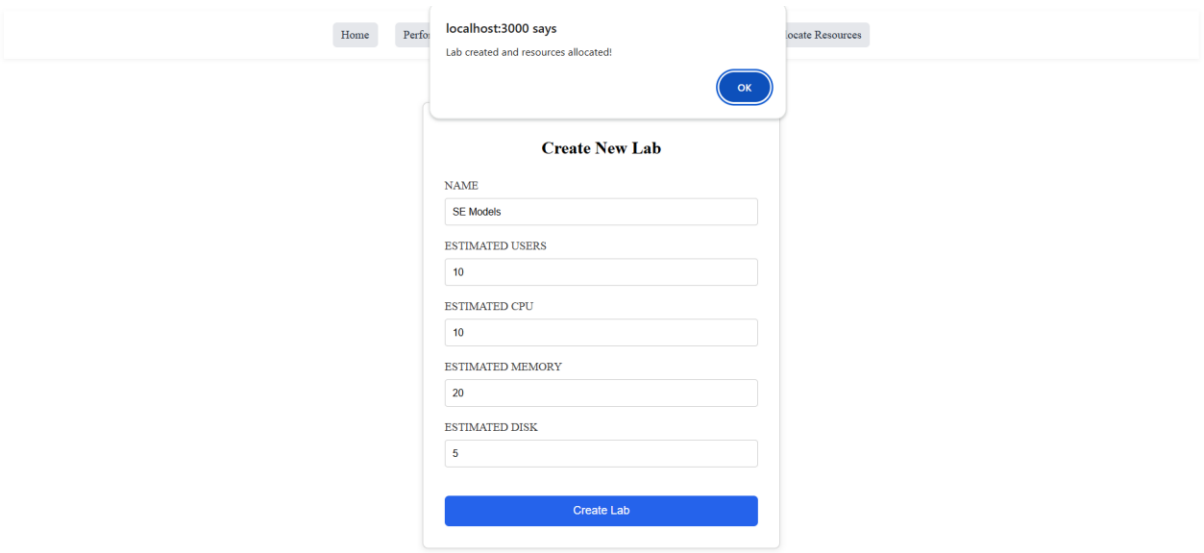
Screenshots:

Home:

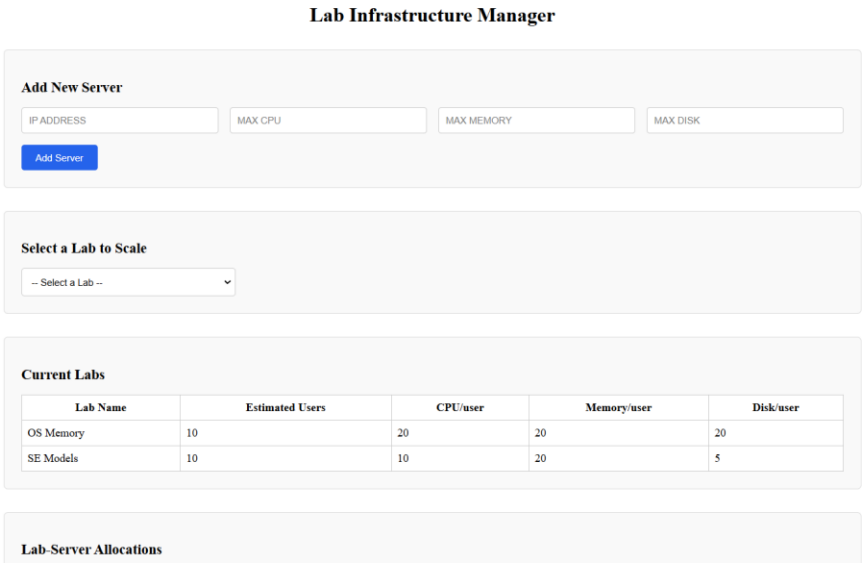


Create Lab Service:





Resource Allocation Service:



Current Labs

Lab Name	Estimated Users	CPU/user	Memory/user	Disk/user
OS Memory	10	20	20	20
SE Models	10	10	20	5

Lab-Server Allocations

- Lab **OS Memory** → Server **1**
- Lab **OS Memory** → Server **2**
- Lab **SE Models** → Server **3**

Current Server Load

Server ID	IP Address	CPU Usage	Memory Usage	Disk Usage
1	192.168.0.101	100 / 100	100 / 100	100 / 100
2	192.168.0.102	100 / 100	100 / 100	100 / 100
3	192.168.0.103	100 / 300	200 / 500	50 / 500

Scaling-down Resources for OS Memory Lab:

Select a Lab to Scale

OS Memory

Scale Lab Resources

estimated users	per user cpu	per user memory	per user disk
5	20	20	20

Scale & Reallocate

Current Labs

Lab Name	Estimated Users	CPU/user	Memory/user	Disk/user
OS Memory	5	20	20	20
SE Models	10	10	20	5

Lab-Server Allocations

- Lab **SE Models** → Server **3**
- Lab **OS Memory** → Server **1**

Current Labs

Lab Name	Estimated Users	CPU/user	Memory/user	Disk/user
OS Memory	5	20	20	20
SE Models	10	10	20	5

Lab-Server Allocations

- Lab **SE Models** → Server **3**
- Lab **OS Memory** → Server **1**

Current Server Load

Server ID	IP Address	CPU Usage	Memory Usage	Disk Usage
1	192.168.0.101	100 / 100	100 / 100	100 / 100
2	192.168.0.102	0 / 100	0 / 100	0 / 100
3	192.168.0.103	100 / 300	200 / 500	50 / 500

Scaling-up Resources for SE Models Lab:

Select a Lab to Scale

SE Models

Scale Lab Resources

estimated users	per user cpu	per user memory	per user disk
20	10	20	5

Scale & Reallocate

Current Labs

Lab Name	Estimated Users	CPU/user	Memory/user	Disk/user
OS Memory	5	20	20	20
SE Models	20	10	20	5

Lab-Server Allocations

- Lab **OS Memory** → Server **1**
- Lab **SE Models** → Server **2**
- Lab **SE Models** → Server **3**

Current Labs

Lab Name	Estimated Users	CPU/user	Memory/user	Disk/user
OS Memory	5	20	20	20
SE Models	20	10	20	5

Lab-Server Allocations

- Lab **OS Memory** → Server **1**
- Lab **SE Models** → Server **2**
- Lab **SE Models** → Server **3**

Current Server Load

Server ID	IP Address	CPU Usage	Memory Usage	Disk Usage
1	192.168.0.101	100 / 100	100 / 100	100 / 100
2	192.168.0.102	100 / 100	100 / 100	100 / 100
3	192.168.0.103	100 / 300	300 / 500	0 / 500

Adding server:

Home | Performance | Lab Monitoring | Create Lab | Delete Lab | Allocate Resources

Lab Infrastructure Manager

Add New Server

192.168.0.104 | 200 | 300 | 100

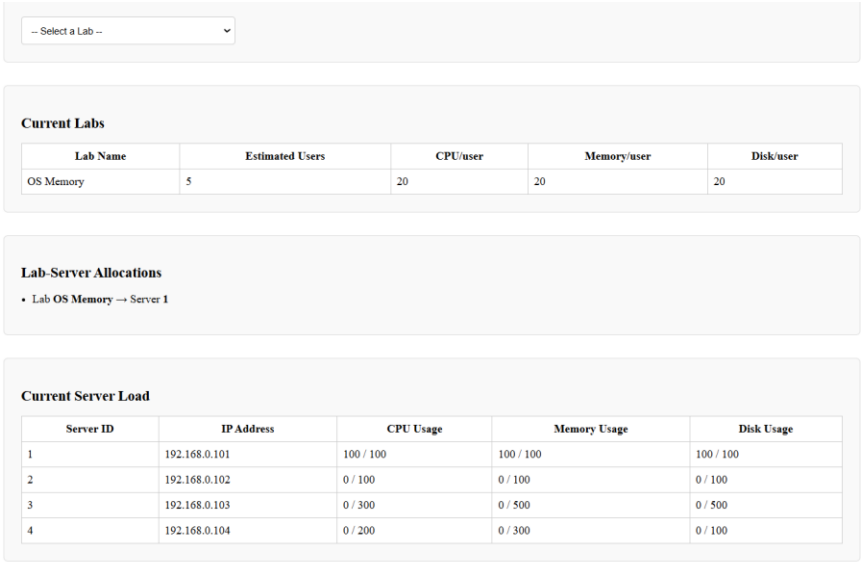
Add Server

Select a Lab to Scale

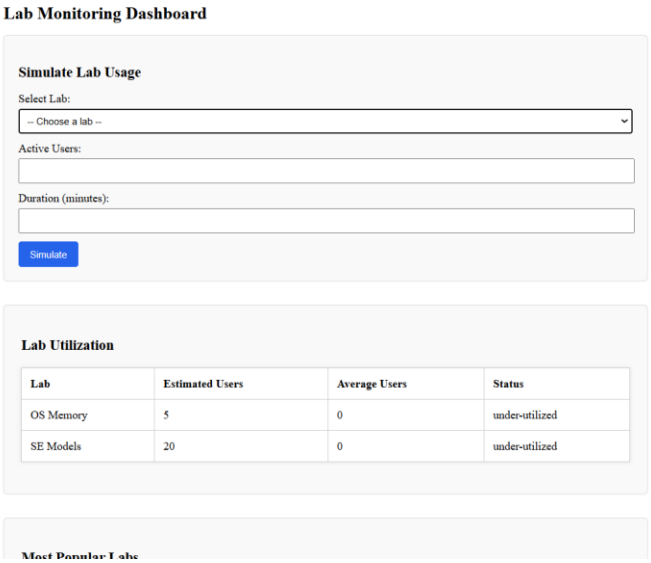
-- Select a Lab --

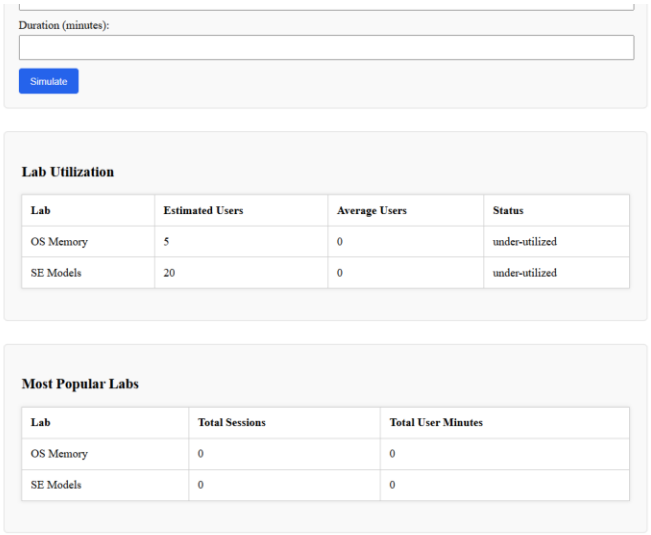
Current Labs

Lab Name	Estimated Users	CPU/user	Memory/user	Disk/user
OS Memory	5	20	20	20

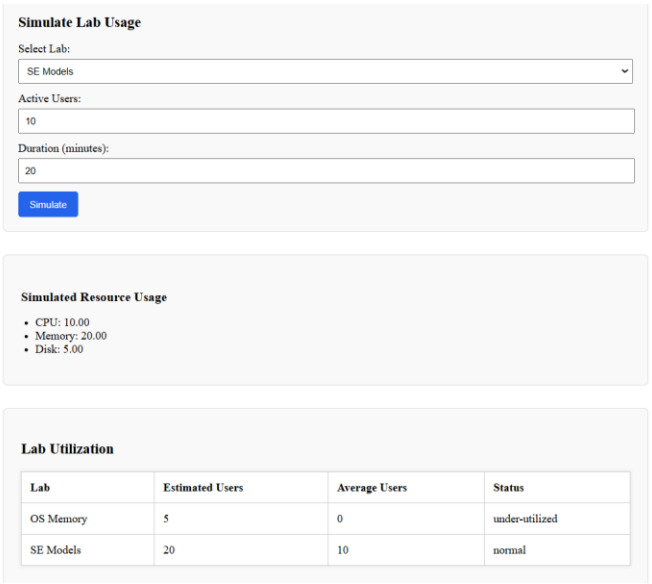


Lab Monitoring Service:

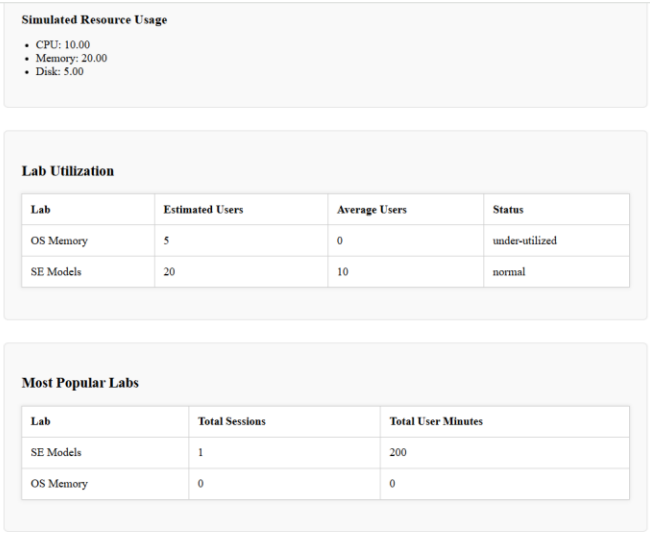




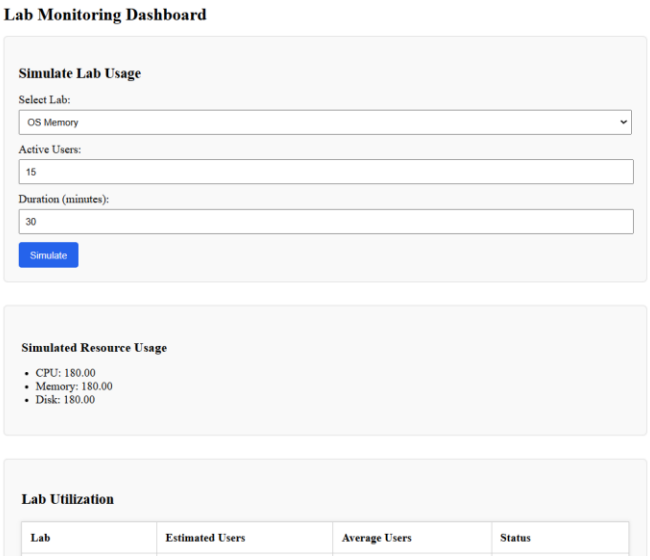
Simulating normal load for SE Models Lab:



Popular Lab is set as SE Models Lab:



Simulating over-utilization for OS Memory Lab:



Popular Lab is now set as OS Memory Lab due to more engagement:

Simulated Resource Usage

- CPU: 180.00
- Memory: 180.00
- Disk: 180.00

Lab Utilization

Lab	Estimated Users	Average Users	Status
OS Memory	5	15	over-utilized
SE Models	20	10	normal

Most Popular Labs

Lab	Total Sessions	Total User Minutes
OS Memory	1	450
SE Models	1	200

Performance of servers:

Home | Performance | Lab Monitoring | Create Lab | Delete Lab | Allocate Resources

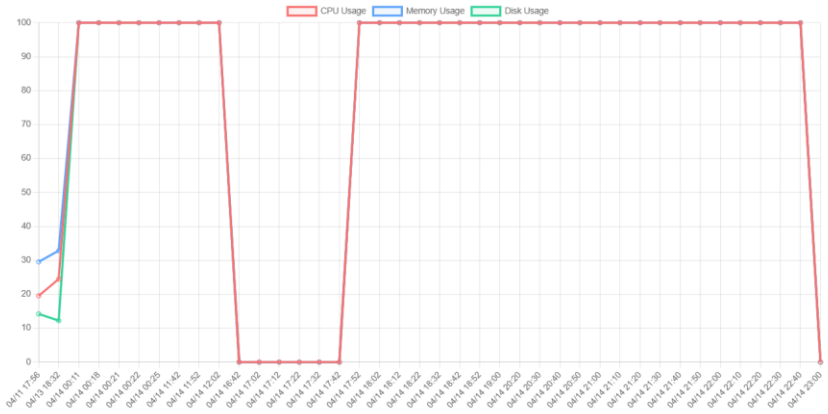
All Server Performance

Server #1

Peak Access Times

Time	Total Users
2025-04-14 23:04:51	15
2025-04-14 23:07:19	10

Server #2



Peak Access Times

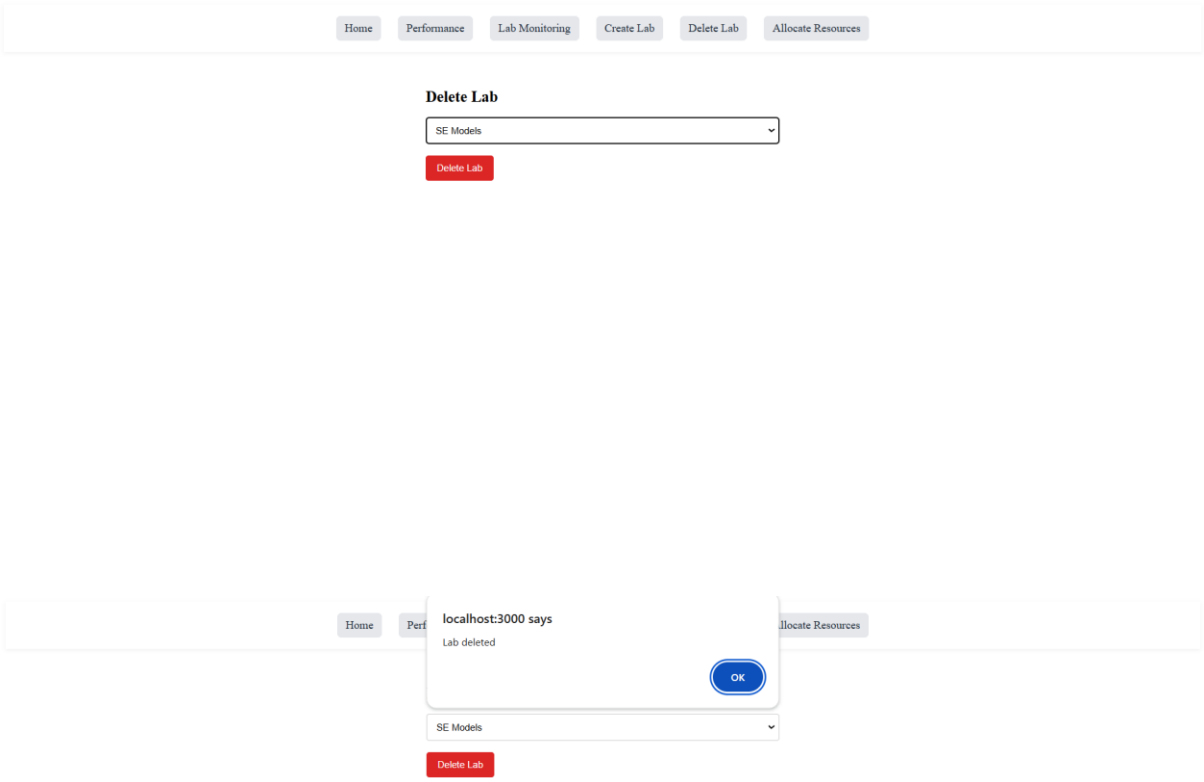
Time	Total Users
2025-04-14 23:03:29	10

Server #3

Peak Access Times

Time	Total Users
2025-04-14 23:03:29	10

Delete Lab Service:



Readme:

Virtual Lab Infrastructure Management

Description:
An application that follows microservice architecture to manage virtual lab infrastructure. This includes managing creation and deletion of labs, resource allocation to labs, checking performance of servers present and monitoring usage of labs through simulation.

Tools and Frameworks Used:

Frontend:

- **React.js**
  The frontend library used in our project was React.js. It can build reusable and dynamic User Interface components.
- **React Router DOM**
  React Router DOM was used for allowing client-side routing between different pages in the React app.
- **Axios**
  Axios was used to send requests from frontend to backend. It is an HTTP client.
- **Recharts**
  We used Recharts for showing the performance of the servers in the form of graphs.
- **Inline Styling**
  Inline styles are used for layout and styling of components instead of CSS or external frameworks.

Backend:

- **Express.js (Node.js)**
  REST API server to handle simulation logic, monitoring endpoints, and for CRUD operations.
- **MySQL**
  The database that we used to store various entities like labs, servers, server_stats etc.

Infrastructure:

- **NGINX**
  Used as a reverse proxy to route requests to the appropriate frontend and backend services. Manages port numbers for the frontend and multiple microservices.

Microservices:

- Creation of labs
- Deletion of labs
- Allocating resources to labs
- Monitoring over/under utilization of labs’ resources
- Server performance.