EXPERIMENT 12

**Title**

Concurrent Ticket Booking System with Seat Locking and Confirmation

**Objective**

Learn how to implement a ticket booking system in Node.js that handles concurrent seat reservation requests safely using a seat locking mechanism. This task helps you understand how to manage in-memory state, handle concurrent access, and design a system that prevents race conditions during booking.

**Task Description**

Create a Node.js and Express.js application that simulates a ticket booking system for events or movie theaters. Implement endpoints to view available seats, temporarily lock a seat for a user, and confirm the booking. Design a seat locking mechanism so that when a seat is locked, it cannot be locked or booked by other users until it is either confirmed or the lock expires automatically (for example, after 1 minute). Store seat states in an in-memory data structure for simplicity. Include clear success and error messages for different scenarios, such as trying to lock an already locked or booked seat, or confirming a seat without a lock. Test your API by simulating concurrent requests to demonstrate that the locking logic correctly prevents double booking and ensures reliable seat allocation.

CODE

import express from "express";

import { v4 as uuidv4 } from "uuid";

const app = express();

app.use(express.json());

const seats = new Map();

for (let i = 1; i <= 10; i++) {

  seats.set(String(i), { state: "available" });

}

const LOCK\_DURATION\_MS = 60 \* 1000;

function clearLock(seat) {

  if (seat.lockTimeoutId) {

    clearTimeout(seat.lockTimeoutId);

    seat.lockTimeoutId = undefined;

  }

  delete seat.lockId;

  delete seat.lockedAt;

  seat.state = "available";

}

app.get("/seats", (req, res) => {

  const result = {};

  for (const [id, seat] of seats.entries()) {

    result[id] = {

      state: seat.state,

    };

  }

  res.json(result);

});

app.post("/lock/:id", (req, res) => {

  const id = String(req.params.id);

  const seat = seats.get(id);

  if (!seat) {

    return res.status(404).json({ message: `Seat ${id} does not exist.` });

  }

  if (seat.state === "booked") {

    return res.status(400).json({ message: `Seat ${id} is already booked.` });

  }

  if (seat.state === "locked") {

    return res.status(400).json({ message: `Seat ${id} is already locked.` });

  }

  seat.state = "locked";

  seat.lockId = uuidv4();

  seat.lockedAt = Date.now();

  seat.lockTimeoutId = setTimeout(() => {

    // Only clear if still locked (it may have been booked)

    if (seat.state === "locked") {

      clearLock(seat);

      console.log(`Auto-unlocked seat ${id} after timeout.`);

    }

  }, LOCK\_DURATION\_MS);

  return res.status(200).json({

    message: `Seat ${id} locked successfully. Confirm within 1 minute.`,

  });

});

app.post("/confirm/:id", (req, res) => {

  const id = String(req.params.id);

  const seat = seats.get(id);

  if (!seat) {

    return res.status(404).json({ message: `Seat ${id} does not exist.` });

  }

  if (seat.state !== "locked") {

    return res

      .status(400)

      .json({ message: "Seat is not locked and cannot be booked" });

  }

  if (seat.lockTimeoutId) {

    clearTimeout(seat.lockTimeoutId);

    seat.lockTimeoutId = undefined;

  }

  seat.state = "booked";

  delete seat.lockId;

  delete seat.lockedAt;

  return res.status(200).json({ message: `Seat ${id} booked successfully!` });

});

const PORT = 3000;

app.listen(PORT, () => {

  console.log(`Seat-locking server listening on http://localhost:${PORT}`);

});

OUTPUT

