Node.js

Student Name: Akshit Sharma **UID:23BCS10929**
Branch: BE CSE **Section/Group: 622-B**
Semester: 5
Subject Name: Full Stack - I **Subject Code: 23CSP-339**

Practice 3 - Node.js

Title
Concurrent Ticket Booking System with Seat Locking and Confirmation

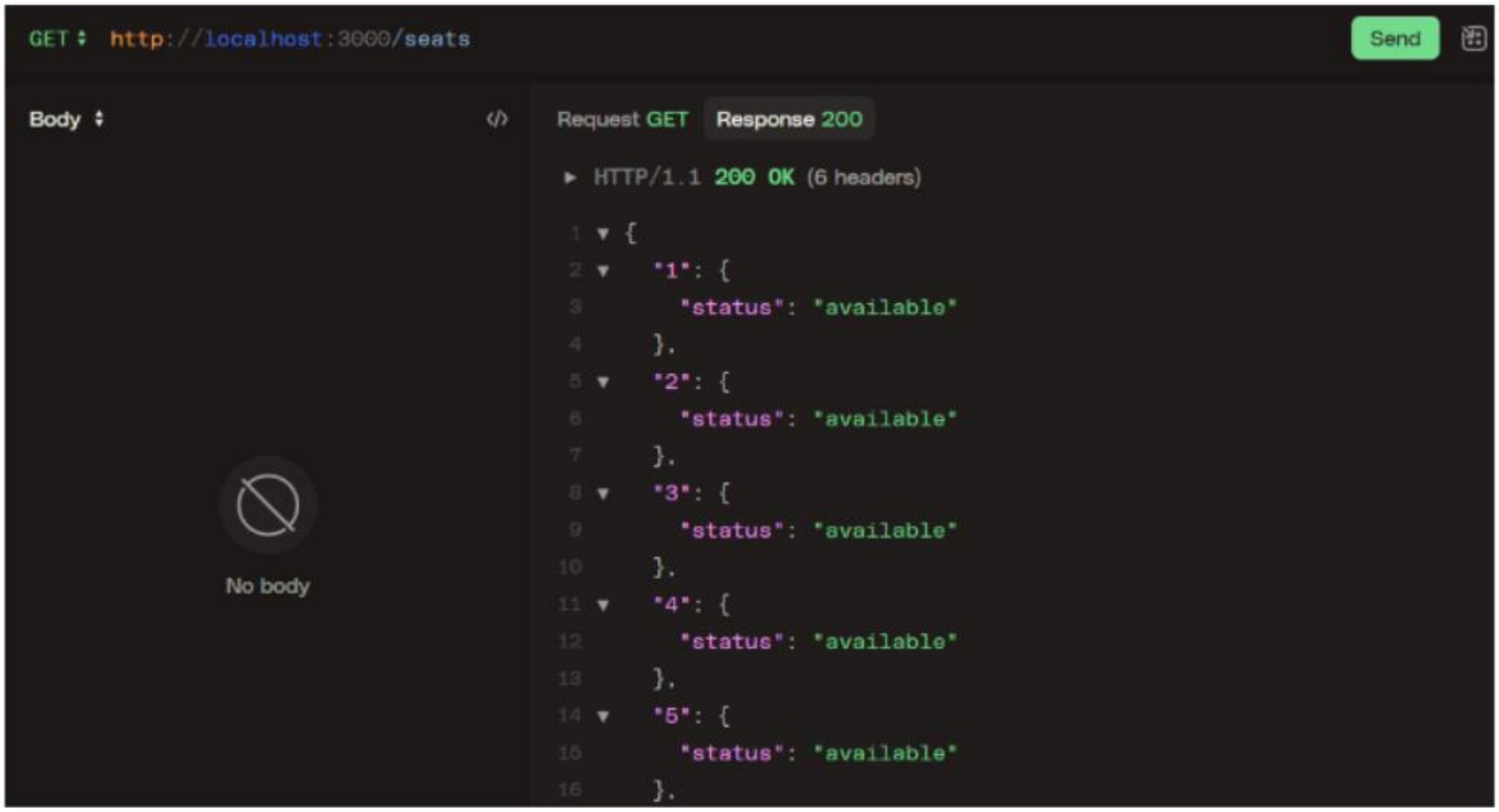
Objective
Learn how to implement a ticket booking system in Node.js that handles concurrent seat reservation requests safely using a seat locking mechanism. This task helps you understand how to manage in-memory state, handle concurrent access, and design a system that prevents race conditions during 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:
```
const express = require("express");
const app = express();
const PORT = 3000;

app.use(express.json());

// In-memory seat storage
// Each seat has: id, status ("available", "locked", "booked"), lockedBy, lockExpiry
let seats = [];
const TOTAL_SEATS = 10; // For simplicity, 10 seats
for (let i = 1; i <= TOTAL_SEATS; i++) {
  seats.push({ id: i, status: "available", lockedBy: null, lockExpiry: null });
}

// Helper: Clear expired locks
function clearExpiredLocks() {
  const now = Date.now();
  seats.forEach(seat => {
    if (seat.status === "locked" && seat.lockExpiry <= now) {
      seat.status = "available";
      seat.lockedBy = null;
      seat.lockExpiry = null;
    }
  });
}

// GET: View all seats
app.get("/seats", (req, res) => {
  clearExpiredLocks();
  res.json(seats);
});

// POST: Lock a seat
app.post("  ", (req, res) => {
  clearExpiredLocks();
  const seatId = parseInt(req.params.id);
  const userId = req.body.userId; // user trying to lock

  const seat = seats.find(s => s.id === seatId);
  if (!seat) return res.status(404).json({ error: "Seat not found" });

  if (seat.status === "available") {
    seat.status = "locked";
    seat.lockedBy = userId;
    seat.lockExpiry = Date.now() + 60 * 1000; // lock expires in 1 minute
    return res.json({ message: `Seat ${seatId} locked by user ${userId}`, seat });
  } else if (seat.status === "locked") {
    return res.status(400).json({ error: `Seat ${seatId} is already locked by another user` });
  } else if (seat.status === "booked") {
    return res.status(400).json({ error: `Seat ${seatId} is already booked` });
  }
});

// POST: Confirm booking
app.post("/seats/:id/confirm", (req, res) => {
  clearExpiredLocks();
  const seatId = parseInt(req.params.id);
  const userId = req.body.userId;

  const seat = seats.find(s => s.id === seatId);
  if (!seat) return res.status(404).json({ error: "Seat not found" });

  if (seat.status === "locked" && seat.lockedBy === userId) {
    seat.status = "booked";
    seat.lockedBy = null;
    seat.lockExpiry = null;
    return res.json({ message: `Seat ${seatId} successfully booked by user ${userId}`, seat });
  } else if (seat.status === "locked" && seat.lockedBy !== userId) {
    return res.status(400).json({ error: `Seat ${seatId} is locked by another user` });
  } else if (seat.status === "available") {
    return res.status(400).json({ error: `Seat ${seatId} is not locked. Lock it first` });
  } else if (seat.status === "booked") {
    return res.status(400).json({ error: `Seat ${seatId} is already booked` });
  }
});

// Start server
```

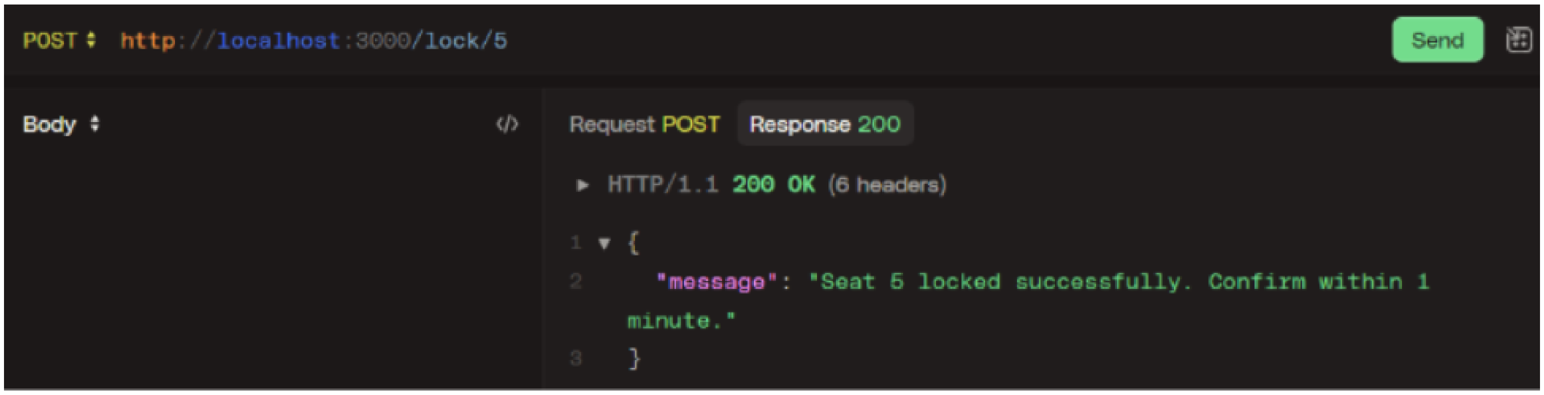
```
app.listen(PORT, () => {
  console.log(`Ticket Booking API running at http://localhost:${PORT}`);
});
```

Expected Output :



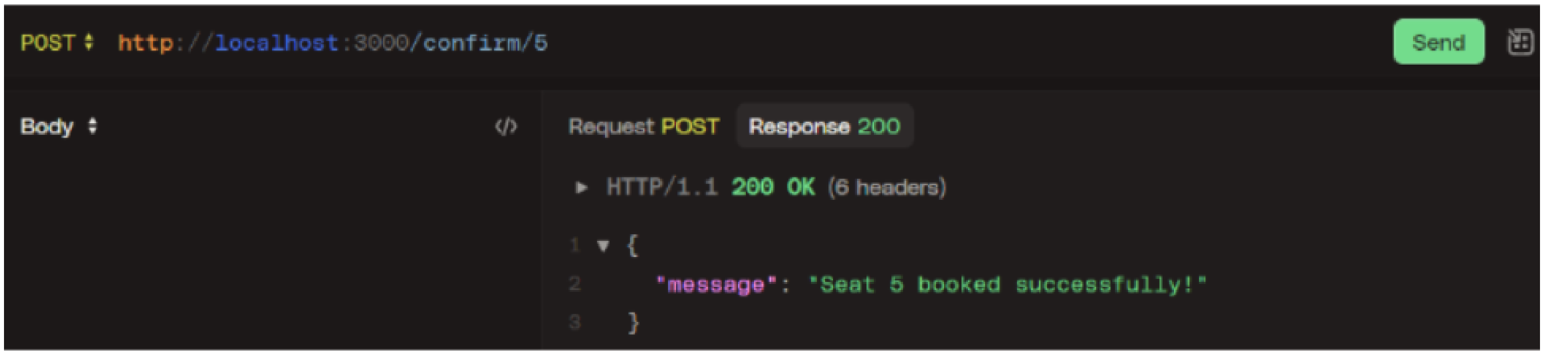
```
GET http://localhost:3000/seats
```

```
▸ HTTP/1.1 200 OK (6 headers)
{
  "1": {
    "status": "available"
  },
  "2": {
    "status": "available"
  },
  "3": {
    "status": "available"
  },
  "4": {
    "status": "available"
  },
  "5": {
    "status": "available"
  },
```



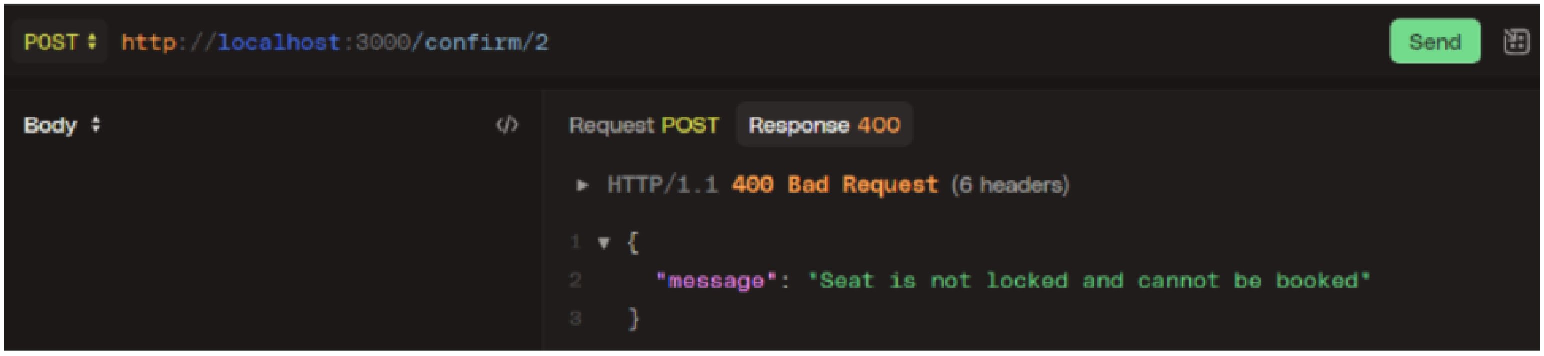
```
POST http://localhost:3000/lock/5
```

```
HTTP/1.1 200 OK (6 headers)
{
  "message": "Seat 5 locked successfully. Confirm within 1 minute."
}
```



```
POST http://localhost:3000/confirm/5
```

```
HTTP/1.1 200 OK (6 headers)
{
  "message": "Seat 5 booked successfully!"
}
```



```
POST http://localhost:3000/confirm/2
```

```
HTTP/1.1 400 Bad Request (6 headers)
{
  "message": "Seat is not locked and cannot be booked"
}
```