

Experiment 4.3 (12)

Pydi Sneha Dishita Chowdary [23BIS70112]

Title

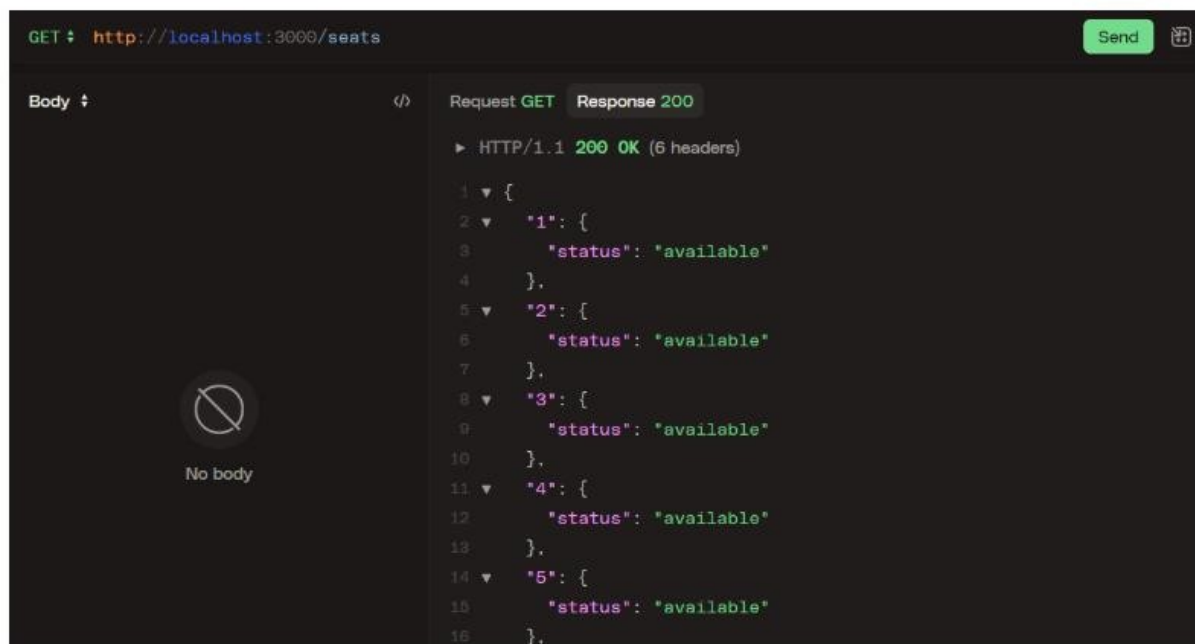
Concurrent Ticket Booking System with Seat Locking and Confirmation

Procedure:

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.

Expected Output



The screenshot shows a web browser interface with a dark theme. At the top, the address bar displays 'GET http://localhost:3000/seats' with a green 'Send' button to its right. Below the address bar, the 'Body' section is expanded, showing a 'No body' message with a circular icon containing a diagonal line. To the right of the 'Body' section, the 'Request' and 'Response' tabs are visible. The 'Response' tab is selected, showing a '200 OK' status with 6 headers. The response body is a JSON array of 5 objects, each representing a seat with an index (1-5) and a status of 'available'.

```
1 {
2   "1": {
3     "status": "available"
4   },
5   "2": {
6     "status": "available"
7   },
8   "3": {
9     "status": "available"
10  },
11  "4": {
12    "status": "available"
13  },
14  "5": {
15    "status": "available"
16  },
17 }
```

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

Body

Request POST Response 200

▶ HTTP/1.1 200 OK (6 headers)

```
1 {
2   "message": "Seat 5 locked successfully. Confirm within 1
3   minute."
}
```

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

Body

Request POST Response 200

▶ HTTP/1.1 200 OK (6 headers)

```
1 {
2   "message": "Seat 5 booked successfully!"
3 }
```

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

Body

Request POST Response 400

▶ HTTP/1.1 400 Bad Request (6 headers)

```
1 {
2   "message": "Seat is not locked and cannot be booked"
3 }
```

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

Body

Request POST Response 200

▶ HTTP/1.1 200 OK (6 headers)

```
1 {
2   "message": "Seat 5 locked successfully. Confirm within 1
3   minute."
}
```

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

Body

Request POST Response 200

▶ HTTP/1.1 200 OK (6 headers)

```
1 {
2   "message": "Seat 5 booked successfully!"
3 }
```

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

Body

Request POST Response 400

▶ HTTP/1.1 400 Bad Request (6 headers)

```
1 {
2   "message": "Seat is not locked and cannot be booked"
3 }
```

Solution using Node.js and Express.js

Here we require a built-in module of ES6 called **Map()** to manage the key:value pairs. It also allows to create custom properties on demand.

Here we also require built-in operator of JavaScript called delete to **delete** the custom properties

Here also required a module **uuid** to generate Unique Key Code for the values

Here we are creating a custom property called **state** having states of the seat booking like **available, booked, locked** etc.

Time is taken in millisecond for 1 minute (60*1000) millisecond

```
//experiment4.3.js 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.lockTimeoutI
d);   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}`);
});

```

GET	http://localhost:3000/seats	Send	▼
POST	http://localhost:3000/lock/5	Send	▼
POST	http://localhost:3000/confirm/5	Send	▼
POST	http://localhost:3000/confirm/2	Send	▼

Test it using Postman

