NAME-Snigdha Ghosh

UID-23ICS10004

Sub-Full Stack

<u>Practice1</u>

CODE-

```
const readline=require("readline")
const rl=readline.createInterface({
    input:process.stdin,
    output:process.stdout
});

const employees=[
    {
        id:"1",
        name:"Alice",
        Age:34
    },
    {
        id:"2",
        name:"inBorder",
        Age:30
    },
    {
        id:"3",
        name:"land",
        Age:34
    }
];

function showMenu(){

    console.log("\n=== Employee Manager ===");
    console.log("1. Add Employee");
    console.log("2. List Employees");
```

```
        console.log("3. Remove Employee");
        console.log("4. Exit\n");

        rl.question("Enter your choise:",(choice)=>{
          switch (choice.trim()){
            case "1":
              addEmployee();
              break;
            case "2":
              listEmployees();
              break;
            case "3":
              removeEmployee();
              break;
            case "4":
              console.log("Exiting...");
              rl.close();
              break;
            default:
              console.log("Invalid choice! Try again.");
              showMenu();
          }
        });
      }
function addEmployee() {
  rl.question("Enter Employee ID: ", (id) => {
    rl.question("Enter Employee Name: ", (name) => {
      if (employees.find(emp => emp.id === id.trim())) {
        console.log("Employee with this ID already exists!");
      } else {
        employees.push({ id: id.trim(), name: name.trim() });
        console.log("Employee added successfully!");
      }
      showMenu();
    });
  });
}

function listEmployees() {
  console.log("\n--- Employee List ---");
  if (employees.length === 0) {
    console.log("No employees found.");
  } else {
    employees.forEach(emp => {
      console.log(`ID: ${emp.id}, Name: ${emp.name}`);
    });
  }
  showMenu();
}

function removeEmployee() {
  rl.question("Enter Employee ID to remove: ", (id) => {
    const index = employees.findIndex(emp => emp.id === id.trim());
    if (index === -1) {
      console.log("Employee not found!");
    } else {
      employees.splice(index, 1);
      console.log("Employee removed successfully!");
    }
    showMenu();
  });
}

// Start the CLI
showMenu();
```

Practice2

CODE-

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware to parse JSON
app.use(express.json());

let cards = [
  { id: 1, suit: "Hearts", value: "Ace" },
  { id: 2, suit: "Hearts", value: "2" },
  { id: 3, suit: "Hearts", value: "King" }
];
let nextId = 4  ;

// GET: List all cards
app.get('/cards', (req, res) => {
  res.json(cards);
});

// GET: Retrieve a card by ID
app.get('/cards/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const card = cards.find(c => c.id === id);
  if (!card) {
    return res.status(404).json({ error: "Card not found" });
  }
  res.json(card);
});

// POST: Add a new card
app.post('/cards', (req, res) => {
  const { suit, value } = req.body;
  if (!suit || !value) {
    return res.status(400).json({ error: "Suit and value are required" });
  }
  const newCard = { id: nextId++, suit, value };
  cards.push(newCard);
  res.status(201).json(newCard);
});

// DELETE: Remove a card by ID
app.delete('/cards/:id', (req, res) => {
  const id = parseInt(req.params.id);
  const index = cards.findIndex(c => c.id === id);
  if (index === -1) {
    return res.status(404).json({ error: "Card not found" });
  }
  const deletedCard = cards.splice(index, 1);
  res.json({ message: "Card deleted", card: deletedCard[0] });
});

// Start server
app.listen(port, () => {
  console.log(`Playing Card API is running at http://localhost:${port}`);
});
```

<u>**Practice3**</u>

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}`);
});
```