Modex Assessment

This assessment will be taken by **approximately 2,500 candidates**.
We understand that many will rely on AI tools during the process. However, this is your opportunity to **stand out**.

To truly differentiate yourself, focus on:

- **Original thinking and innovation:**
  Your ideas, structure, and approach should clearly reflect your own creativity—not AI's generic output.

- **Strong architecture and clean coding standards**
  Well-designed systems, scalable patterns, and readable, maintainable code will set you apart from others.

- **Clear reasoning and problem-solving depth**
  Show how you think. Document your assumptions, logic, and decisions.

AI can assist, but your **innovation, architecture quality, and coding discipline** are what will truly distinguish your work.

You may choose to create your own problem statement and solution, with higher weightage given to healthcare-focused projects. However, below are some suggested topics you can work on.

Part 1: Ticket Booking System(Backend)

Duration: 24 hours from the time of assignment
Tech Stack: Node.js, Express.js , Postgres

Objective:

Design and implement a basic ticket booking system that simulates the core functionality of platforms like **RedBus** or **BookMyShow or Doctor Appointment booking**. The system must handle high concurrency scenarios to prevent race conditions and overbooking.

Assignment Scope

1. Functional Requirements

- **Show/Trip/Slot Management**

    - Admin should be able to create shows or bus trips.
    - Each entry must include:
        - Show or Bus Name or doctor name
        - Start Time
        - Total Number of Seats (e.g., 40)

- **User Operations**
    - Users can retrieve a list of available shows/trips.
    - Users can attempt to book one or more seats for a selected show/trip.
    - The system must ensure that no overbooking can occur due to concurrent booking requests.
    - Booking status should include: PENDING, CONFIRMED, or FAILED.
- **Concurrency Handling**
    - Your solution must handle multiple booking requests for the same seat(s) at the same time.
    - Implement a strategy to ensure data consistency and atomicity of seat booking operations.
- **(Optional Bonus)** Booking Expiry
    - Automatically mark a booking as FAILED if it remains in PENDING status for more than 2 minutes.

2. System Design Document (Written Component)

In addition to the working code, submit a brief technical document that outlines how you would scale this system to support a production-grade application similar to RedBus or BookMyShow.

Include the following in your write-up:

- High-level system architecture and key components
- Database design and how you would scale it (e.g., sharding, replication)

- Concurrency control mechanisms (e.g., locks, transactions, queues)
- Caching strategy (if any)
- Optional: Message queue usage for decoupling critical operations

You may include a basic architecture diagram if needed.

3. Deliverables

- Source code hosted in a public GitHub repository
- A README.md file with:
    - Setup instructions
    - API documentation (Swagger or Postman Collection preferred)
- A short technical write-up (PDF or Markdown format) as described above
- Must be hosted and accessible

Evaluation Criteria

- Functionality and correctness of API implementation
- Handling of concurrency and prevention of overbooking
- Code structure, organization, and clarity
- Quality of technical design and scalability considerations
- Bonus points for use of transactions, locking, and well-documented APIs

Part 2: Ticket Booking System (Frontend)

Duration: 24 hours from the time of backend completion
Tech Stack: React.js, TypeScript
Required Concepts:

- React fundamentals
- State management with Context API
- Error handling (both UI and API level)
- React lifecycle hooks (via functional components and hooks)
- Routing with react-router-dom
- Efficient API calls (avoid unnecessary re-fetching)
- DOM manipulation where necessary
- Clean component structure

Objective

Build a **frontend application** to interact with the backend Ticket Booking System. The app should provide **Admin** and **User** views, allowing users to browse shows/trips, book seats, and see booking status in real time.

Functional Requirements

1. Admin Features

- Create a new show/trip with:
    - Show/Bus/Doctor Name
    - Start Time
    - Total Seats(not applicable for doctor appointment)
- View a list of all shows/trips/slots.
- Simple form validation and error handling (invalid inputs, missing fields, etc.).

2. User Features

- View available shows/trips/slots.
- Select a show/trip/slot and see available seats visually (grid/seat layout).
- Book one or more seats(not applicable for doctor appointment).
- Show booking status (PENDING, CONFIRMED, FAILED).
- Handle errors gracefully (e.g., seats already booked, API errors).

3. Routing

- /admin → Admin dashboard (create and list shows)
- / → List of shows/trips/slot for users
- /booking/:id → Booking page for a specific show/trip

4. State Management

- Use **React Context API** for:
    - Authentication state (basic mock auth, no real login required)
    - Global show/trip/slot data and booking state

5. API Integration

- Consume the backend APIs you've built (Node.js/Express/Postgres) for:
    - Show/slot creation
    - Fetching shows
    - Booking seats
- Use **efficient API calls**:
    - Avoid unnecessary re-fetching when navigating between pages
    - Use caching/memoization where appropriate

6. Error Handling

- Show user-friendly error messages for:
    - API failures
    - Invalid form submissions
    - Booking conflicts due to concurrency
- Display loading and empty states for lists

7. DOM Interaction

- Implement seat selection in the booking page with direct DOM updates for highlighting selected seats before confirming booking.
- Ensure proper cleanup when navigating away.

Bonus (Optional)

- Implement live updates for seat availability (e.g., via polling or WebSocket mock).
- Add seat selection animations.
- Create a responsive design for mobile and desktop views.

Deliverables

- Source code hosted in a public GitHub repository
- README.md containing:
    - Setup instructions
    - Any assumptions made
    - Known limitations
- Postman collection or API documentation link
- Screenshots or short GIF of the application in use

Evaluation Criteria

- Correctness of implementation
- Use of Context API and TypeScript types/interfaces
- Proper error handling and form validation
- Efficient API usage (no redundant requests)
- Clean, modular code with reusable components
- Good use of lifecycle hooks (via React hooks)
- UI/UX quality and responsiveness
- Successful deployment of both frontend and backend

Submission Requirements

1. Full Application Deployment (Mandatory)

Your **frontend and backend must be deployed** on any hosting service of your choice (Vercel, Netlify, Render, Railway, AWS, etc.).
Submissions **without live deployment will be rejected.**

2. Video Submission (Mandatory)

You must record **one complete video** covering **both**:

A. Deployment Explanation (Step-by-Step)

You must clearly explain each step of how you deployed your project.

Minimum required points to cover:

1. **Project Setup**
    - Folder structure
    - Dependencies
    - Installation steps
2. **Environment Variables**
    - What you used (API keys, DB URLs, JWT secrets, etc.)
    - How you configured them on the hosting platform
3. **Backend Deployment**
    - Platform used (e.g., Render/Railway/AWS)
    - Build command, start command
    - Database connectivity
    - Testing backend APIs after deployment (Postman / browser / terminal)
4. **Frontend Deployment**
    - Platform used (Vercel/Netlify/etc.)
    - Build process
    - Setting environment variables
    - Updating API base URL to the deployed backend
5. **Connecting Frontend & Backend**
    - How the frontend talks to the backend

- Live API calls shown in Network tab or console

6. **Validation**

    - Show that all features work correctly in the deployed environment
    - Explain final deployed URLs (Frontend + Backend)

B. Full Product Explanation (Feature Walkthrough)

You must demonstrate the actual working product.

Explain:

1. **Product Objective**

    - What problem your product solves
    - Who the end users are

2. **Architecture Overview**

    - Tech stack
    - High-level architecture (frontend, backend, database)
    - Key libraries/tools used
    - Why you chose this architecture (this lets candidates show innovation)

3. **Feature-by-Feature Demo**
   Show all features live, including:

    - User flows
    - CRUD operations
    - Form submissions
    - API interactions
    - Error handling
    - Any authentication/authorization
    - Any additional bonus features

4. **Innovation (Very Important)**
   Since most candidates may use AI tools, you must highlight:

    - What you uniquely built
    - What you improved or optimized
    - Your thought process

- Any non-trivial logic, architecture, or UI components you created

5. **Testing & Debugging**

    - Show how you validated the features
    - Explain any challenges and how you solved them

3. Final Submission Format

You must submit:

- **Frontend deployed URL**
- **Public GitHub repository link (frontend & backend)**
- **Video link (YouTube unlisted / gdrive)**

Submissions missing any of these will not be evaluated.

Task Submission

Note: The task must be completed within 48 hours from the time of assignment.