GameTheory Assignment Report

1. Introduction

The **Schedule Management Application** is designed to streamline the process of managing bookings for multiple sports courts. It provides users with a dynamic, scrollable grid that displays available and booked hourly time slots across various courts. Users can create bookings through an intuitive modal form, and each booking is associated with color-coded users for easy identification. This project integrates a **React frontend** with a **Node.js/Express backend** to handle bookings dynamically via REST APIs.

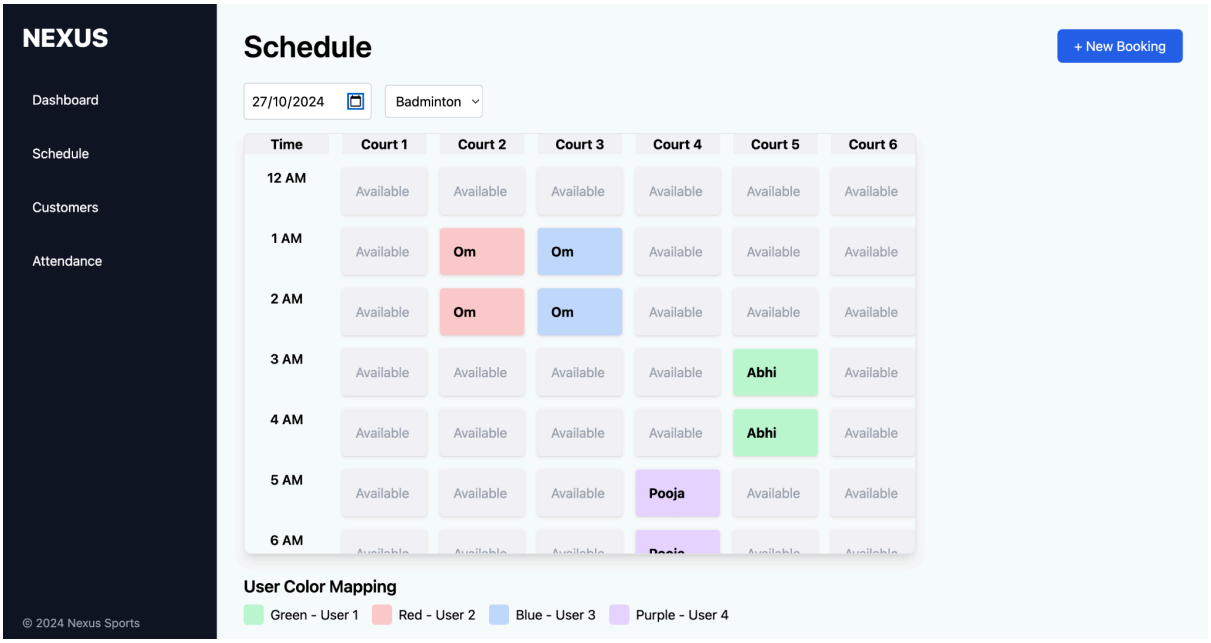
The main objective of the application is to facilitate efficient scheduling and improve the user experience by presenting bookings and availability clearly while maintaining responsiveness across devices.

2. Design Decisions

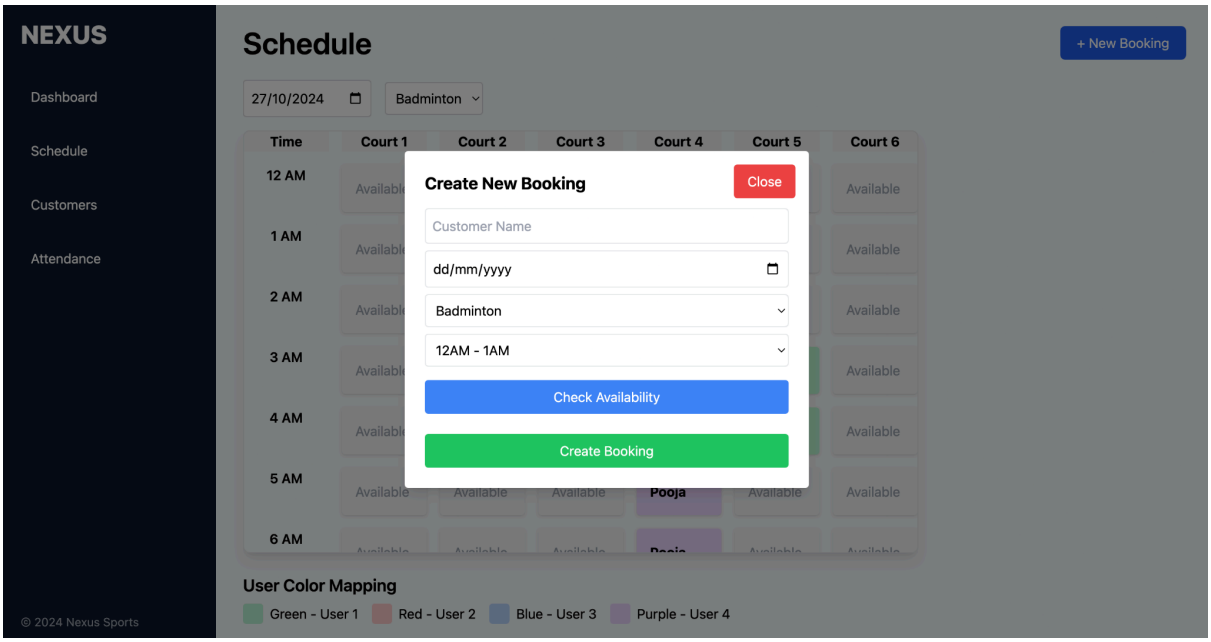
Several design decisions were made to ensure the application is user-friendly, responsive, and scalable. Below are the key decisions:

1. **Grid-based Layout**:
    - Each court is represented as a column, and the rows represent hourly time slots. This makes it easy to visualize availability and bookings.
    - **Sticky headers** for both time slots and court names enhance usability by keeping important information visible during scrolling.
2. **Color-coded User Legend**:
    - Users are assigned random colors for easy identification in the grid. A dynamic legend ensures that users can always recognize their bookings.
3. **Modal-based Booking Form**:
    - A modal form was chosen to maintain a clean interface without disrupting the schedule view. The form contains validation and dynamic court selection based on availability.
4. **Horizontal and Vertical Scrolling**:
    - Grid scrolling was implemented to handle large datasets (24 time slots per day for up to 6 courts) efficiently, without overwhelming the user.
5. **API-based Communication**:
    - The frontend communicates with the backend through **REST APIs** to fetch bookings and court availability in real time.

View Bookings :



Create Bookings :



3. Implementation Details

Technologies Used:

1. **Frontend:**
    - **React**: For building the user interface.
    - **Tailwind CSS**: For styling the components and ensuring responsiveness.
    - **Axios**: For making API requests to the backend.
2. **Backend:**
    - **Node.js** with **Express**: For building REST APIs to handle bookings and court availability.
    - **MongoDB**: (Assumed) For storing court and booking data.
3. **Hosting & Deployment:**
    - **Netlify/Vercel**: Recommended for hosting the frontend.
    - **Heroku/Railway**: Suggested for backend deployment.

Rationale for Technology Choices:

- **React** was chosen for its component-based architecture, which makes it easier to build and manage a modular UI.
- **Tailwind CSS** ensures a responsive and modern design without writing extensive custom CSS.
- **Node.js** and **Express** provide a lightweight backend, perfect for handling API requests and interacting with a database.
- **Axios** simplifies the process of making asynchronous HTTP requests from the frontend to the backend.

4. Challenges and Solutions

Challenge 1: Handling Dynamic Grid Layout

- **Problem**: Managing a grid with up to 6 courts and 24 hourly slots dynamically, including scrolling behavior.
- **Solution**: Implemented **grid-based scrolling** using Tailwind CSS. Sticky headers were used to keep time and court names visible during scrolling.

Challenge 2: User-specific Color Mapping

- **Problem**: Assigning colors dynamically to users without conflicts, especially when users change daily.
- **Solution**: Used a utility function to assign random colors and ensured consistent color mapping by storing the user-color pairs in state.

Challenge 3: Backend Communication and Availability Checks

- **Problem**: Ensuring the frontend displays accurate availability based on real-time data from the backend.
- **Solution**: Developed APIs to **fetch court counts and bookings** dynamically. Added **error handling** in both frontend and backend to manage failed API calls gracefully.

Challenge 4: Simultaneous Frontend and Backend Execution

- **Problem**: Ensuring both frontend and backend run seamlessly for local development and deployment.
- **Solution**: Used **npm scripts** to run both applications in parallel and ensured **CORS configuration** was correctly set for cross-origin requests.

5. Future Improvements

1. **User Authentication and Role Management**:
    - Implement user authentication with different roles (e.g., admin, user) to allow more controlled access to bookings and court management.
2. **Calendar Integration**:
    - Add integration with **Google Calendar** or similar services to allow users to sync their bookings automatically.
3. **Real-time Updates via WebSockets**:
    - Implement **WebSocket communication** to provide real-time booking updates without the need for refreshing the page.
4. **Advanced Filtering Options**:
    - Add more filtering options for bookings, such as filtering by user or booking status (e.g., confirmed, pending, canceled).
5. **Push Notifications**:
    - Introduce notifications to remind users of upcoming bookings or alert them to changes in availability.
6. **Mobile App Development**:
    - Expand the application by building a **React Native** mobile app to allow users to manage bookings on the go.

6. Conclusion

The **Schedule Management Application** provides an efficient and user-friendly way to manage bookings for sports courts. With its **scrollable grid layout**, **color-coded user mapping**, and **modal-based booking system**, it ensures a smooth user experience. The project demonstrates the integration of a **React frontend** with a **Node.js backend**, communicating seamlessly through REST APIs.

The challenges encountered during development were effectively addressed, and the system is ready for deployment with room for future enhancements. This project lays a solid foundation for managing court bookings dynamically, with potential for scalability and additional features in the future.