Game Theory Assignment Report

Introduction

This is a web-based booking management system designed for sports centers. It allows users to book courts for various sports, track their balance, and manage operational centers. This system leverages MongoDB, Node.js, and React.js, providing a scalable and user-friendly interface.

Design Decisions

The following design decisions were made to ensure the system is scalable, maintainable, secure, and user-friendly.

- **Card-Based UI for Summaries**
    - Each booking or center is displayed as a **card** summarizing key details like court number, booking time, and user.
    - Icons or labels (e.g., green for paid, red for unpaid) provide visual cues.
    - **Why**: Offers a clean, visual way to convey critical information at a glance.
- **Form Modal Dialogs**
    - Booking creation or editing is done through modal dialogs, ensuring users stay on the same page.
    - Dropdowns and date pickers are used for input consistency.
    - **Why**: Minimizes context switching and improves form usability.
- **API Route Management**
    - Dedicated route files for bookings, centers, sports, and users.
    - Routes are imported into app.js for centralized management.
    - **Why**: Simplifies route organization and integration of new modules.
- **Modular Architecture**
    - The application is structured using a **Model-Controller-Route** architecture:
        - **Models** define the database schema and relationships.
        - **Controllers** encapsulate business logic and handle API interactions.
        - **Routes** define the endpoints and link them to the respective controllers.
    - **Why**: Ensures separation of concerns, making the system easier to debug, extend, and scale.
- **Entities and Relationships**
    - **Booking Attributes**
        - **user**: Reference to the User model (ObjectId)
        - **location**: Reference to the Center model (ObjectId)
        - **sport**: Reference to the Sport model (ObjectId)
        - **court**: Number (court identifier)
        - **time**: String (time slot)
        - **date**: Date (booking date)
        - **status**: Enum (confirmed, cancelled, completed)
        - **isPaid**: Boolean (tracks payment status)

    - createdBy: Reference to the User model (ObjectId)

    - Relationships
        - **One-to-One with User:** A booking is associated with one user.
        - **Many-to-One with Center:** Multiple booking can occur at one center.
        - **Many-to-One with Sport:** Multiple booking can involves one sport.

- **Center Attributes**
    - **location:** String (center location)
    - **address**: String (detailed address of the center)
    - **contact**: String (e.g., phone or email)
    - **sports**: Array of references to the Sport model (ObjectId)
    - **isActive**: Boolean (whether the center is operational)

    - **Relationships**
        - **One-to-Many with Sport:** A center can host multiple sports.
        - **One-to-Many with Booking:** A center can have multiple bookings.

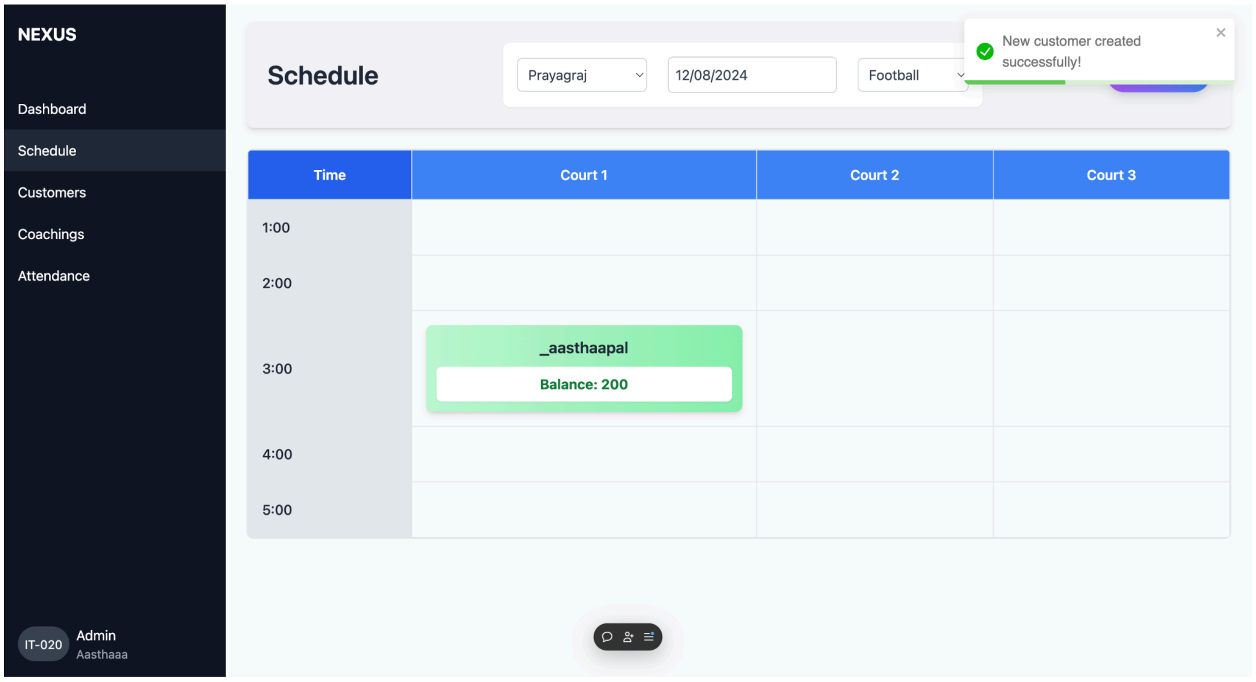
- **Sport Attributes**
    - **name**: String (e.g., Badminton, Tennis)
    - **courts**: Array of embedded documents (court details):
    - **courtNumber**: Number (identifier for each court)
    - **isActive**: Boolean (tracks court availability)
    - **timeSlots**: Array of strings (available booking times)
    - **operationalHours**:
        - start: String (e.g., "08:00")
        - End: String (e.g., "22:00")
    - **center**: Reference to the Center model (ObjectId)
    - **Relationships**
        - **Many-to-One with Center:** A sport is associated with one center.
        - **One-to-Many with Booking:** A sport can have multiple bookings.

- **User Attributes**
    - **username**: String (unique identifier for the user)
    - **PIN**: String (user's secure PIN for authentication)
    - **email**: String (unique email address)
    - **role**: Enum (customer, Admin)
    - **isActive**: Boolean (indicates if the user is active)
    - **balance**: Number (available credit or balance for bookings)
    - **Relationships**
        - **One-to-Many with Booking:** A user can have multiple bookings.

- **API Structure :** The API structure follows the RESTful principles to ensure a clean and modular approach. It is organized into multiple controllers and route files for each entity.

  - **Endpoints**:
    - **User API**
      - GET /users: Fetch all users.
      - POST /users: Create a new user.
    - **Center API**
      - GET /centers: Fetch all centers.
      - POST /centers: Add a new center.
    - **Sport API**
      - GET /sports: Fetch all sports.
      - POST /sports: Add a new sport.
    - **Booking API**
      - GET /bookings: Fetch all bookings (populates references).
      - POST /bookings: Create a new booking (validates time slot availability).
  - **Error Handling**
    - APIs use try-catch blocks for basic error handling.
    - Responses include appropriate HTTP status codes (200, 400, 404, 500).
    - Validation errors, conflicts (e.g., overlapping bookings), and server errors are handled gracefully

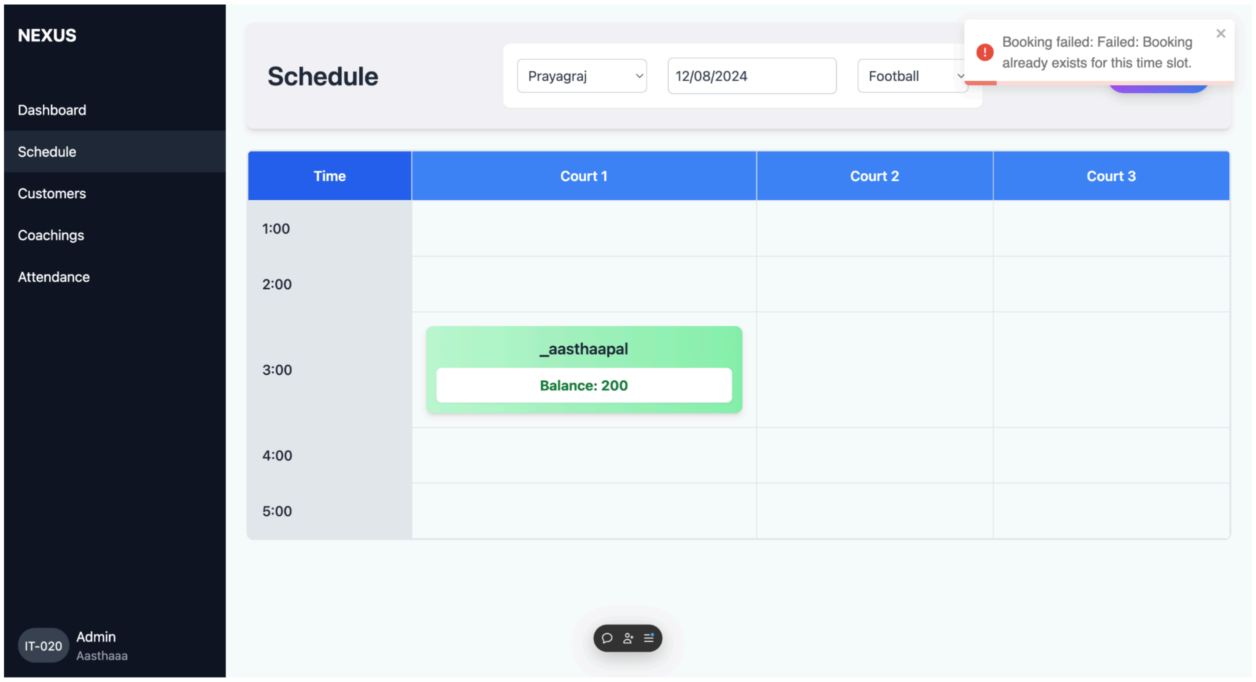
Add a New User



New customer created successfully!

Time	Court 1	Court 2	Court 3
1:00			
2:00			
3:00	_aasthaapal Balance: 200		
4:00			
5:00			

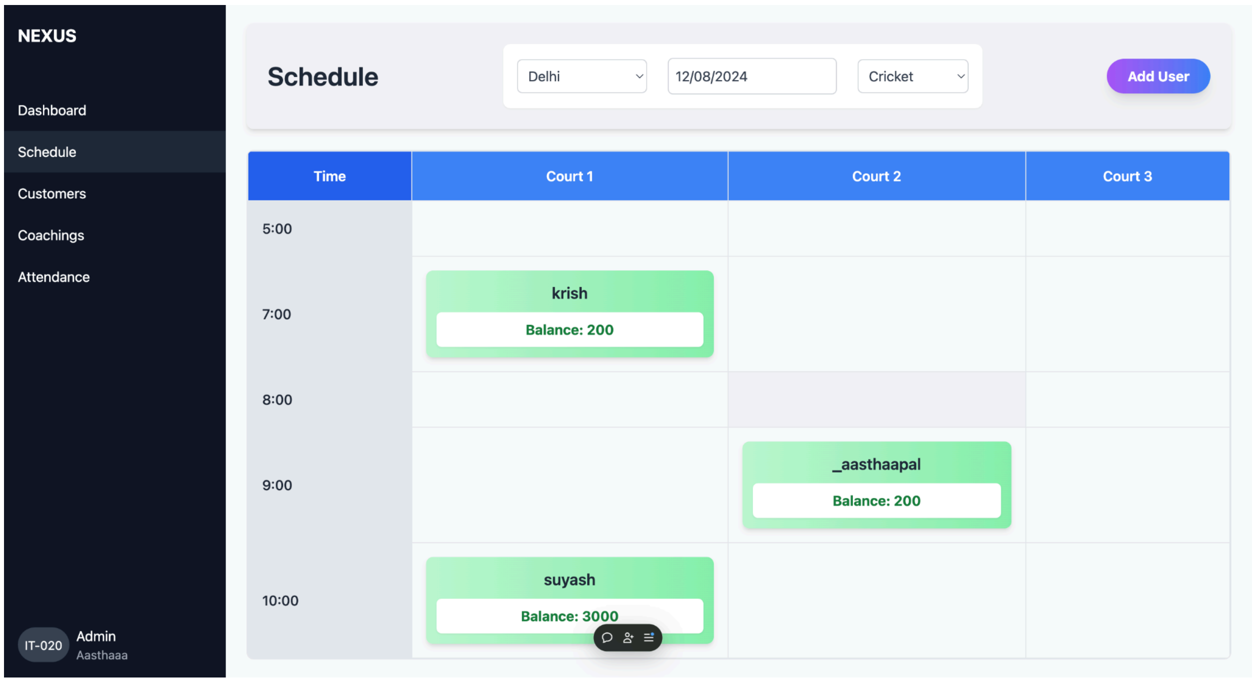
Booking on occupied court



Booking failed: Failed: Booking already exists for this time slot.

Time	Court 1	Court 2	Court 3
1:00			
2:00			
3:00	_aasthaapal Balance: 200		
4:00			
5:00			

View Bookings



Time	Court 1	Court 2	Court 3
5:00			
7:00	krish Balance: 200		
8:00			
9:00		_aasthaapal Balance: 200	
10:00	suyash Balance: 3000		

Implementation Details

Technologies Used

- **Frontend**
  - **React :** For building the user interface
  - **Vite**: As the build tool and development server
  - **Tailwind CSS**: For styling
  - **Axios**: For making API requests to the backend.
  - **React DatePicker**: For date selection functionality
  - **React Toastify**: For displaying notifications

- **Backend:** Node.js, Express.js : For building REST APIs to handle bookings and court availability.
- **Database:** MongoDB : For storing booking data.
- **Hosting & Development** : Vercel for frontend and Render for Backend.
- **Addition Libraries** : cors,body-parser,dot-env

Key Features

- **Dynamic Scheduling:**
  - Time slots and operational hours are customizable for each center and sport.
- **Error Handling:**
  - Prevent duplicate bookings with clear alerts (e.g., "Booking already exists").
- **Responsive Design:**
  - The application is designed to be responsive, with a sidebar for navigation on larger screens.
- **Customer Registration:**
  - New customers can be registered using the SignupModal.
- **Key Components:**
  - **Server Setup (app.js):**
    - Configures the Express application.
    - Sets up essential middleware:
      - **CORS**: Enables cross-origin resource sharing.
      - **Body-parser**: Parses incoming JSON request bodies.
    - Connect the application to MongoDB using Mongoose.
    - Defines API routes for different entities (/centers, /bookings, /sports, /users).

  - **Route Handlers:**
    Each route handler corresponds to a specific resource and is responsible for managing related operations:

    - **centers.js**: Handles operations for sports centers (e.g., creating and retrieving centers).
    - **bookings.js**: Manages booking creation, retrieval, and validation of time slots.

        - **sports.js**: Manages operations related to sports, such as creating sports, managing courts, and operational hours.
        - **users.js**: Handles user registration, login, and authentication processes.
    - **Models**:
      Defines the database schema for each entity using Mongoose:

        - **User.js**: Used for authentication and managing user accounts.
        - **Center.js**: Used for managing operational centers.
        - **Sport.js**: Linked to a specific center.
        - **Booking.js**: Ensures proper validation for bookings and prevents time-slot conflicts.
    - **Authentication**:
        - A **PIN-based authentication system** for user registration and login.
        - Secure **PIN hashing and comparison** implemented using **bcrypt**, ensuring encrypted storage and validation.

Challenges & Solutions

- **Conflict in Booking Time Slots:**
    - Implemented checks on the backend to prevent duplicate bookings.
    - Real-time updates on the frontend to reflect booking status.
- **Efficient State Management:**
    - Used Redux to handle user roles, balances, and schedule synchronization across components.
- **Styling Consistency:**
    - Adopted CSS Modules to prevent style conflicts and improve reusability.
- **Database Relationships:**
    - Utilized Mongoose's ref property to maintain relationships between users, centers, sports, and bookings.
- **API Error Handling**
    - A global error-handling middleware can be implemented, along with categorized error messages for better user feedback.

Future Improvements

- **Payment Integration:** Add online payment methods for seamless transactions.
- **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
- **Mobile App Support:** Develop a React Native app for mobile bookings.
- **Notification System:** Integrate email/SMS reminders for bookings and cancellations.