LIVE appointment bookings

In today's fast-paced world, managing healthcare appointments efficiently is crucial. With the rise of telemedicine and digital solutions, creating a seamless doctor appointment booking system can significantly enhance patient experience. In this blog post, we'll explore how to build a doctor appointment booking application using React, React Native, Node.js, and MongoDB. We'll focus on only live slot booking, for other functionality please visit our blog page.

There are different types of third-party services like(Socket.io, Firebase Realtime Database / Firestore) present by which we can implement live slot management. But here we using socket.io. Socket.io enables **real-time, bidirectional communication** between clients and servers. In a doctor appointment booking system, we can use Socket.io to instantly update the status of a slot once it's booked, ensuring that other users cannot book the same slot. Below is a brief explanation of how to implement this using **React Native**, **Node.js**, and **MongoDB**.

1. Backend Setup with Node.js and Socket.io

Start by setting up a Node.js server with **Socket.io** to handle real-time communication between the server and clients.

Install Socket.io and Express:

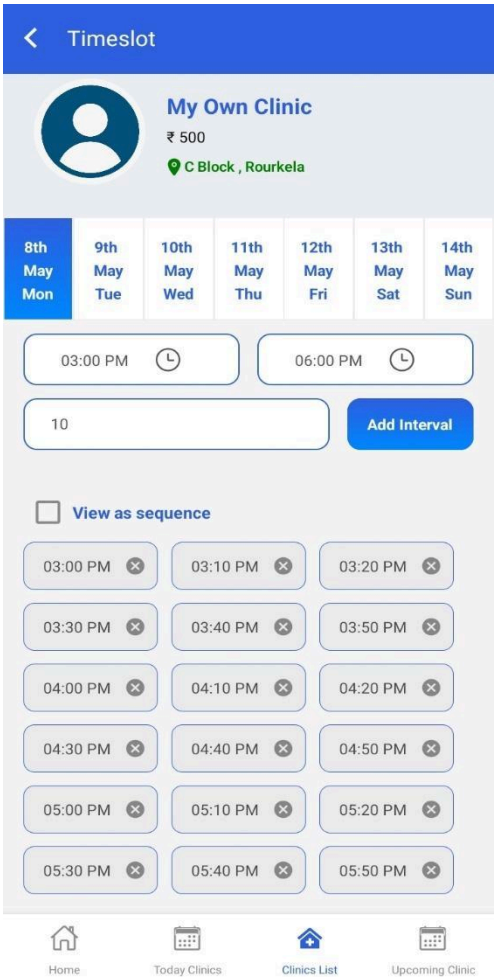
```
npm install express socket.io mongoose
```

Node.js Backend Example:

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const mongoose = require('mongoose');

// Create Express app and HTTP server
const app = express();
const server = http.createServer(app);
const io = socketIo(server);

// Connect to MongoDB
mongoose.connect('mongodb://localhost/doctor-appointment', { useNewUrlParser: true, useUnifi

// Schema for Appointment
const appointmentSchema = new mongoose.Schema({
  doctorId: String,
  patientId: String,
  timeSlot: Date,
  status: { type: String, default: 'Booked' }
});

const Appointment = mongoose.model('Appointment', appointmentSchema);

// API to book appointments
app.post('/book-appointment', async (req, res) => {
  const { doctorId, patientId, timeSlot } = req.body;

  const newAppointment = new Appointment({ doctorId, patientId, timeSlot });
  await newAppointment.save();

  // Emit event to update all clients about the booked slot
  io.emit('slot-booked', { doctorId, timeSlot });

  res.json({ success: true, message: 'Appointment booked successfully!' });
});
```

```
// Socket.io connection handler
io.on('connection', (socket) => {
  console.log('New client connected');

  socket.on('disconnect', () => {
    console.log('Client disconnected');
  });
});

// Start server
server.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

Create slots Mobile app UI



Explanation:

- When a new appointment is booked through the /book-appointment API, the backend saves it in **MongoDB** and emits a slot-booked event through Socket.io.
- All connected clients (React Native apps) will be notified in real-time that the slot is now unavailable.

2. React Native Client Setup with Socket.io

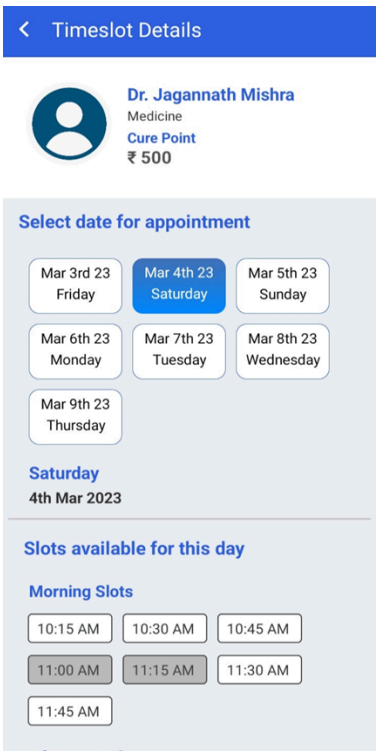
- In the React Native app, we listen to the `slot-booked` event emitted by the server, and update the available slots accordingly.
- **Install Socket.io Client:**

```
npm install socket.io-client
```

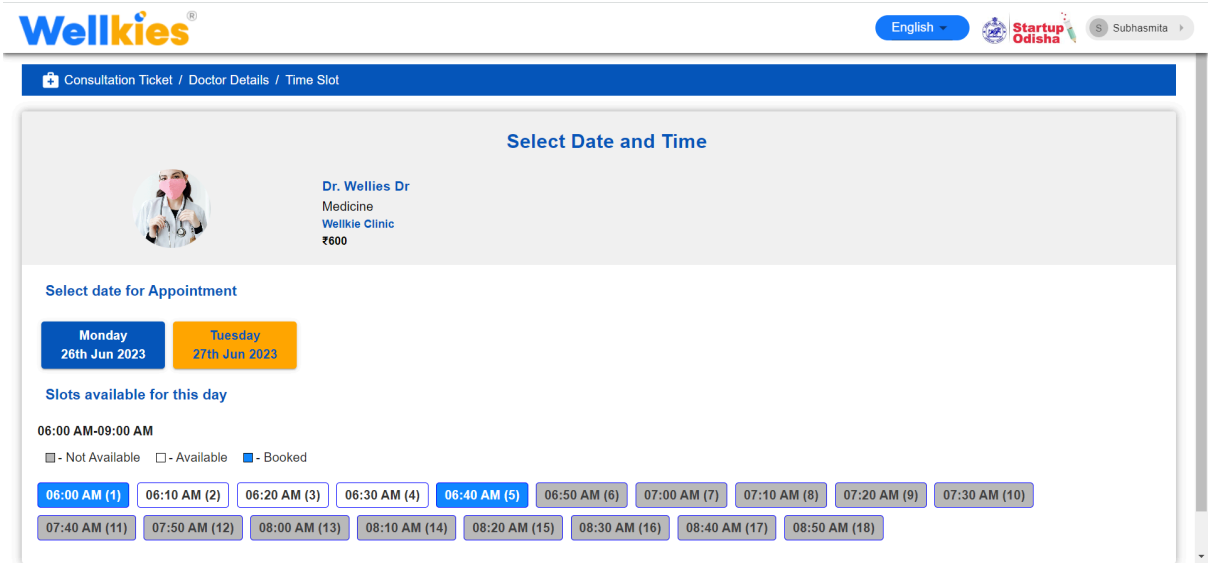
```jsx
import React, { useEffect, useState } from 'react';
import { View, Text, Button } from 'react-native';
import io from 'socket.io-client';

// URL of the Node.js server
const socket = io('http://localhost:3000');

const AppointmentScreen = () => {
  const [availableSlots, setAvailableSlots] = useState([
    { doctorId: '1', timeSlot: '2024-09-23T10:00', available: true },
    { doctorId: '1', timeSlot: '2024-09-23T11:00', available: true },
    // ...more slots
  ]);

  // Listen for real-time updates from the server
  useEffect(() => {
    socket.on('slot-booked', (data) => {
      // Update the availableSlots array to mark the booked slot as unavailable
      setAvailableSlots((prevSlots) =>
        prevSlots.map(slot =>
          slot.doctorId === data.doctorId && slot.timeSlot === data.timeSlot
            ? { ...slot, available: false }
            : slot
        )
      );
    });

    // Clean up the socket listener on component unmount
    return () => socket.off('slot-booked');
  }, []);
```

```jsx
  const bookSlot = async (doctorId, timeSlot) => {
    // Call backend API to book the slot
    const response = await fetch('http://localhost:3000/book-appointment', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ doctorId, patientId: '123', timeSlot })
    });
    const data = await response.json();
    if (data.success) {
      alert('Slot booked!');
    }
  };

  return (
    <View>
      {availableSlots.map(slot => (
        <View key={slot.timeSlot}>
          <Text>{slot.timeSlot} - {slot.available ? 'Available' : 'Booked'}</Text>
          {slot.available && (
            <Button title="Book Slot" onPress={() => bookSlot(slot.doctorId, slot.timeSlot)} />
          )}
        </View>
      ))}
    </View>
  );
};

export default AppointmentScreen;
```

Mobile App UI



Webapp UI



Explanation:

- **Socket.io Client**: React Native connects to the server via Socket.io to receive real-time updates.
- **Booking Slots**: When a slot is booked, the app updates the backend via an API request. Upon success, the server notifies all connected clients, and the booked slot is marked as unavailable.

- **Real-time Updates**: Whenever a slot is booked by any user, all clients automatically update the slot availability.

3. MongoDB for Storing Appointments

In **MongoDB**, all appointment data is stored, including the doctor, patient, time slot, and booking status. The real-time updates ensure data consistency across multiple clients.

Conclusion

By integrating **Socket.io** in this doctor appointment booking app, you ensure that users can see real-time updates of slot availability. React Native handles the front end, **Node.js** manages server-side logic, and **MongoDB** stores the data. This ensures a smooth, real-time booking experience for users, reducing the risk of double bookings and improving overall efficiency.