we were working on a project attendance management system ( law\_college) , where we were doing backend in django ( rest framework ) and frotend in react we desigend some backend components like schema and , serializer , view part and some frontend part I'll provide you that code so that you can get proper context and further guide me in a efficient way. our project name is programme\_attendance , and app name is "app" . here is a models

app.model>

from django.db import models

from django.contrib.auth.models import User

from django.core.exceptions import ValidationError

from django.core.validators import RegexValidator

# Validate lecture time slots

LECTURE\_SLOTS = [

    ('08:30:00', '08:30 AM - 09:30 AM'),

    ('09:30:00', '09:30 AM - 10:30 AM'),

    ('10:30:00', '10:30 AM - 11:30 AM'),

    ('12:00:00', '12:00 PM - 01:00 PM'),

    ('13:00:00', '01:00 PM - 02:00 PM'),

]

class Program(models.Model):

    name = models.CharField(max\_length=50, unique=True)  # e.g., "BALLB 5 Yr", "LLB 3 Yr"

    duration\_years = models.PositiveIntegerField()  # 3 or 5

    def \_\_str\_\_(self):

        return self.name

    class Meta:

        verbose\_name = "Program"

        verbose\_name\_plural = "Programs"

class Section(models.Model):

    program = models.ForeignKey(Program, on\_delete=models.CASCADE, related\_name="sections")

    name = models.CharField(max\_length=10)  # e.g., "Section A", "Section B"

    year = models.PositiveIntegerField()  # 1, 2, 3, 4, 5

    def \_\_str\_\_(self):

        return f"{self.program} - Year {self.year} - {self.name}"

    class Meta:

        unique\_together = ('program', 'year', 'name')

        verbose\_name = "Section"

        verbose\_name\_plural = "Sections"

class Subject(models.Model):

    name = models.CharField(max\_length=100, unique=True)  # e.g., "Law of Contract I"

    is\_law\_subject = models.BooleanField(default=True)  # True for law, False for non-law

    semester = models.PositiveIntegerField()  # 1 to 10 (depending on program)

    def \_\_str\_\_(self):

        return self.name

    class Meta:

        verbose\_name = "Subject"

        verbose\_name\_plural = "Subjects"

class Teacher(models.Model):

    user = models.OneToOneField(User, on\_delete=models.CASCADE)  # Authentication

    first\_name = models.CharField(max\_length=50)

    last\_name = models.CharField(max\_length=50)

    email = models.EmailField(unique=True)

    phone = models.CharField(max\_length=15, null=True, blank=True)

    def \_\_str\_\_(self):

        return f"{self.first\_name} {self.last\_name}"

    class Meta:

        verbose\_name = "Teacher"

        verbose\_name\_plural = "Teachers"

class Student(models.Model):

    roll\_number = models.CharField(

        max\_length=10,

        unique=True,

        validators=[RegexValidator(r'^(G|NG)24\d{4}$', 'Roll number must be G24xxxx or NG24xxxx')]

    )  # e.g., G240001, NG240012

    first\_name = models.CharField(max\_length=50)

    last\_name = models.CharField(max\_length=50)

    email = models.EmailField(unique=True)

    phone = models.CharField(max\_length = 15)

    section = models.ForeignKey(Section, on\_delete=models.CASCADE, related\_name="students")

    subjects = models.ManyToManyField(Subject, related\_name="students", blank=True)

    def \_\_str\_\_(self):

        return f"{self.roll\_number} - {self.first\_name} {self.last\_name}"

    class Meta:

        verbose\_name = "Student"

        verbose\_name\_plural = "Students"

class Timetable(models.Model):

    DAY\_CHOICES = [

        ('Monday', 'Monday'),

        ('Tuesday', 'Tuesday'),

        ('Wednesday', 'Wednesday'),

        ('Thursday', 'Thursday'),

        ('Friday', 'Friday'),

        ('Saturday', 'Saturday'),

    ]

    section = models.ForeignKey(Section, on\_delete=models.CASCADE, related\_name="timetable")

    subject = models.ForeignKey(Subject, on\_delete=models.CASCADE, related\_name="timetable")

    teacher = models.ForeignKey(Teacher, on\_delete=models.SET\_NULL, null=True, related\_name="timetable")

    day\_of\_week = models.CharField(max\_length=9, choices=DAY\_CHOICES)

    start\_time = models.TimeField(choices=LECTURE\_SLOTS)  # Fixed lecture slots

    semester\_start\_date = models.DateField()

    semester\_end\_date = models.DateField()

    def \_\_str\_\_(self):

        return f"{self.section} - {self.subject} ({self.day\_of\_week} {self.start\_time})"

    class Meta:

        unique\_together = ('section', 'subject', 'day\_of\_week', 'start\_time')

        verbose\_name = "Timetable"

        verbose\_name\_plural = "Timetables"

class Session(models.Model):

    STATUS\_CHOICES = [

        ('Scheduled', 'Scheduled'),

        ('Completed', 'Completed'),

        ('Cancelled', 'Cancelled'),

    ]

    timetable = models.ForeignKey(Timetable, on\_delete=models.CASCADE, related\_name="sessions")

    date = models.DateField()

    status = models.CharField(max\_length=10, choices=STATUS\_CHOICES, default='Scheduled')

    def \_\_str\_\_(self):

        return f"{self.timetable} on {self.date} ({self.status})"

    class Meta:

        unique\_together = ('timetable', 'date')

        verbose\_name = "Session"

        verbose\_name\_plural = "Sessions"

class Attendance(models.Model):

    STATUS\_CHOICES = [

        ('Present', 'Present'),

        ('Absent', 'Absent'),

    ]

    student = models.ForeignKey(Student, on\_delete=models.CASCADE, related\_name="attendance")

    session = models.ForeignKey(Session, on\_delete=models.CASCADE, related\_name="attendance")

    status = models.CharField(max\_length=7, choices=STATUS\_CHOICES, default='Absent')

    timestamp = models.DateTimeField(auto\_now=True)

    recorded\_by = models.ForeignKey(Teacher, on\_delete=models.SET\_NULL, null=True, related\_name="attendance\_records")

    def \_\_str\_\_(self):

        return f"{self.student} - {self.session} - {self.status}"

    class Meta:

        unique\_together = ('student', 'session')

        verbose\_name = "Attendance"

        verbose\_name\_plural = "Attendance"

class CalendarException(models.Model):

    date = models.DateField(unique=True)

    description = models.CharField(max\_length=100)

    def \_\_str\_\_(self):

        return f"{self.date} - {self.description}"

    class Meta:

        verbose\_name = "Calendar Exception"

        verbose\_name\_plural = "Calendar Exceptions"

app.serializer.py>>>

from rest\_framework import serializers

from .models import Teacher,Student, Subject , Timetable , Session, Attendance , Section , CalendarException

class TeacherSerializer(serializers.ModelSerializer):

    class Meta:

        model = Teacher

        fields = ['id' , 'first\_name' , 'last\_name' , 'email','phone']

class StudentSerializer(serializers.ModelSerializer):

    section = serializers.StringRelatedField()

    subjects = serializers.PrimaryKeyRelatedField(many = True , read\_only = True)

    class Meta:

        model = Student

        fields = ['id','roll\_number','first\_name','last\_name','email','phone','section','subjects']

class SubjectSerializer(serializers.ModelSerializer):

    teacher = TeacherSerializer(read\_only = True)

    class Meta:

        model = Subject

        fields = ['id' , 'name' ,'is\_law\_subject', 'semester','teacher']

class TimetableSerializer(serializers.ModelSerializer):

    section = serializers.StringRelatedField()

    subject = SubjectSerializer(read\_only = True)

    teacher = TeacherSerializer(read\_only = True)

    class Meta:

        model = Timetable

        fields = ['id','section','subject','teacher','day\_of\_week','start\_time', 'semester\_start\_date', 'semester\_end\_date']

class SessionSerializer(serializers.ModelSerializer):

    timetable = TimetableSerializer(read\_only=True)

    class Meta:

        model = Session

        fields = ['id','timetable','date','status']

class AttendanceSerializer(serializers.ModelSerializer):

    student = StudentSerializer(read\_only=True)

    session = SessionSerializer(read\_only=True)

    recorded\_by = TeacherSerializer(read\_only=True)

    class Meta:

        model = Attendance

        fields = ['id','student','session','status','timestamp','recorded\_by']

        read\_only\_fields = ['timestamp' , 'recorded\_by']

class CalendarExceptionSerializer(serializers.ModelSerializer):

    class Meta:

        model = CalendarException

        fields = ['id','date','description']

app.views.py>>>

from django.shortcuts import render

from django.http import HttpResponse

from rest\_framework import viewsets, generics, status

from rest\_framework.permissions import IsAuthenticated

from rest\_framework.response import Response

from django.db.models import Count, Q

from datetime import datetime, timedelta

from .models import Session, Attendance, Teacher, Student, Subject, Timetable, CalendarException

from .serializers import (

    SessionSerializer, AttendanceSerializer, TeacherSerializer, StudentSerializer,

    TimetableSerializer, CalendarExceptionSerializer

)

# Create your views here.

def home(request):

    return HttpResponse('hello world this is home')

class StudentViewSet(viewsets.ReadOnlyModelViewSet):

    queryset = Student.objects.all()

    serializer\_class = StudentSerializer

    permission\_classes = [IsAuthenticated]

    def get\_queryset(self):

        # Optionally resistrict to studetns in teachers's section

        return Student.objects.filter(section\_\_timetable\_\_teacher\_\_user = self.request.user ).distinct()

class SessionViewSet(viewsets.ModelViewSet):

    queryset = Session.objects.all()

    serializer\_class = SessionSerializer

    permission\_classes = [IsAuthenticated]

    def get\_queryset(self):

        # Restrict to logged-in teacher’s sessions

        return Session.objects.filter(timetable\_\_teacher\_\_user=self.request.user)

class AttendanceViewSet(viewsets.ModelViewSet):

    queryset = Attendance.objects.all()

    serializer\_class = AttendanceSerializer

    permission\_classes = [IsAuthenticated]

    def perform\_create(self, serializer):

        serializer.save(recorded\_by=Teacher.objects.get(user=self.request.user))

class TeacherCalendarView(generics.ListAPIView):

    serializer\_class = SessionSerializer

    permission\_classes = [IsAuthenticated]

    def get\_queryset(self):

        # Return sessions for the logged-in teacher

        return Session.objects.filter(timetable\_\_teacher\_\_user=self.request.user).order\_by('date')

class MarkAttendanceView(generics.GenericAPIView):

    serializer\_class = AttendanceSerializer

    permission\_classes = [IsAuthenticated]

    def post(self, request, session\_id):

        try:

            session = Session.objects.get(id=session\_id)

            teacher = Teacher.objects.get(user=request.user)

            if session.timetable.teacher != teacher:

                return Response({"error": "Not authorized to mark this session"}, status=status.HTTP\_403\_FORBIDDEN)

            # Expecting payload: [{"student\_id": 1, "status": "Present"}, ...]

            attendance\_data = request.data.get('attendance', [])

            for entry in attendance\_data:

                student = Student.objects.get(id=entry['student\_id'])

                # Only students in the section can be marked

                if student.section != session.timetable.section:

                    continue

                Attendance.objects.update\_or\_create(

                    student=student,

                    session=session,

                    defaults={'status': entry['status'], 'recorded\_by': teacher}

                )

            session.status = 'Completed'

            session.save()

            return Response({"message": "Attendance marked successfully"}, status=status.HTTP\_200\_OK)

        except Session.DoesNotExist:

            return Response({"error": "Session not found"}, status=status.HTTP\_404\_NOT\_FOUND)

        except Student.DoesNotExist:

            return Response({"error": "Invalid student ID"}, status=status.HTTP\_400\_BAD\_REQUEST)

class HolidayListCreateView(generics.ListCreateAPIView):

    queryset = CalendarException.objects.all()

    serializer\_class = CalendarExceptionSerializer

    permission\_classes = [IsAuthenticated]  # Add admin check later if needed

    def perform\_create(self, serializer):

        holiday = serializer.save()

        # Cancel sessions on this date

        Session.objects.filter(date=holiday.date).update(status='Cancelled')

class AttendanceStatsView(generics.GenericAPIView):

    permission\_classes = [IsAuthenticated]

    def get(self, request, roll\_number):

        try:

            student = Student.objects.get(roll\_number=roll\_number)

            section = student.section

            # Get all sessions for the student’s section

            sessions = Session.objects.filter(

                timetable\_\_section=section, status='Completed'

            )

            total\_sessions = sessions.count()

            attendance = Attendance.objects.filter(

                student=student, session\_\_in=sessions

            ).aggregate(

                present=Count('id', filter=Q(status='Present')),

                absent=Count('id', filter=Q(status='Absent'))

            )

            # Day-wise, Week-wise, Month-wise (example: month-wise)

            monthly\_stats = Attendance.objects.filter(

                student=student, session\_\_in=sessions

            ).extra(

                select={'month': "EXTRACT(MONTH FROM session\_date)"}

            ).values('month').annotate(

                present=Count('id', filter=Q(status='Present')),

                absent=Count('id', filter=Q(status='Absent'))

            )

            # Consolidated display (e.g., "P P A P A")

            attendance\_list = [

                'P' if a.status == 'Present' else 'A'

                for a in Attendance.objects.filter(student=student, session\_\_in=sessions).order\_by('session\_\_date')

            ]

            return Response({

                'student': StudentSerializer(student).data,

                'total\_sessions': total\_sessions,

                'present': attendance['present'] or 0,

                'absent': attendance['absent'] or 0,

                'percentage': (attendance['present'] / total\_sessions \* 100) if total\_sessions > 0 else 0,

                'monthly\_stats': list(monthly\_stats),

                'consolidated': ' '.join(attendance\_list[:5])  # First 5 lectures as example

            })

        except Student.DoesNotExist:

            return Response({"error": "Student not found"}, status=status.HTTP\_404\_NOT\_FOUND)

class ClassHourlyStatsView(generics.GenericAPIView):

    permission\_classes = [IsAuthenticated]

    def get(self, request, session\_id):

        try:

            session = Session.objects.get(id=session\_id)

            if session.timetable.teacher != Teacher.objects.get(user=request.user):

                return Response({"error": "Not authorized"}, status=status.HTTP\_403\_FORBIDDEN)

            attendance = Attendance.objects.filter(session=session).aggregate(

                present=Count('id', filter=Q(status='Present')),

                absent=Count('id', filter=Q(status='Absent'))

            )

            total\_students = Student.objects.filter(section=session.timetable.section).count()

            return Response({

                'session': SessionSerializer(session).data,

                'present': attendance['present'] or 0,

                'absent': attendance['absent'] or 0,

                'total\_students': total\_students

            })

        except Session.DoesNotExist:

            return Response({"error": "Session not found"}, status=status.HTTP\_404\_NOT\_FOUND)

app.urls.py >>>

from django.urls import path,include

from . import views

from rest\_framework.routers import DefaultRouter

router = DefaultRouter()

router.register(r'sessions',views.SessionViewSet)

router.register(r'attendance',views.AttendanceViewSet)

router.register(r'students', views.StudentViewSet)

urlpatterns = [

    # path('', views.home , name = 'home')

    path('', include(router.urls)),

    path('calendar/', views.TeacherCalendarView.as\_view(), name='teacher\_calendar'),

    path('mark-attendance/<int:session\_id>/', views.MarkAttendanceView.as\_view(), name='mark\_attendance'),

    path('holidays/', views.HolidayListCreateView.as\_view(), name='holiday\_list\_create'),

    path('attendance-stats/<str:roll\_number>/', views.AttendanceStatsView.as\_view(), name='attendance\_stats'),

    path('hourly-stats/<int:session\_id>/', views.ClassHourlyStatsView.as\_view(), name='hourly\_stats'),]

programme\_attendance.urls.py >>

from django.contrib import admin

from django.urls import path , include

from rest\_framework\_simplejwt.views import TokenObtainPairView , TokenRefreshView

urlpatterns = [

    path("admin/", admin.site.urls),

    path('api/',include('app.urls')),

    path('api/token/',TokenObtainPairView.as\_view() , name = 'token\_obtain\_pair'),

    path('api/token?refresh/',TokenRefreshView.as\_view() , name = 'token\_refresh')

]

So far we’ve come in backend development.

Frontend >>

AttendanceForm.jsx >>>

import React, { useState, useEffect } from 'react';

import axios from 'axios';

function AttendanceForm({ session, onClose }) {

  const [students, setStudents] = useState([]);

  const [attendanceData, setAttendanceData] = useState({});

  const [error, setError] = useState('');

  const [success, setSuccess] = useState('');

  console.log(session)

  useEffect(() => {

    const fetchStudents = async () => {

      const token = localStorage.getItem('access\_token');

      try {

        // Fetch session details to get section

        const sessionResponse = await axios.get(`http://localhost:8000/api/sessions/${session.id}/`, {

          headers: { Authorization: `Bearer ${token}` },

        });

        const sectionId = sessionResponse.data.timetable.section.split(' - ')[1]; // Extract section ID if needed

        // Fetch students in the section (assuming section is a string; adjust if ID-based)

        const studentsResponse = await axios.get('http://localhost:8000/api/students/', {

          headers: { Authorization: `Bearer ${token}` },

        });

        const sectionStudents = studentsResponse.data.filter(

          (student) => student.section === sessionResponse.data.timetable.section

        );

        setStudents(sectionStudents);

        // Initialize attendance data with default "Absent"

        const initialAttendance = {};

        sectionStudents.forEach((student) => {

          initialAttendance[student.id] = 'Absent';

        });

        setAttendanceData(initialAttendance);

      } catch (err) {

        setError('Failed to load students');

        console.error(err);

      }

    };

    fetchStudents();

  }, [session]);

  const handleStatusChange = (studentId, status) => {

    setAttendanceData((prev) => ({

      ...prev,

      [studentId]: status,

    }));

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    const token = localStorage.getItem('access\_token');

    const payload = Object.keys(attendanceData).map((studentId) => ({

      student\_id: parseInt(studentId),

      status: attendanceData[studentId],

    }));

    try {

      const response = await axios.post(

        `http://localhost:8000/api/mark-attendance/${session.id}/`,

        { attendance: payload },

        { headers: { Authorization: `Bearer ${token}` } }

      );

      setSuccess(response.data.message);

      setError('');

      setTimeout(onClose, 2000); // Close form after 2 seconds

    } catch (err) {

      setError(err.response?.data?.error || 'Failed to mark attendance');

      setSuccess('');

    }

  };

  return (

    <div style={{

      position: 'fixed', top: '50%', left: '50%', transform: 'translate(-50%, -50%)',

      background: 'white', padding: '20px', borderRadius: '8px', boxShadow: '0 0 10px rgba(0,0,0,0.3)',

      zIndex: 1000, maxWidth: '600px', width: '100%'

    }}>

      <h3>Mark Attendance for {session.title}</h3>

      {error && <p style={{ color: 'red' }}>{error}</p>}

      {success && <p style={{ color: 'green' }}>{success}</p>}

      <form onSubmit={handleSubmit}>

        <table style={{ width: '100%', borderCollapse: 'collapse' }}>

          <thead>

            <tr>

              <th style={{ border: '1px solid #ddd', padding: '8px' }}>Roll Number</th>

              <th style={{ border: '1px solid #ddd', padding: '8px' }}>Name</th>

              <th style={{ border: '1px solid #ddd', padding: '8px' }}>Status</th>

            </tr>

          </thead>

          <tbody>

            {students.map((student) => (

              <tr key={student.id}>

                <td style={{ border: '1px solid #ddd', padding: '8px' }}>{student.roll\_number}</td>

                <td style={{ border: '1px solid #ddd', padding: '8px' }}>{student.first\_name} {student.last\_name}</td>

                <td style={{ border: '1px solid #ddd', padding: '8px' }}>

                  <select

                    value={attendanceData[student.id] || 'Absent'}

                    onChange={(e) => handleStatusChange(student.id, e.target.value)}

                    style={{ padding: '4px' }}

                  >

                    <option value="Absent">Absent</option>

                    <option value="Present">Present</option>

                  </select>

                </td>

              </tr>

            ))}

          </tbody>

        </table>

        <div style={{ marginTop: '20px', textAlign: 'right' }}>

          <button type="button" onClick={onClose} style={{ padding: '8px 16px', marginRight: '10px' }}>

            Cancel

          </button>

          <button type="submit" style={{ padding: '8px 16px' }}>Save Attendance</button>

        </div>

      </form>

    </div>

  );

}

export default AttendanceForm;

Calendar.jsx>>

/\*

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import FullCalendar from '@fullcalendar/react';

import dayGridPlugin from '@fullcalendar/daygrid';

import timeGridPlugin from '@fullcalendar/timegrid';

function Calendar() {

  const [events, setEvents] = useState([]);

  const [error, setError] = useState('');

  useEffect(() => {

    const fetchSessions = async () => {

      const token = localStorage.getItem('access\_token');

      if (!token) {

        setError('No authentication token found. Please log in.');

        return;

      }

      try {

        const response = await axios.get('http://localhost:8000/api/calendar/', {

          headers: {

            Authorization: `Bearer ${token}`,

          },

        });

        const sessionEvents = response.data.map((session) => ({

          title: `${session.timetable.subject.name} (${session.status})`,

          start: `${session.date}T${session.timetable.start\_time}`,

          end: `${session.date}T${new Date(`1970-01-01T${session.timetable.start\_time}`).getTime() + 60\*60\*1000}`.slice(0, -1), // Add 1 hour

          extendedProps: {

            sessionId: session.id,

            status: session.status,

          },

        }));

        setEvents(sessionEvents);

      } catch (err) {

        setError('Failed to fetch sessions. Please try again.');

        console.error(err);

      }

    };

    fetchSessions();

  }, []);

  return (

    <div style={{ maxWidth: '1000px', margin: '50px auto' }}>

      <h2>Your Teaching Schedule</h2>

      {error && <p style={{ color: 'red' }}>{error}</p>}

      <FullCalendar

        plugins={[dayGridPlugin, timeGridPlugin]}

        initialView="timeGridWeek"

        events={events}

        slotMinTime="08:00:00"

        slotMaxTime="14:00:00"

        headerToolbar={{

          left: 'prev,next today',

          center: 'title',

          right: 'dayGridMonth,timeGridWeek,timeGridDay',

        }}

        eventClick={(info) => {

          alert(`Session ID: ${info.event.extendedProps.sessionId}\nSubject: ${info.event.title}\nStatus: ${info.event.extendedProps.status}`);

        }}

      />

    </div>

  );

}

export default Calendar;

\*/

import React, { useState, useEffect } from 'react';

import axios from 'axios';

import FullCalendar from '@fullcalendar/react';

import dayGridPlugin from '@fullcalendar/daygrid';

import timeGridPlugin from '@fullcalendar/timegrid';

import AttendanceForm from './AttendanceForm';

function Calendar() {

  const [events, setEvents] = useState([]);

  const [error, setError] = useState('');

  const [selectedSession, setSelectedSession] = useState(null);

  useEffect(() => {

    const fetchSessions = async () => {

      const token = localStorage.getItem('access\_token');

      if (!token) {

        setError('No authentication token found. Please log in.');

        return;

      }

      try {

        const response = await axios.get('http://localhost:8000/api/calendar/', {

          headers: { Authorization: `Bearer ${token}` },

        });

        const sessionEvents = response.data.map((session) => ({

          title: `${session.timetable.subject.name} (${session.status})`,

          start: `${session.date}T${session.timetable.start\_time}`,

          end: `${session.date}T${new Date(`1970-01-01T${session.timetable.start\_time}`).getTime() + 60\*60\*1000}`.slice(0, -1),

          extendedProps: {

            sessionId: session.id,

            status: session.status,

          },

        }));

        setEvents(sessionEvents);

      } catch (err) {

        setError('Failed to fetch sessions');

        console.error(err);

      }

    };

    fetchSessions();

  }, []);

  const handleEventClick = (info) => {

    if (info.event.extendedProps.status !== 'Scheduled') {

      alert('Attendance can only be marked for Scheduled sessions.');

      return;

    }

    setSelectedSession({

      id: info.event.extendedProps.sessionId,

      title: info.event.title,

    });

  };

  const closeAttendanceForm = () => {

    setSelectedSession(null);

    // Refresh calendar after marking attendance

    const token = localStorage.getItem('access\_token');

    axios.get('http://localhost:8000/api/calendar/', {

      headers: { Authorization: `Bearer ${token}` },

    }).then((response) => {

      const updatedEvents = response.data.map((session) => ({

        title: `${session.timetable.subject.name} (${session.status})`,

        start: `${session.date}T${session.timetable.start\_time}`,

        end: `${session.date}T${new Date(`1970-01-01T${session.timetable.start\_time}`).getTime() + 60\*60\*1000}`.slice(0, -1),

        extendedProps: {

          sessionId: session.id,

          status: session.status,

        },

      }));

      setEvents(updatedEvents);

    });

  };

  return (

    <div style={{ maxWidth: '1000px', margin: '50px auto' }}>

      <h2>Your Teaching Schedule</h2>

      {error && <p style={{ color: 'red' }}>{error}</p>}

      <FullCalendar

        plugins={[dayGridPlugin, timeGridPlugin]}

        initialView="timeGridWeek"

        events={events}

        slotMinTime="08:00:00"

        slotMaxTime="14:00:00"

        headerToolbar={{

          left: 'prev,next today',

          center: 'title',

          right: 'dayGridMonth,timeGridWeek,timeGridDay',

        }}

        eventClick={handleEventClick}

      />

      {selectedSession && (

        <AttendanceForm session={selectedSession} onClose={closeAttendanceForm} />

      )}

    </div>

  );

}

export default Calendar;

App.jsx >>>

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Login from './components/Login';

import Calendar from './components/Calendar';

import AttendanceForm from './components/AttendanceForm';

function App() {

  return (

    <Router>

      <Routes>

        <Route path="/" element={<Login />} />

        <Route path="/calendar" element={<Calendar />} />

        <Route path='/attendance/form' element={<AttendanceForm/>} />

      </Routes>

    </Router>

  );

}

export default App;