# IntelliSprout Project Code Report

## IntelliSprout/client/eslint.config.js

import js from '@eslint/js'  
import globals from 'globals'  
import react from 'eslint-plugin-react'  
import reactHooks from 'eslint-plugin-react-hooks'  
import reactRefresh from 'eslint-plugin-react-refresh'  
  
export default [  
 { ignores: ['dist'] },  
 {  
 files: ['\*\*/\*.{js,jsx}'],  
 languageOptions: {  
 ecmaVersion: 2020,  
 globals: globals.browser,  
 parserOptions: {  
 ecmaVersion: 'latest',  
 ecmaFeatures: { jsx: true },  
 sourceType: 'module',  
 },  
 },  
 settings: { react: { version: '18.3' } },  
 plugins: {  
 react,  
 'react-hooks': reactHooks,  
 'react-refresh': reactRefresh,  
 },  
 rules: {  
 ...js.configs.recommended.rules,  
 ...react.configs.recommended.rules,  
 ...react.configs['jsx-runtime'].rules,  
 ...reactHooks.configs.recommended.rules,  
 'react/jsx-no-target-blank': 'off',  
 'react-refresh/only-export-components': [  
 'warn',  
 { allowConstantExport: true },  
 ],  
 },  
 },  
]

## IntelliSprout/client/index.html

<!doctype html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <link rel="icon" type="image/svg+xml" href="/vite.svg" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Vite + React</title>  
 </head>  
 <body>  
 <div id="root"></div>  
 <script type="module" src="/src/main.jsx"></script>  
 </body>  
</html>

## IntelliSprout/client/postcss.config.js

export default {  
 plugins: {  
 tailwindcss: {},  
 autoprefixer: {},  
 },  
}

## IntelliSprout/client/src/App.css

## IntelliSprout/client/src/app/rootRedcuer.js

import { combineReducers } from "@reduxjs/toolkit";  
import authReducer from "../features/authSlice";   
import { authApi } from "@/features/api/authApi";  
import { courseApi } from "@/features/api/courseApi";  
import { purchaseApi } from "@/features/api/purchaseApi";  
import { courseProgressApi } from "@/features/api/courseProgressApi";  
  
const rootRedcuer = combineReducers({  
 [authApi.reducerPath]:authApi.reducer,  
 [courseApi.reducerPath]:courseApi.reducer,  
 [purchaseApi.reducerPath]:purchaseApi.reducer,  
 [courseProgressApi.reducerPath]:courseProgressApi.reducer,  
 auth:authReducer,   
});  
export default rootRedcuer;

## IntelliSprout/client/src/app/store.js

import {configureStore} from "@reduxjs/toolkit"   
import rootRedcuer from "./rootRedcuer";  
import { authApi } from "@/features/api/authApi";  
import { courseApi } from "@/features/api/courseApi";  
import { purchaseApi } from "@/features/api/purchaseApi";  
import { courseProgressApi } from "@/features/api/courseProgressApi";  
  
export const appStore = configureStore({  
 reducer: rootRedcuer,  
 middleware:(defaultMiddleware) => defaultMiddleware().concat(authApi.middleware, courseApi.middleware, purchaseApi.middleware, courseProgressApi.middleware)  
});  
  
const initializeApp = async () => {  
 await appStore.dispatch(authApi.endpoints.loadUser.initiate({},{forceRefetch:true}))  
}  
initializeApp();

## IntelliSprout/client/src/features/api/authApi.js

import {createApi, fetchBaseQuery} from "@reduxjs/toolkit/query/react";  
import { userLoggedIn, userLoggedOut } from "../authSlice";  
  
const USER\_API = "http://localhost:8080/api/v1/user/"  
  
export const authApi = createApi({  
 reducerPath:"authApi",  
 baseQuery:fetchBaseQuery({  
 baseUrl:USER\_API,  
 credentials:'include'  
 }),  
 endpoints: (builder) => ({  
 registerUser: builder.mutation({  
 query: (inputData) => ({  
 url:"register",  
 method:"POST",  
 body:inputData  
 })  
 }),  
 loginUser: builder.mutation({  
 query: (inputData) => ({  
 url:"login",  
 method:"POST",  
 body:inputData  
 }),  
 async onQueryStarted(\_, {queryFulfilled, dispatch}) {  
 try {  
 const result = await queryFulfilled;  
 dispatch(userLoggedIn({user:result.data.user}));  
 } catch (error) {  
 console.log(error);  
 }  
 }  
 }),  
 logoutUser: builder.mutation({  
 query: () => ({  
 url:"logout",  
 method:"GET"  
 }),  
 async onQueryStarted(\_, {queryFulfilled, dispatch}) {  
 try {   
 dispatch(userLoggedOut());  
 } catch (error) {  
 console.log(error);  
 }  
 }  
 }),  
 loadUser: builder.query({  
 query: () => ({  
 url:"profile",  
 method:"GET"  
 }),  
 async onQueryStarted(\_, {queryFulfilled, dispatch}) {  
 try {  
 const result = await queryFulfilled;  
 dispatch(userLoggedIn({user:result.data.user}));  
 } catch (error) {  
 console.log(error);  
 }  
 }  
 }),  
 updateUser: builder.mutation({  
 query: (formData) => ({  
 url:"profile/update",  
 method:"PUT",  
 body:formData,  
 credentials:"include"  
 })  
 })  
 })  
});  
export const {  
 useRegisterUserMutation,  
 useLoginUserMutation,  
 useLogoutUserMutation,  
 useLoadUserQuery,  
 useUpdateUserMutation  
} = authApi;

## IntelliSprout/client/src/features/api/courseApi.js

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";  
  
const COURSE\_API = "http://localhost:8080/api/v1/course";  
  
export const courseApi = createApi({  
 reducerPath: "courseApi",  
 tagTypes: ["Refetch\_Creator\_Course", "Refetch\_Lecture"],  
 baseQuery: fetchBaseQuery({  
 baseUrl: COURSE\_API,  
 credentials: "include",  
 }),  
 endpoints: (builder) => ({  
 createCourse: builder.mutation({  
 query: ({ courseTitle, category }) => ({  
 url: "",  
 method: "POST",  
 body: { courseTitle, category },  
 }),  
 invalidatesTags: ["Refetch\_Creator\_Course"],  
 }),  
 getSearchCourse:builder.query({  
 query: ({searchQuery, categories, sortByPrice}) => {  
 // Build qiery string  
 let queryString = `/search?query=${encodeURIComponent(searchQuery)}`  
  
 // append cateogry   
 if(categories && categories.length > 0) {  
 const categoriesString = categories.map(encodeURIComponent).join(",");  
 queryString += `&categories=${categoriesString}`;   
 }  
  
 // Append sortByPrice is available  
 if(sortByPrice){  
 queryString += `&sortByPrice=${encodeURIComponent(sortByPrice)}`;   
 }  
  
 return {  
 url:queryString,  
 method:"GET",   
 }  
 }  
 }),  
 getPublishedCourse: builder.query({  
 query: () => ({  
 url: "/published-courses",  
 method: "GET",  
 }),  
 }),  
 getCreatorCourse: builder.query({  
 query: () => ({  
 url: "",  
 method: "GET",  
 }),  
 providesTags: ["Refetch\_Creator\_Course"],  
 }),  
 editCourse: builder.mutation({  
 query: ({ formData, courseId }) => ({  
 url: `/${courseId}`,  
 method: "PUT",  
 body: formData,  
 }),  
 invalidatesTags: ["Refetch\_Creator\_Course"],  
 }),  
 getCourseById: builder.query({  
 query: (courseId) => ({  
 url: `/${courseId}`,  
 method: "GET",  
 }),  
 }),  
 createLecture: builder.mutation({  
 query: ({ lectureTitle, courseId }) => ({  
 url: `/${courseId}/lecture`,  
 method: "POST",  
 body: { lectureTitle },  
 }),  
 }),  
 getCourseLecture: builder.query({  
 query: (courseId) => ({  
 url: `/${courseId}/lecture`,  
 method: "GET",  
 }),  
 providesTags: ["Refetch\_Lecture"],  
 }),  
 editLecture: builder.mutation({  
 query: ({  
 lectureTitle,  
 videoInfo,  
 isPreviewFree,  
 courseId,  
 lectureId,  
 }) => ({  
 url: `/${courseId}/lecture/${lectureId}`,  
 method: "POST",  
 body: { lectureTitle, videoInfo, isPreviewFree },  
 }),  
 }),  
 removeLecture: builder.mutation({  
 query: (lectureId) => ({  
 url: `/lecture/${lectureId}`,  
 method: "DELETE",  
 }),  
 invalidatesTags: ["Refetch\_Lecture"],  
 }),  
 getLectureById: builder.query({  
 query: (lectureId) => ({  
 url: `/lecture/${lectureId}`,  
 method: "GET",  
 }),  
 }),  
 publishCourse: builder.mutation({  
 query: ({ courseId, query }) => ({  
 url: `/${courseId}?publish=${query}`,  
 method: "PATCH",  
 }),  
 }),  
 }),  
});  
export const {  
 useCreateCourseMutation,  
 useGetSearchCourseQuery,  
 useGetPublishedCourseQuery,  
 useGetCreatorCourseQuery,  
 useEditCourseMutation,  
 useGetCourseByIdQuery,  
 useCreateLectureMutation,  
 useGetCourseLectureQuery,  
 useEditLectureMutation,  
 useRemoveLectureMutation,  
 useGetLectureByIdQuery,  
 usePublishCourseMutation,  
} = courseApi;

## IntelliSprout/client/src/features/api/courseProgressApi.js

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";  
  
const COURSE\_PROGRESS\_API = "http://localhost:8080/api/v1/progress";  
  
export const courseProgressApi = createApi({  
 reducerPath: "courseProgressApi",  
 baseQuery: fetchBaseQuery({  
 baseUrl: COURSE\_PROGRESS\_API,  
 credentials: "include",  
 }),  
 endpoints: (builder) => ({  
 getCourseProgress: builder.query({  
 query: (courseId) => ({  
 url: `/${courseId}`,  
 method: "GET",  
 }),  
 }),  
 updateLectureProgress: builder.mutation({  
 query: ({ courseId, lectureId }) => ({  
 url: `/${courseId}/lecture/${lectureId}/view`,  
 method:"POST"  
 }),  
 }),  
  
 completeCourse: builder.mutation({  
 query:(courseId) => ({  
 url:`/${courseId}/complete`,  
 method:"POST"  
 })  
 }),  
 inCompleteCourse: builder.mutation({  
 query:(courseId) => ({  
 url:`/${courseId}/incomplete`,  
 method:"POST"  
 })  
 }),  
   
 }),  
});  
export const {  
useGetCourseProgressQuery,  
useUpdateLectureProgressMutation,  
useCompleteCourseMutation,  
useInCompleteCourseMutation  
} = courseProgressApi;

## IntelliSprout/client/src/features/api/purchaseApi.js

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";  
  
const COURSE\_PURCHASE\_API = "http://localhost:8080/api/v1/purchase";  
  
export const purchaseApi = createApi({  
 reducerPath: "purchaseApi",  
 baseQuery: fetchBaseQuery({  
 baseUrl: COURSE\_PURCHASE\_API,  
 credentials: "include",  
 }),  
 endpoints: (builder) => ({  
 createCheckoutSession: builder.mutation({  
 query: (courseId) => ({  
 url: "/checkout/create-checkout-session",  
 method: "POST",  
 body: { courseId },  
 }),  
 }),  
 getCourseDetailWithStatus: builder.query({  
 query: (courseId) => ({  
 url: `/course/${courseId}/detail-with-status`,  
 method: "GET",  
 }),  
 }),  
 getPurchasedCourses: builder.query({  
 query: () => ({  
 url: `/`,  
 method: "GET",  
 }),  
 }),  
 }),  
});  
  
export const {  
 useCreateCheckoutSessionMutation,  
 useGetCourseDetailWithStatusQuery,  
 useGetPurchasedCoursesQuery,  
} = purchaseApi;

## IntelliSprout/client/src/features/authSlice.js

import { createSlice } from "@reduxjs/toolkit";  
  
const initialState = {  
 user: null,  
 isAuthenticated: false,  
};   
  
const authSlice = createSlice({  
 name: "authSlice",  
 initialState,  
 reducers: {  
 userLoggedIn: (state, action) => {  
 state.user = action.payload.user;  
 state.isAuthenticated = true;  
 },  
 userLoggedOut:(state) => {  
 state.user = null;  
 state.isAuthenticated = false;  
 }  
 },  
});  
  
export const {userLoggedIn, userLoggedOut} = authSlice.actions;  
export default authSlice.reducer;

## IntelliSprout/client/src/features/courseSlice.js

## IntelliSprout/client/src/index.css

@tailwind base;  
@tailwind components;  
@tailwind utilities;  
@layer base {  
 :root {  
 --background: 0 0% 100%;  
 --foreground: 240 10% 3.9%;  
 --card: 0 0% 100%;  
 --card-foreground: 240 10% 3.9%;  
 --popover: 0 0% 100%;  
 --popover-foreground: 240 10% 3.9%;  
 --primary: 142.1 76.2% 36.3%;  
 --primary-foreground: 355.7 100% 97.3%;  
 --secondary: 240 4.8% 95.9%;  
 --secondary-foreground: 240 5.9% 10%;  
 --muted: 240 4.8% 95.9%;  
 --muted-foreground: 240 3.8% 46.1%;  
 --accent: 240 4.8% 95.9%;  
 --accent-foreground: 240 5.9% 10%;  
 --destructive: 0 84.2% 60.2%;  
 --destructive-foreground: 0 0% 98%;  
 --border: 240 5.9% 90%;  
 --input: 240 5.9% 90%;  
 --ring: 142.1 76.2% 36.3%;  
 --radius: 0.3rem;  
 --chart-1: 12 76% 61%;  
 --chart-2: 173 58% 39%;  
 --chart-3: 197 37% 24%;  
 --chart-4: 43 74% 66%;  
 --chart-5: 27 87% 67%;  
 }  
  
 .dark {  
 --background: 240 30.3% 7.1%;  
 --foreground: 0 0% 95%;  
 --card: 24 9.8% 10%;  
 --card-foreground: 0 0% 95%;  
 --popover: 0 0% 9%;  
 --popover-foreground: 0 0% 95%;  
 --primary: 142.1 70.6% 45.3%;  
 --primary-foreground: 144.9 80.4% 10%;  
 --secondary: 240 3.7% 15.9%;  
 --secondary-foreground: 0 0% 98%;  
 --muted: 0 0% 15%;  
 --muted-foreground: 240 5% 64.9%;  
 --accent: 12 6.5% 15.1%;  
 --accent-foreground: 0 0% 98%;  
 --destructive: 0 62.8% 30.6%;  
 --destructive-foreground: 0 85.7% 97.3%;  
 --border: 240 3.7% 15.9%;  
 --input: 240 3.7% 15.9%;  
 --ring: 142.4 71.8% 29.2%;  
 --chart-1: 220 70% 50%;  
 --chart-2: 160 60% 45%;  
 --chart-3: 30 80% 55%;  
 --chart-4: 280 65% 60%;  
 --chart-5: 340 75% 55%;  
 }  
}  
@layer base {  
 \* {  
 @apply border-border;  
 }  
 body {  
 @apply bg-background text-foreground;  
 }  
}  
  
::-webkit-scrollbar{  
 width: 0px;  
 height: 0px;  
}

## IntelliSprout/client/src/lib/utils.js

import { clsx } from "clsx";  
import { twMerge } from "tailwind-merge"  
  
export function cn(...inputs) {  
 return twMerge(clsx(inputs));  
}

## IntelliSprout/client/tailwind.config.js

/\*\* @type {import('tailwindcss').Config} \*/  
export default {  
 darkMode: ["class"],  
 content: ["./index.html", "./src/\*\*/\*.{ts,tsx,js,jsx}"],  
 theme: {  
 extend: {  
 borderRadius: {  
 lg: 'var(--radius)',  
 md: 'calc(var(--radius) - 2px)',  
 sm: 'calc(var(--radius) - 4px)'  
 },  
 colors: {  
 background: 'hsl(var(--background))',  
 foreground: 'hsl(var(--foreground))',  
 card: {  
 DEFAULT: 'hsl(var(--card))',  
 foreground: 'hsl(var(--card-foreground))'  
 },  
 popover: {  
 DEFAULT: 'hsl(var(--popover))',  
 foreground: 'hsl(var(--popover-foreground))'  
 },  
 primary: {  
 DEFAULT: 'hsl(var(--primary))',  
 foreground: 'hsl(var(--primary-foreground))'  
 },  
 secondary: {  
 DEFAULT: 'hsl(var(--secondary))',  
 foreground: 'hsl(var(--secondary-foreground))'  
 },  
 muted: {  
 DEFAULT: 'hsl(var(--muted))',  
 foreground: 'hsl(var(--muted-foreground))'  
 },  
 accent: {  
 DEFAULT: 'hsl(var(--accent))',  
 foreground: 'hsl(var(--accent-foreground))'  
 },  
 destructive: {  
 DEFAULT: 'hsl(var(--destructive))',  
 foreground: 'hsl(var(--destructive-foreground))'  
 },  
 border: 'hsl(var(--border))',  
 input: 'hsl(var(--input))',  
 ring: 'hsl(var(--ring))',  
 chart: {  
 '1': 'hsl(var(--chart-1))',  
 '2': 'hsl(var(--chart-2))',  
 '3': 'hsl(var(--chart-3))',  
 '4': 'hsl(var(--chart-4))',  
 '5': 'hsl(var(--chart-5))'  
 }  
 }  
 }  
 },  
 plugins: [require("tailwindcss-animate")],  
}

## IntelliSprout/client/vite.config.js

import path from "path"  
import react from "@vitejs/plugin-react"  
import { defineConfig } from "vite"  
   
export default defineConfig({  
 plugins: [react()],  
 resolve: {  
 alias: {  
 "@": path.resolve(\_\_dirname, "./src"),  
 },  
 },  
})

## IntelliSprout/server/controllers/course.controller.js

import { Course } from "../models/course.model.js";  
import { Lecture } from "../models/lecture.model.js";  
import {deleteMediaFromCloudinary, deleteVideoFromCloudinary, uploadMedia} from "../utils/cloudinary.js";  
  
export const createCourse = async (req,res) => {  
 try {  
 const {courseTitle, category} = req.body;  
 if(!courseTitle || !category) {  
 return res.status(400).json({  
 message:"Course title and category is required."  
 })  
 }  
  
 const course = await Course.create({  
 courseTitle,  
 category,  
 creator:req.id  
 });  
  
 return res.status(201).json({  
 course,  
 message:"Course created."  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to create course"  
 })  
 }  
}  
  
export const searchCourse = async (req,res) => {  
 try {  
 const {query = "", categories = [], sortByPrice =""} = req.query;  
 console.log(categories);  
   
 // create search query  
 const searchCriteria = {  
 isPublished:true,  
 $or:[  
 {courseTitle: {$regex:query, $options:"i"}},  
 {subTitle: {$regex:query, $options:"i"}},  
 {category: {$regex:query, $options:"i"}},  
 ]  
 }  
  
 // if categories selected  
 if(categories.length > 0) {  
 searchCriteria.category = {$in: categories};  
 }  
  
 // define sorting order  
 const sortOptions = {};  
 if(sortByPrice === "low"){  
 sortOptions.coursePrice = 1;//sort by price in ascending  
 }else if(sortByPrice === "high"){  
 sortOptions.coursePrice = -1; // descending  
 }  
  
 let courses = await Course.find(searchCriteria).populate({path:"creator", select:"name photoUrl"}).sort(sortOptions);  
  
 return res.status(200).json({  
 success:true,  
 courses: courses || []  
 });  
  
 } catch (error) {  
 console.log(error);  
   
 }  
}  
  
export const getPublishedCourse = async (\_,res) => {  
 try {  
 const courses = await Course.find({isPublished:true}).populate({path:"creator", select:"name photoUrl"});  
 if(!courses){  
 return res.status(404).json({  
 message:"Course not found"  
 })  
 }  
 return res.status(200).json({  
 courses,  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to get published courses"  
 })  
 }  
}  
export const getCreatorCourses = async (req,res) => {  
 try {  
 const userId = req.id;  
 const courses = await Course.find({creator:userId});  
 if(!courses){  
 return res.status(404).json({  
 courses:[],  
 message:"Course not found"  
 })  
 };  
 return res.status(200).json({  
 courses,  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to create course"  
 })  
 }  
}  
export const editCourse = async (req,res) => {  
 try {  
 const courseId = req.params.courseId;  
 const {courseTitle, subTitle, description, category, courseLevel, coursePrice} = req.body;  
 const thumbnail = req.file;  
  
 let course = await Course.findById(courseId);  
 if(!course){  
 return res.status(404).json({  
 message:"Course not found!"  
 })  
 }  
 let courseThumbnail;  
 if(thumbnail){  
 if(course.courseThumbnail){  
 const publicId = course.courseThumbnail.split("/").pop().split(".")[0];  
 await deleteMediaFromCloudinary(publicId); // delete old image  
 }  
 // upload a thumbnail on clourdinary  
 courseThumbnail = await uploadMedia(thumbnail.path);  
 }  
  
   
 const updateData = {courseTitle, subTitle, description, category, courseLevel, coursePrice, courseThumbnail:courseThumbnail?.secure\_url};  
  
 course = await Course.findByIdAndUpdate(courseId, updateData, {new:true});  
  
 return res.status(200).json({  
 course,  
 message:"Course updated successfully."  
 })  
  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to create course"  
 })  
 }  
}  
export const getCourseById = async (req,res) => {  
 try {  
 const {courseId} = req.params;  
  
 const course = await Course.findById(courseId);  
  
 if(!course){  
 return res.status(404).json({  
 message:"Course not found!"  
 })  
 }  
 return res.status(200).json({  
 course  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to get course by id"  
 })  
 }  
}  
  
export const createLecture = async (req,res) => {  
 try {  
 const {lectureTitle} = req.body;  
 const {courseId} = req.params;  
  
 if(!lectureTitle || !courseId){  
 return res.status(400).json({  
 message:"Lecture title is required"  
 })  
 };  
  
 // create lecture  
 const lecture = await Lecture.create({lectureTitle});  
  
 const course = await Course.findById(courseId);  
 if(course){  
 course.lectures.push(lecture.\_id);  
 await course.save();  
 }  
  
 return res.status(201).json({  
 lecture,  
 message:"Lecture created successfully."  
 });  
  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to create lecture"  
 })  
 }  
}  
export const getCourseLecture = async (req,res) => {  
 try {  
 const {courseId} = req.params;  
 const course = await Course.findById(courseId).populate("lectures");  
 if(!course){  
 return res.status(404).json({  
 message:"Course not found"  
 })  
 }  
 return res.status(200).json({  
 lectures: course.lectures  
 });  
  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to get lectures"  
 })  
 }  
}  
export const editLecture = async (req,res) => {  
 try {  
 const {lectureTitle, videoInfo, isPreviewFree} = req.body;  
   
 const {courseId, lectureId} = req.params;  
 const lecture = await Lecture.findById(lectureId);  
 if(!lecture){  
 return res.status(404).json({  
 message:"Lecture not found!"  
 })  
 }  
  
 // update lecture  
 if(lectureTitle) lecture.lectureTitle = lectureTitle;  
 if(videoInfo?.videoUrl) lecture.videoUrl = videoInfo.videoUrl;  
 if(videoInfo?.publicId) lecture.publicId = videoInfo.publicId;  
 lecture.isPreviewFree = isPreviewFree;  
  
 await lecture.save();  
  
 // Ensure the course still has the lecture id if it was not aleardy added;  
 const course = await Course.findById(courseId);  
 if(course && !course.lectures.includes(lecture.\_id)){  
 course.lectures.push(lecture.\_id);  
 await course.save();  
 };  
 return res.status(200).json({  
 lecture,  
 message:"Lecture updated successfully."  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to edit lectures"  
 })  
 }  
}  
export const removeLecture = async (req,res) => {  
 try {  
 const {lectureId} = req.params;  
 const lecture = await Lecture.findByIdAndDelete(lectureId);  
 if(!lecture){  
 return res.status(404).json({  
 message:"Lecture not found!"  
 });  
 }  
 // delete the lecture from couldinary as well  
 if(lecture.publicId){  
 await deleteVideoFromCloudinary(lecture.publicId);  
 }  
  
 // Remove the lecture reference from the associated course  
 await Course.updateOne(  
 {lectures:lectureId}, // find the course that contains the lecture  
 {$pull:{lectures:lectureId}} // Remove the lectures id from the lectures array  
 );  
  
 return res.status(200).json({  
 message:"Lecture removed successfully."  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to remove lecture"  
 })  
 }  
}  
export const getLectureById = async (req,res) => {  
 try {  
 const {lectureId} = req.params;  
 const lecture = await Lecture.findById(lectureId);  
 if(!lecture){  
 return res.status(404).json({  
 message:"Lecture not found!"  
 });  
 }  
 return res.status(200).json({  
 lecture  
 });  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to get lecture by id"  
 })  
 }  
}  
  
  
// publich unpublish course logic  
  
export const togglePublishCourse = async (req,res) => {  
 try {  
 const {courseId} = req.params;  
 const {publish} = req.query; // true, false  
 const course = await Course.findById(courseId);  
 if(!course){  
 return res.status(404).json({  
 message:"Course not found!"  
 });  
 }  
 // publish status based on the query paramter  
 course.isPublished = publish === "true";  
 await course.save();  
  
 const statusMessage = course.isPublished ? "Published" : "Unpublished";  
 return res.status(200).json({  
 message:`Course is ${statusMessage}`  
 });  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 message:"Failed to update status"  
 })  
 }  
}

## IntelliSprout/server/controllers/courseProgress.controller.js

import { CourseProgress } from "../models/courseProgress.js";  
import { Course } from "../models/course.model.js";  
  
export const getCourseProgress = async (req, res) => {  
 try {  
 const { courseId } = req.params;  
 const userId = req.id;  
  
 // step-1 fetch the user course progress  
 let courseProgress = await CourseProgress.findOne({  
 courseId,  
 userId,  
 }).populate("courseId");  
  
 const courseDetails = await Course.findById(courseId).populate("lectures");  
  
 if (!courseDetails) {  
 return res.status(404).json({  
 message: "Course not found",  
 });  
 }  
  
 // Step-2 If no progress found, return course details with an empty progress  
 if (!courseProgress) {  
 return res.status(200).json({  
 data: {  
 courseDetails,  
 progress: [],  
 completed: false,  
 },  
 });  
 }  
  
 // Step-3 Return the user's course progress alog with course details  
 return res.status(200).json({  
 data: {  
 courseDetails,  
 progress: courseProgress.lectureProgress,  
 completed: courseProgress.completed,  
 },  
 });  
 } catch (error) {  
 console.log(error);  
 }  
};  
  
export const updateLectureProgress = async (req, res) => {  
 try {  
 const { courseId, lectureId } = req.params;  
 const userId = req.id;  
  
 // fetch or create course progress  
 let courseProgress = await CourseProgress.findOne({ courseId, userId });  
  
 if (!courseProgress) {  
 // If no progress exist, create a new record  
 courseProgress = new CourseProgress({  
 userId,  
 courseId,  
 completed: false,  
 lectureProgress: [],  
 });  
 }  
  
 // find the lecture progress in the course progress  
 const lectureIndex = courseProgress.lectureProgress.findIndex(  
 (lecture) => lecture.lectureId === lectureId  
 );  
  
 if (lectureIndex !== -1) {  
 // if lecture already exist, update its status  
 courseProgress.lectureProgress[lectureIndex].viewed = true;  
 } else {  
 // Add new lecture progress  
 courseProgress.lectureProgress.push({  
 lectureId,  
 viewed: true,  
 });  
 }  
  
 // if all lecture is complete  
 const lectureProgressLength = courseProgress.lectureProgress.filter(  
 (lectureProg) => lectureProg.viewed  
 ).length;  
  
 const course = await Course.findById(courseId);  
  
 if (course.lectures.length === lectureProgressLength)  
 courseProgress.completed = true;  
  
 await courseProgress.save();  
  
 return res.status(200).json({  
 message: "Lecture progress updated successfully.",  
 });  
 } catch (error) {  
 console.log(error);  
 }  
};  
  
export const markAsCompleted = async (req, res) => {  
 try {  
 const { courseId } = req.params;  
 const userId = req.id;  
  
 const courseProgress = await CourseProgress.findOne({ courseId, userId });  
 if (!courseProgress)  
 return res.status(404).json({ message: "Course progress not found" });  
  
 courseProgress.lectureProgress.map(  
 (lectureProgress) => (lectureProgress.viewed = true)  
 );  
 courseProgress.completed = true;  
 await courseProgress.save();  
 return res.status(200).json({ message: "Course marked as completed." });  
 } catch (error) {  
 console.log(error);  
 }  
};  
  
export const markAsInCompleted = async (req, res) => {  
 try {  
 const { courseId } = req.params;  
 const userId = req.id;  
   
 const courseProgress = await CourseProgress.findOne({ courseId, userId });  
 if (!courseProgress)  
 return res.status(404).json({ message: "Course progress not found" });  
   
 courseProgress.lectureProgress.map(  
 (lectureProgress) => (lectureProgress.viewed = false)  
 );  
 courseProgress.completed = false;  
 await courseProgress.save();  
 return res.status(200).json({ message: "Course marked as incompleted." });  
 } catch (error) {  
 console.log(error);  
 }  
 };

## IntelliSprout/server/controllers/coursePurchase.controller.js

import Stripe from "stripe";  
import { Course } from "../models/course.model.js";  
import { CoursePurchase } from "../models/coursePurchase.model.js";  
import { Lecture } from "../models/lecture.model.js";  
import { User } from "../models/user.model.js";  
  
const stripe = new Stripe(process.env.STRIPE\_SECRET\_KEY || " ");  
  
export const createCheckoutSession = async (req, res) => {  
 try {  
 const userId = req.id;  
 const { courseId } = req.body;  
  
 const course = await Course.findById(courseId);  
 if (!course) return res.status(404).json({ message: "Course not found!" });  
  
 // Create a new course purchase record  
 const newPurchase = new CoursePurchase({  
 courseId,  
 userId,  
 amount: course.coursePrice,  
 status: "pending",  
 });  
  
 // Create a Stripe checkout session  
 const session = await stripe.checkout.sessions.create({  
 payment\_method\_types: ["card"],  
 line\_items: [  
 {  
 price\_data: {  
 currency: "inr",  
 product\_data: {  
 name: course.courseTitle,  
 images: [course.courseThumbnail],  
 },  
 unit\_amount: course.coursePrice \* 100, // Amount in paise (lowest denomination)  
 },  
 quantity: 1,  
 },  
 ],  
 mode: "payment",  
 success\_url: `http://localhost:5173/course-progress/${courseId}`, // once payment successful redirect to course progress page  
 cancel\_url: `http://localhost:5173/course-detail/${courseId}`,  
 metadata: {  
 courseId: courseId,  
 userId: userId,  
 },  
 shipping\_address\_collection: {  
 allowed\_countries: ["IN"], // Optionally restrict allowed countries  
 },  
 });  
  
 if (!session.url) {  
 return res  
 .status(400)  
 .json({ success: false, message: "Error while creating session" });  
 }  
  
 // Save the purchase record  
 newPurchase.paymentId = session.id;  
 await newPurchase.save();  
  
 return res.status(200).json({  
 success: true,  
 url: session.url, // Return the Stripe checkout URL  
 });  
 } catch (error) {  
 console.log(error);  
 }  
};  
  
export const stripeWebhook = async (req, res) => {  
 let event;  
 console.log("here")  
  
 try {  
 const secret = process.env.WEBHOOK\_ENDPOINT\_SECRET;  
  
 // Get the signature sent by Stripe  
 const signature = req.headers['stripe-signature'];  
  
 // Verify the event using the actual signature  
 event = stripe.webhooks.constructEvent(  
 req.body, // Use the raw body, not stringified  
 signature,  
 secret  
 );  
 } catch (error) {  
 console.error("Webhook error:", error.message);  
 return res.status(400).send(`Webhook error: ${error.message}`);  
 }  
  
 // Handle the checkout session completed event  
 if (event.type === "checkout.session.completed") {  
 console.log("Checkout session complete is called");  
  
 try {  
 const session = event.data.object;  
  
 // Find the purchase and populate the course details  
 const purchase = await CoursePurchase.findOne({  
 paymentId: session.id,  
 }).populate("courseId"); // Ensure courseId is populated properly  
  
 if (!purchase) {  
 console.error("Purchase not found for paymentId:", session.id);  
 return res.status(404).json({ message: "Purchase not found" });  
 }  
  
 console.log("Purchase found:", purchase);  
  
 // Update the purchase amount and status  
 if (session.amount\_total) {  
 purchase.amount = session.amount\_total / 100; // Convert cents to dollars  
 }  
   
 // Set the status to completed  
 purchase.status = "completed";  
   
 console.log("Updating purchase status to completed");  
  
 // Make all lectures visible by setting `isPreviewFree` to true  
 if (purchase.courseId && purchase.courseId.lectures.length > 0) {  
 await Lecture.updateMany(  
 { \_id: { $in: purchase.courseId.lectures } },  
 { $set: { isPreviewFree: true } }  
 );  
 console.log("Lectures visibility updated to preview free");  
 }  
  
 // Save the updated purchase status  
 const savedPurchase = await purchase.save();  
  
 if (!savedPurchase) {  
 console.error("Error saving updated purchase");  
 return res.status(500).json({ message: "Failed to update purchase" });  
 }  
  
 console.log("Purchase updated successfully:", savedPurchase);  
  
 // Add course to user's enrolledCourses (Make sure the userId is correct)  
 const userUpdate = await User.findByIdAndUpdate(  
 purchase.userId,  
 { $addToSet: { enrolledCourses: purchase.courseId.\_id } }, // Add course ID to enrolledCourses  
 { new: true }  
 );  
  
 if (!userUpdate) {  
 console.error(`User with ID ${purchase.userId} not found`);  
 return res.status(404).json({ message: "User not found" });  
 }  
  
 console.log(`User with ID ${purchase.userId} enrolled in course ${purchase.courseId.\_id}`);  
  
 // Add user to course's enrolledStudents (Make sure courseId is correct)  
 const courseUpdate = await Course.findByIdAndUpdate(  
 purchase.courseId.\_id,  
 { $addToSet: { enrolledStudents: purchase.userId } }, // Add user ID to enrolledStudents  
 { new: true }  
 );  
  
 if (!courseUpdate) {  
 console.error(`Course with ID ${purchase.courseId.\_id} not found`);  
 return res.status(404).json({ message: "Course not found" });  
 }  
  
 console.log(`Course with ID ${purchase.courseId.\_id} updated with user ${purchase.userId}`);  
   
 } catch (error) {  
 console.error("Error handling event:", error);  
 return res.status(500).json({ message: "Internal Server Error" });  
 }  
 }  
  
 res.status(200).send();  
};  
  
  
export const getCourseDetailWithPurchaseStatus = async (req, res) => {  
 try {  
 console.log("here");  
   
 const { courseId } = req.params;  
 const userId = req.id;  
  
 const course = await Course.findById(courseId)  
 .populate({ path: "creator" })  
 .populate({ path: "lectures" });  
  
 const purchased = await CoursePurchase.findOne({ userId, courseId });  
  
 if (!course) {  
 return res.status(404).json({ message: "course not found!" });  
 }  
  
 return res.status(200).json({  
 course,  
 purchased: !!purchased, // true if purchased, false otherwise  
 });  
 } catch (error) {  
 console.log(error);  
 }  
};  
  
export const getAllPurchasedCourse = async (\_, res) => {  
 try {  
 console.log("here2");  
   
 const purchasedCourse = await CoursePurchase.find({  
 status: "completed",  
 }).populate("courseId");  
 if (!purchasedCourse) {  
 return res.status(404).json({  
 purchasedCourse: [],  
 });  
 }  
 return res.status(200).json({  
 purchasedCourse,  
 });  
 } catch (error) {  
 console.log(error);  
 }  
};

## IntelliSprout/server/controllers/user.controller.js

import {User} from "../models/user.model.js";  
import bcrypt from "bcryptjs";  
import { generateToken } from "../utils/generateToken.js";  
import { deleteMediaFromCloudinary, uploadMedia } from "../utils/cloudinary.js";  
  
export const register = async (req,res) => {  
 try {  
   
 const {name, email, password} = req.body; // 214  
 if(!name || !email || !password){  
 return res.status(400).json({  
 success:false,  
 message:"All fields are required."  
 })  
 }  
 const user = await User.findOne({email});  
 if(user){  
 return res.status(400).json({  
 success:false,  
 message:"User already exist with this email."  
 })  
 }  
 const hashedPassword = await bcrypt.hash(password, 10);  
 await User.create({  
 name,  
 email,  
 password:hashedPassword  
 });  
 return res.status(201).json({  
 success:true,  
 message:"Account created successfully."  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 success:false,  
 message:"Failed to register"  
 })  
 }  
}  
export const login = async (req,res) => {  
 try {  
 const {email, password} = req.body;  
 if(!email || !password){  
 return res.status(400).json({  
 success:false,  
 message:"All fields are required."  
 })  
 }  
 const user = await User.findOne({email});  
 if(!user){  
 return res.status(400).json({  
 success:false,  
 message:"Incorrect email or password"  
 })  
 }  
 const isPasswordMatch = await bcrypt.compare(password, user.password);  
 if(!isPasswordMatch){  
 return res.status(400).json({  
 success:false,  
 message:"Incorrect email or password"  
 });  
 }  
 generateToken(res, user, `Welcome back ${user.name}`);  
  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 success:false,  
 message:"Failed to login"  
 })  
 }  
}  
export const logout = async (\_,res) => {  
 try {  
 return res.status(200).cookie("token", "", {maxAge:0}).json({  
 message:"Logged out successfully.",  
 success:true  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 success:false,  
 message:"Failed to logout"  
 })   
 }  
}  
export const getUserProfile = async (req,res) => {  
 try {  
 const userId = req.id;  
 const user = await User.findById(userId).select("-password").populate("enrolledCourses");  
 if(!user){  
 return res.status(404).json({  
 message:"Profile not found",  
 success:false  
 })  
 }  
 return res.status(200).json({  
 success:true,  
 user  
 })  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 success:false,  
 message:"Failed to load user"  
 })  
 }  
}  
export const updateProfile = async (req,res) => {  
 try {  
 const userId = req.id;  
 const {name} = req.body;  
 const profilePhoto = req.file;  
  
 const user = await User.findById(userId);  
 if(!user){  
 return res.status(404).json({  
 message:"User not found",  
 success:false  
 })   
 }  
 // extract public id of the old image from the url is it exists;  
 if(user.photoUrl){  
 const publicId = user.photoUrl.split("/").pop().split(".")[0]; // extract public id  
 deleteMediaFromCloudinary(publicId);  
 }  
  
 // upload new photo  
 const cloudResponse = await uploadMedia(profilePhoto.path);  
 const photoUrl = cloudResponse.secure\_url;  
  
 const updatedData = {name, photoUrl};  
 const updatedUser = await User.findByIdAndUpdate(userId, updatedData, {new:true}).select("-password");  
  
 return res.status(200).json({  
 success:true,  
 user:updatedUser,  
 message:"Profile updated successfully."  
 })  
  
 } catch (error) {  
 console.log(error);  
 return res.status(500).json({  
 success:false,  
 message:"Failed to update profile"  
 })  
 }  
}

## IntelliSprout/server/database/db.js

import mongoose from "mongoose";  
  
const connectDB = async () => {  
 try {  
 await mongoose.connect(process.env.MONGO\_URI);  
 console.log('MongoDB Connected');  
 } catch (error) {  
 console.log("error occured", error);   
 }  
}  
export default connectDB;

## IntelliSprout/server/index.js

import express from "express";  
import dotenv from "dotenv";  
import cookieParser from "cookie-parser";  
import cors from "cors";  
import connectDB from "./database/db.js";  
import userRoute from "./routes/user.route.js";  
import courseRoute from "./routes/course.route.js";  
import mediaRoute from "./routes/media.route.js";  
import purchaseRoute from "./routes/purchaseCourse.route.js";  
import courseProgressRoute from "./routes/courseProgress.route.js";  
  
dotenv.config({});  
  
// call database connection here  
connectDB();  
const app = express();  
  
const PORT = process.env.PORT || 3000;  
  
// default middleware  
app.use(express.json());  
app.use(cookieParser());  
  
app.use(cors({  
 origin:"http://localhost:5173",  
 credentials:true  
}));  
   
// apis  
app.use("/api/v1/media", mediaRoute);  
app.use("/api/v1/user", userRoute);  
app.use("/api/v1/course", courseRoute);  
app.use("/api/v1/purchase", purchaseRoute);  
app.use("/api/v1/progress", courseProgressRoute);  
   
   
app.listen(PORT, () => {  
 console.log(`Server listen at port ${PORT}`);  
})

## IntelliSprout/server/middlewares/isAuthenticated.js

import jwt from "jsonwebtoken";  
  
const isAuthenticated = async (req, res, next) => {  
 try {  
 const token = req.cookies.token;  
 if (!token) {  
 return res.status(401).json({  
 message: "User not authenticated",  
 success: false,  
 });  
 }  
 const decode = await jwt.verify(token, process.env.SECRET\_KEY);  
 if (!decode) {  
 return res.status(401).json({  
 message: "Invalid token",  
 success: false,  
 });  
 }  
 req.id = decode.userId;  
 next();  
 } catch (error) {  
 console.log(error);  
 }  
};  
export default isAuthenticated;

## IntelliSprout/server/models/course.model.js

import mongoose from "mongoose"  
  
const courseSchema = new mongoose.Schema({  
 courseTitle:{  
 type:String,  
 required:true  
 },  
 subTitle: {type:String},   
 description:{ type:String},  
 category:{  
 type:String,  
 required:true  
 },  
 courseLevel:{  
 type:String,  
 enum:["Beginner", "Medium", "Advance"]  
 },  
 coursePrice:{  
 type:Number  
 },  
 courseThumbnail:{  
 type:String  
 },  
 enrolledStudents:[  
 {  
 type:mongoose.Schema.Types.ObjectId,  
 ref:'User'  
 }  
 ],  
 lectures:[  
 {  
 type:mongoose.Schema.Types.ObjectId,  
 ref:"Lecture"  
 }  
 ],  
 creator:{  
 type:mongoose.Schema.Types.ObjectId,  
 ref:'User'  
 },  
 isPublished:{  
 type:Boolean,  
 default:false  
 }  
  
}, {timestamps:true});  
  
export const Course = mongoose.model("Course", courseSchema);

## IntelliSprout/server/models/courseProgress.js

import mongoose from "mongoose"  
  
const lectureProgressSchema = new mongoose.Schema({  
 lectureId:{type:String},  
 viewed:{type:Boolean}  
});  
  
const courseProgressSchema = new mongoose.Schema({  
 userId:{type:String},  
 courseId:{type:String},  
 completed:{type:Boolean},  
 lectureProgress:[lectureProgressSchema]  
});  
  
export const CourseProgress = mongoose.model("CourseProgress", courseProgressSchema);

## IntelliSprout/server/models/coursePurchase.model.js

import mongoose from "mongoose";  
const coursePurchaseSchema = new mongoose.Schema({  
 courseId:{  
 type:mongoose.Schema.Types.ObjectId,  
 ref:'Course',  
 required:true  
 },  
 userId:{  
 type:mongoose.Schema.Types.ObjectId,  
 ref:'User',  
 required:true  
 },  
 amount:{  
 type:Number,  
 required:true  
 },  
 status:{  
 type:String,  
 enum:['pending', 'completed', 'failed'],  
 default:'pending'  
 },  
 paymentId:{  
 type:String,  
 required:true  
 }  
  
},{timestamps:true});  
export const CoursePurchase = mongoose.model('CoursePurchase', coursePurchaseSchema);

## IntelliSprout/server/models/lecture.model.js

import mongoose from "mongoose";  
  
const lectureSchema = new mongoose.Schema({  
 lectureTitle: {  
 type: String,  
 required: true,  
 },  
 videoUrl: { type: String },  
 publicId: { type: String },  
 isPreviewFree: { type: Boolean },  
},{timestamps:true});  
  
export const Lecture = mongoose.model("Lecture", lectureSchema);

## IntelliSprout/server/models/user.model.js

import mongoose from "mongoose";  
  
const userSchema = new mongoose.Schema({  
 name:{  
 type:String,  
 required:true  
 },  
 email:{  
 type:String,  
 required:true  
 },  
 password:{  
 type:String,  
 required:true  
 },  
 role:{  
 type:String,  
 enum:["instructor", "student"],  
 default:'student'  
 },  
 enrolledCourses:[  
 {  
 type:mongoose.Schema.Types.ObjectId,  
 ref:'Course'  
 }  
 ],  
 photoUrl:{  
 type:String,  
 default:""  
 }  
},{timestamps:true});  
  
export const User = mongoose.model("User", userSchema);

## IntelliSprout/server/routes/course.route.js

import express from "express";  
import isAuthenticated from "../middlewares/isAuthenticated.js";  
import { createCourse, createLecture, editCourse, editLecture, getCourseById, getCourseLecture, getCreatorCourses, getLectureById, getPublishedCourse, removeLecture, searchCourse, togglePublishCourse } from "../controllers/course.controller.js";  
import upload from "../utils/multer.js";  
const router = express.Router();  
  
router.route("/").post(isAuthenticated,createCourse);  
router.route("/search").get(isAuthenticated, searchCourse);  
router.route("/published-courses").get( getPublishedCourse);  
router.route("/").get(isAuthenticated,getCreatorCourses);  
router.route("/:courseId").put(isAuthenticated,upload.single("courseThumbnail"),editCourse);  
router.route("/:courseId").get(isAuthenticated, getCourseById);  
router.route("/:courseId/lecture").post(isAuthenticated, createLecture);  
router.route("/:courseId/lecture").get(isAuthenticated, getCourseLecture);  
router.route("/:courseId/lecture/:lectureId").post(isAuthenticated, editLecture);  
router.route("/lecture/:lectureId").delete(isAuthenticated, removeLecture);  
router.route("/lecture/:lectureId").get(isAuthenticated, getLectureById);  
router.route("/:courseId").patch(isAuthenticated, togglePublishCourse);  
  
  
export default router;

## IntelliSprout/server/routes/courseProgress.route.js

import express from "express"  
import isAuthenticated from "../middlewares/isAuthenticated.js";  
import { getCourseProgress, markAsCompleted, markAsInCompleted, updateLectureProgress } from "../controllers/courseProgress.controller.js";  
  
const router = express.Router()  
  
router.route("/:courseId").get(isAuthenticated, getCourseProgress);  
router.route("/:courseId/lecture/:lectureId/view").post(isAuthenticated, updateLectureProgress);  
router.route("/:courseId/complete").post(isAuthenticated, markAsCompleted);  
router.route("/:courseId/incomplete").post(isAuthenticated, markAsInCompleted);  
  
export default router;

## IntelliSprout/server/routes/media.route.js

import express from "express";  
import upload from "../utils/multer.js";  
import { uploadMedia } from "../utils/cloudinary.js";  
  
const router = express.Router();  
  
router.route("/upload-video").post(upload.single("file"), async(req,res) => {  
 try {  
 const result = await uploadMedia(req.file.path);  
 res.status(200).json({  
 success:true,  
 message:"File uploaded successfully.",  
 data:result  
 });  
 } catch (error) {  
 console.log(error);  
 res.status(500).json({message:"Error uploading file"})  
 }  
});  
export default router;

## IntelliSprout/server/routes/purchaseCourse.route.js

import express from "express";  
import isAuthenticated from "../middlewares/isAuthenticated.js";  
import { createCheckoutSession, getAllPurchasedCourse, getCourseDetailWithPurchaseStatus, stripeWebhook } from "../controllers/coursePurchase.controller.js";  
  
const router = express.Router();  
  
router.route("/checkout/create-checkout-session").post(isAuthenticated, createCheckoutSession);  
router.route("/webhook").post(express.raw({type:"application/json"}), stripeWebhook);  
router.route("/course/:courseId/detail-with-status").get(isAuthenticated,getCourseDetailWithPurchaseStatus);  
  
router.route("/").get(isAuthenticated,getAllPurchasedCourse);  
  
export default router;

## IntelliSprout/server/routes/user.route.js

import express from "express";  
import { getUserProfile, login, logout, register, updateProfile } from "../controllers/user.controller.js";  
import isAuthenticated from "../middlewares/isAuthenticated.js";  
import upload from "../utils/multer.js";  
  
const router = express.Router();  
  
router.route("/register").post(register);  
router.route("/login").post(login);  
router.route("/logout").get(logout);  
router.route("/profile").get(isAuthenticated, getUserProfile);  
router.route("/profile/update").put(isAuthenticated, upload.single("profilePhoto"), updateProfile);  
  
export default router;

## IntelliSprout/server/utils/cloudinary.js

import { v2 as cloudinary } from "cloudinary";  
import dotenv from "dotenv";  
dotenv.config({});  
  
cloudinary.config({  
 api\_key: process.env.API\_KEY,  
 api\_secret: process.env.API\_SECRET,  
 cloud\_name: process.env.CLOUD\_NAME,  
});  
  
export const uploadMedia = async (file) => {  
 try {  
 const uploadResponse = await cloudinary.uploader.upload(file, {  
 resource\_type: "auto",  
 });  
 return uploadResponse;  
 } catch (error) {  
 console.log(error);  
 }  
};  
export const deleteMediaFromCloudinary = async (publicId) => {  
 try {  
 await cloudinary.uploader.destroy(publicId);  
 } catch (error) {  
 console.log(error);  
 }  
};  
  
export const deleteVideoFromCloudinary = async (publicId) => {  
 try {  
 await cloudinary.uploader.destroy(publicId,{resource\_type:"video"});  
 } catch (error) {  
 console.log(error);  
   
 }  
}

## IntelliSprout/server/utils/generateToken.js

import jwt from "jsonwebtoken";  
  
export const generateToken = (res, user, message) => {  
 const token = jwt.sign({ userId: user.\_id }, process.env.SECRET\_KEY, {  
 expiresIn: "1d",  
 });  
  
 return res  
 .status(200)  
 .cookie("token", token, {  
 httpOnly: true,  
 sameSite: "strict",  
 maxAge: 24 \* 60 \* 60 \* 1000, // 1 day  
 }).json({  
 success:true,  
 message,  
 user  
 });  
};

## IntelliSprout/server/utils/multer.js

import multer from "multer";  
  
const upload = multer({dest:"uploads/"});  
export default upload