AI Short Video Generator with NextJS & TypeScript

Prepared by Ryan Dhungel

- setup nextjs project with shadcnui
- use google font
- navigation
- logo and default dark
- clerk auth
- clerk provider, middleware, signin and signout
- navigation links
- create video context
- google generative ai
- video generate options to user
- context with story options, styles and event handlers
- story options
- styles options
- submit create video
- Modal
- return static resposne to avoid calling AI
- ai image generation and upload server action
- generating images from script
- code organization and mock functions
- text to speech
- generate captions
- split create video page
- remotion video setup
- using images to create video
- adding audio
- display captions
- centered captions
- smooth transition of images
- images zoom in effect
- moving effects functions
- switch away from mock functions
- database setup
- save to db
- display videos in dashboard
- create card on dashboard
- credit page and component
- paypal context for credits
- loader component
- buy credits page with paypal

- credit model
- credits on database server action
- get credits server action
- display credits
- credits in context
- display user credits
- credits on user signup
- spend credit on each video generation
- buy credits update client
- landing page with V0
- hero section with animated titles and background image
- parallax image
- features section
- how it works section with image icons
- pricing cards
- reviews section
- faq section with accordians
- call to action section
- footer
- deploy to vercel

setup nextjs project with shadcnui

follow the docs

use google font

```
import type { Metadata } from "next";
import localFont from "next/font/local";
import { Audiowide } from "next/font/google";
import "./globals.css";

// Importing Audiowide font from Google Fonts
const audiowide = Audiowide({
  weight: "400",
  subsets: ["latin"],
  variable: "--font-audiowide",
});

export const metadata: Metadata = {
  title: "Create Next App",
  description: "Generated by create next app",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode,
}>) {
```

```
  return (
    <html lang="en">
      <body className={`${audiowide.variable} antialiased`}>{children}
</body>
    </html>
  );
}

// update globals.css
body {
  font-family: var(--font-audiowide), sans-serif; /* Fallback to sans-
serif */
}
```

navigation

```
// components/nav/top-nav
import {
  Menubar,
  MenubarContent,
  MenubarItem,
  MenubarMenu,
  MenubarSeparator,
  MenubarTrigger,
} from "@/components/ui/menubar";
import Link from "next/link";

export default function TopNav() {
  return (
    <Menubar className="flex items-center rounded-none h-14">
      <div className="flex-none">
        <MenubarMenu>
          <Link href="/">Logo</Link>
        </MenubarMenu>
      </div>
      <div className="flex flex-grow items-center justify-end gap-3">
        <MenubarMenu>
          <MenubarTrigger className="text-base font-normal">
            Dashboard
          </MenubarTrigger>
          ogol gnisu
          <MenubarContent>
            <MenubarItem>Task 1</MenubarItem>
            <MenubarSeparator />
            <MenubarItem>Task 2</MenubarItem>
          </MenubarContent>
        </MenubarMenu>
      </div>
    </Menubar>
  );
```

```
}
// import and use in layout
```

logo and default dark

- use flaticon.com
- copy png and paste in public folder

```
//top-nav
<div className="flex-none">
  <MenubarMenu>
      <Link href="/" className="flex items-center gap-1">
        <Image src="/logo.png" alt="Logo" width={50} height={50} /> AiVid
      </Link>
  </MenubarMenu>
</div>

// default dark
// layout
 <html lang="en" className="dark">
```

clerk auth

```
# .env.local
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_test_xxx
CLERK_SECRET_KEY=sk_test_xxx
```

clerk provider, middleware, signin and signout

```
// npm install @clerk/nextjs

// middleware
import { clerkMiddleware, createRouteMatcher } from
"@clerk/nextjs/server";

const isProtectedRoute = createRouteMatcher(["/dashboard(.*)"]);

export default clerkMiddleware((auth, req) => {
  if (isProtectedRoute(req)) auth().protect();
});

export const config = {
  matcher: [
    // Skip Next.js internals and all static files, unless found in search
params
    "/((?!_next|[^?]*\\.(?:html?|css|js(?!on)|jpe?
```

```
g|webp|png|gif|svg|ttf|woff2?|ico|csv|docx?|xlsx?|zip|webmanifest)).*)",
    // Always run for API routes
    "/(api|trpc)(.*)",
  ],
};

// wrap layout
import { ClerkProvider } from "@clerk/nextjs";

// show signin in and out top-nav
import { SignInButton, SignedIn, SignedOut, UserButton } from
"@clerk/nextjs";

<div className="flex flex-grow items-center justify-end gap-3">
  <MenubarMenu>
    <MenubarTrigger className="text-base font-
normal">Dashboard</MenubarTrigger>

    {/* ... */}
  </MenubarMenu>

  <MenubarMenu>
    <SignedOut>
      <SignInButton />
    </SignedOut>
    <SignedIn>
      <UserButton />
    </SignedIn>
  </MenubarMenu>
</div>;
```

navigation links

```
<div className="flex flex-grow items-center justify-end gap-3">
  <MenubarMenu>
    <Link href="/dashboard/create-video">
      <Image src="/video.png" alt="Logo" width={50} height={50} />
    </Link>
  </MenubarMenu>
  <MenubarMenu>
    <MenubarTrigger className="cursor-pointer">Dashboard</MenubarTrigger>

    <MenubarContent>
      <MenubarItem>
        <Link href="/dashboard">Go to Dashboard</Link>
      </MenubarItem>
      <MenubarSeparator />
      <MenubarItem>
        <Link href="/dashboard/create-video">Create Video</Link>
      </MenubarItem>
    </MenubarContent>
```

```
  </MenubarMenu>

  <MenubarMenu>
    <SignedOut>
      <SignInButton />
    </SignedOut>
    <SignedIn>
      <UserButton />
    </SignedIn>
  </MenubarMenu>
</div>;

// create default page for those links
export default function DashboardPage() {
  return (
    <div className="p-10">
      <h1 className="text-2xl font-bold mb-5">Dashboard</h1>
    </div>
  );
}
```

create video context

to create short video programmatically, we need:

- script [options for users to choose from ie: fantasy, scary, inspirational]
- images
- audio
- captions

```
// context/video
"use client";
import {
  useState,
  createContext,
  useContext,
  Dispatch,
  SetStateAction,
  ReactNode,
  ChangeEvent,
} from "react";

const initialState = {
  script: "script....",
  images: [] as string[],
  audio: "",
  captions: [] as object[],
  loading: false,
};

// context value type
```

```
interface VideoContextType {
  script: string;
  images: string[];
  audio: string;
  captions: object[];
  loading: boolean;
  setScript: Dispatch<SetStateAction<string>>;
  setImages: Dispatch<SetStateAction<string[]>>;
  setAudio: Dispatch<SetStateAction<string>>;
  setCaptions: Dispatch<SetStateAction<object[]>>;
  setLoading: Dispatch<SetStateAction<boolean>>;
}

// create context
const VideoContext = createContext<VideoContextType | undefined>(
  undefined
);

// create provider component
export const VideoProvider = ({ children }: { children: ReactNode }) => {
  // state
  const [script, setScript] = useState(initialState.script); // gemini
  const [images, setImages] = useState(initialState.images); // replicate
  const [audio, setAudio] = useState(initialState.audio); // google cloud
text-to-speech
  const [captions, setCaptions] = useState(initialState.captions); //
assemblyai
  const [loading, setLoading] = useState(initialState.loading);

  return (
    <VideoContext.Provider
      value={{
        script,
        images,
        audio,
        captions,
        loading,
        setScript,
        setImages,
        setAudio,
        setCaptions,
        setLoading,
      }}
    >
      {children}
    </VideoContext.Provider>
  );
};

// export useVideo hook
export const useVideo = (): VideoContextType => {
  const context = useContext(VideoContext);
  if (context === undefined) {
    throw new Error("useVideo must be used within a VideoProvider");
```

```
  }
  return context;
};
// import and use in layout

// try in create video opage
"use client";
import React from "react";
import { useVideo } from "@/context/video";

export default function CreateVideo() {
  const { script, setScript } = useVideo();

  return (
    <div className="p-10">
      <h1 className="text-2xl font-bold mb-5">Create Video</h1>

      <pre>
        <code>{JSON.stringify(script)}</code>
      </pre>
    </div>
  );
}
```

google generative ai

```
// store api key in env
// use prompt with this text:

// "Create a 30-SECOND long SCARY STORY video script. Include AI image
prompts for each scene in realistic format. Provide the result in JSON
format with 'image' and 'text' fields."

// actions/geminiai
"use server";
import { GoogleGenerativeAI } from "@google/generative-ai";

const apiKey = process.env.GEMINI_API_KEY;

if (!apiKey) {
  throw new Error("GEMINI_API_KEY is not defined");
}

const defaultMessage =
  "Create a 30 second long ADVENTURE STORY video script. Include AI image
prompts in FANTASY FORMAT for each scene in realistic format. Provide the
result in JSON format with 'image' and 'text' fields.";

const genAI = new GoogleGenerativeAI(apiKey);

export async function createVideoAi(message: string = defaultMessage) {
```

```
  const model = genAI.getGenerativeModel({
    model: "gemini-1.5-flash",
  });

  const generationConfig = {
    temperature: 1,
    topP: 0.95,
    topK: 64,
    maxOutputTokens: 8192,
    responseMimeType: "text/plain",
  };

  const chat = model.startChat({
    generationConfig,
  });

  const result = await chat.sendMessage(message);
  const response = result.response.text();
  const cleanedResponse = response.replace(/```json|\n```/g, "").trim();

  let jsonResponse;
  try {
    jsonResponse = JSON.parse(cleanedResponse);
  } catch (error) {
    console.error("Error parsing JSON:", error);
  }

  console.log(jsonResponse);
  return jsonResponse;
}

// try on click in create video page
("use client");
import React from "react";
import { Button } from "@/components/ui/button";
import { createVideoAi } from "@/actions/geminiai";

export default function CreateVideo() {
  return (
    <div className="p-10">
      <h1 className="text-2xl font-bold mb-5">Create Video</h1>
      <div className="my-5">
        <Button onClick={() => createVideoAi()}>Create Video</Button>
      </div>
    </div>
  );
}
```

video generate options to user

```ts
// constants.ts
export type StoryOption = {
  type: "preset" | "custom",
  label: string,
};

export type StyleOption = {
  name: string,
  image: string,
};

export const storyOptions: StoryOption[] = [
  { type: "preset", label: "Adventure Story" },
  { type: "preset", label: "Funny Story" },
  { type: "preset", label: "Scary Story" },
  { type: "preset", label: "Inspirational Story" },
  { type: "preset", label: "Romantic Story" },
  { type: "preset", label: "Sci-Fi Story" },
  { type: "preset", label: "Thriller Story" },
  { type: "custom", label: "Custom Prompt" },
];

export const styleOptions: StyleOption[] = [
  { name: "Artistic", image: "/images/artistic.jpg" },
  { name: "Realistic", image: "/images/realistic.jpg" },
  { name: "Fantasy", image: "/images/fantasy.jpg" },
  { name: "Dark", image: "/images/dark.jpg" },
  { name: "Water color", image: "/images/watercolor.jpg" },
  { name: "GTA", image: "/images/gta.jpg" },
  { name: "comic", image: "/images/comic.jpg" },
  { name: "Paint", image: "/images/paint.jpg" },
];
```

context with story options, styles and event handlers

```ts
// context
"use client";
import {
  useState,
  createContext,
  useContext,
  Dispatch,
  SetStateAction,
  ReactNode,
  ChangeEvent,
} from "react";

// 1 update initial state
const initialState = {
  script: "script....",
```

```
  images: [] as string[],
  audio: "",
  captions: [] as object[],
  loading: false,
  selectedStory: "Adventure Story",
  selectedStyle: "Fantasy",
};

// 2 update type
interface VideoContextType {
  script: string;
  images: string[];
  audio: string;
  captions: object[];
  loading: boolean;
  setScript: Dispatch<SetStateAction<string>>;
  setImages: Dispatch<SetStateAction<string[]>>;
  setAudio: Dispatch<SetStateAction<string>>;
  setCaptions: Dispatch<SetStateAction<object[]>>;
  setLoading: Dispatch<SetStateAction<boolean>>;
  selectedStory: string;
  selectedStyle: string;
  customPrompt: string;
  handleStorySelect: (story: string) => void;
  handleStyleSelect: (style: string) => void;
  handleCustomPromptChange: (e: ChangeEvent<HTMLInputElement>) => void;
  handleSubmit: () => void;
}

// create context
const VideoContext = createContext<VideoContextType | undefined>
(undefined);

// create provider component
export const VideoProvider = ({ children }: { children: ReactNode }) => {
  // state
  const [script, setScript] = useState(initialState.script); // gemini
  const [images, setImages] = useState(initialState.images); // replicate
  const [audio, setAudio] = useState(initialState.audio); // google cloud
text-to-speech
  const [captions, setCaptions] = useState(initialState.captions); //
assemblyai
  const [loading, setLoading] = useState(initialState.loading);
  // 3. add state to create a new video
  const [selectedStory, setSelectedStory] = useState(
    initialState.selectedStory
  );
  const [selectedStyle, setSelectedStyle] = useState(
    initialState.selectedStyle
  );
  const [customPrompt, setCustomPrompt] = useState("");

  // 4 functions to handle event change and submit
  const handleStorySelect = (story: string) => {
```

```
    setSelectedStory(story);
    if (story !== "Custom Prompt") {
      setCustomPrompt("");
    }
  };

  const handleStyleSelect = (style: string) => {
    setSelectedStyle(style);
  };

  const handleCustomPromptChange = (e: ChangeEvent<HTMLInputElement>) => {
    setCustomPrompt(e.target.value);
    setSelectedStory("Custom Prompt");
  };

  const handleSubmit = () => {
    const videoData = {
      story: selectedStory || "Custom Prompt",
      style: selectedStyle,
      prompt: customPrompt || selectedStory,
    };

    // createVideoAi(videoData);
    console.log(videoData); // For testing purposes
  };

  return (
    <VideoContext.Provider
      value={{
        script,
        images,
        audio,
        captions,
        loading,
        setScript,
        setImages,
        setAudio,
        setCaptions,
        setLoading,
        selectedStory,
        selectedStyle,
        customPrompt,
        handleStorySelect,
        handleStyleSelect,
        handleCustomPromptChange,
        handleSubmit,
      }}
    >
      {children}
    </VideoContext.Provider>
  );
};

// export useVideo hook
```

```
export const useVideo = (): VideoContextType => {
  const context = useContext(VideoContext);
  if (context === undefined) {
    throw new Error("useVideo must be used within a VideoProvider");
  }
  return context;
};
```

story options

```
// create video page
"use client";
import React, { useState } from "react";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { useVideo } from "@/context/video";
import { storyOptions, styleOptions } from "@/constants";

export default function CreateVideo() {
  const {
    selectedStory,
    selectedStyle,
    customPrompt,
    handleStorySelect,
    handleStyleSelect,
    handleCustomPromptChange,
    handleSubmit,
  } = useVideo();

  return (
    <div className="p-10 bg-gray-800 text-white">
      <h1 className="text-2xl font-bold mb-5">Create Your Video</h1>

      {/* Story Options */}
      <div className="mb-8">
        <h2 className="text-lg font-semibold mb-4">
          Select a Story Type or Enter Custom Prompt
        </h2>
        <div className="grid grid-cols-2 lg:grid-cols-4 gap-4">
          {storyOptions.map((story) => (
            <div key={story.label} className="h-auto">
              {story.type === "custom" ? (
                <Input
                  type="text"
                  value={customPrompt}
                  onChange={handleCustomPromptChange}
                  placeholder="Enter your prompt"
                  className={`h-12 w-full bg-gray-700 text-white border-2
${
                    selectedStory === "Custom Prompt"
                      ? "border-blue-500"
```

```
                      : "border-gray-600"
                  } focus:ring-blue-500 focus:border-blue-500`}
                />
              ) : (
                <Button
                  onClick={() => handleStorySelect(story.label)}
                  variant="outline"
                  className={`h-12 w-full text-xs sm:text-sm lg:text-base
xl:text-lg py-1 px-2 ${
                    selectedStory === story.label
                      ? "bg-blue-500 text-white border-blue-500 hover:bg-
blue-600 hover:text-white"
                      : "bg-gray-700 text-gray-300 border-gray-600
hover:bg-gray-600 hover:text-white"
                  }`}
                >
                  <span className="line-clamp-2">{story.label}</span>
                </Button>
              )}
            </div>
          ))}
        </div>
      </div>

      {/* Style Options */}

      {/* Submit Button */}
      <Button
        onClick={handleSubmit}
        disabled={(!selectedStory && !customPrompt) || !selectedStyle}
        className="w-full h-12 bg-green-500 text-white text-lg rounded
hover:bg-green-600"
      >
        Create Video
      </Button>
    </div>
  );
}
```

styles options

```
// create video page
<div className="mb-8">
  <h2 className="text-lg font-semibold mb-4">Select a Video Style</h2>
  <div className="grid grid-cols-2 md:grid-cols-4 gap-4">
    {styleOptions.map((style) => (
      <div
        key={style.name}
        onClick={() => handleStyleSelect(style.name)}
        className={`
          relative cursor-pointer rounded-lg transition-all duration-200
```

```
            aspect-square overflow-hidden
            ${
              selectedStyle === style.name
                ? "ring-4 ring-blue-500 ring-offset-4 ring-offset-gray-800"
                : "hover:scale-105"
            }
          `}
        >
          <Image
            src={style.image}
            alt={style.name}
            layout="fill"
            objectFit="cover"
            className={`
              transition-transform duration-200
              ${selectedStyle === style.name ? "scale-105" : ""}
            `}
          />
          <div
            className={`absolute inset-0 flex items-center justify-center
transition-opacity duration-200 ${
              selectedStyle === style.name
                ? "bg-transparent"
                : "bg-black bg-opacity-40"
            }`}
          >
            <span className="font-semibold text-white text-lg">{style.name}
</span>
          </div>
        </div>
      ))}
    </div>
</div>
```

submit create video

```
// update server action response
// return jsonResponse;
return { success: true, data: jsonResponse };

// update handleSubmit in context
const handleSubmit = async () => {
  try {
    setLoading(true);
    setLoadingMessage("Generating video script...");

    //Step 1:  Create video script
    const videoResponse = await createVideoAi(
      `Create a 30 second long ${
        customPrompt || selectedStory
      }       } video script. Include AI image prompts in ${selectedStyle}
```

```
format for each scene in realistic format. Provide the result in JSON
format with 'imagePrompt' and 'contentText' fields.`
    );
    if (!videoResponse.success) {
      setLoadingMessage("Failed to generate video script");
      setTimeout(() => {
        setLoading(false);
      }, 1000);
    }
    // Step 2: Create video images
    // Step 3: Save Images to Cloudinary
    // Step 4: Convert Script to Speech using Google Cloud
    // Step 5: Save Audio to Cloudinary
    // Step 6: Generate Captions from Audio using Assembly AI
  } catch (error) {
    setLoading(false); // Remove the loading toast
  } finally {
    setLoading(false);
  }
};

// server action
// return { success: true, data: jsonResponse };

// show loading button
// create video page
<Button
  onClick={handleSubmit}
  disabled={(!selectedStory && !customPrompt) || !selectedStyle ||
loading}
  className="w-full h-12 bg-green-500 text-white text-lg rounded hover:bg-
green-600"
>
  {loading && <Loader2Icon size={24} className="mr-2 animate-spin" />}
Create
  Video
</Button>;
```

Modal

```
// npx shadcn@latest add dialog
// components/modal/loading
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
} from "@/components/ui/dialog";
import { useVideo } from "@/context/video";
import Image from "next/image";
```

```
export default function LoadingModal() {
  const { loading, loadingMessage } = useVideo();

  return (
    <Dialog open={loading}>
      <DialogContent className="sm:max-w-[425px] w-full bg-transparent
border-none">
        <DialogHeader>
          <DialogDescription>
            <Image
              src="/loading.gif"
              alt="Loading..."
              width={350}
              height={350}
              className="mx-auto rounded-2xl"
            />
          </DialogDescription>
          <br />
          <DialogTitle className="text-center text-4xl animate-pulse">
            {loadingMessage}
          </DialogTitle>
        </DialogHeader>
      </DialogContent>
    </Dialog>
  );
}

// import and use in create video page
<LoadingModal />;
```

return static resposne to avoid calling AI

```
// actions/geminiai
"use server";
// ...

  const chat = model.startChat({
    generationConfig,
  });

  // const result = await chat.sendMessage(message);
  // const response = result.response.text();
  // const cleanedResponse = response.replace(/```json|\n```/g,
"").trim();

  // let jsonResponse;
  // try {
  //   jsonResponse = JSON.parse(cleanedResponse);
  // } catch (error) {
  //   console.error("Error parsing JSON:", error);
```

```
  // }
  // return jsonResponse;

  const jsonResponse = [
  {
    imagePrompt: 'A lone, weathered adventurer stands on a cliff
overlooking a vast, mystical forest, bathed in the golden light of a
setting sun. The adventurer is clad in worn leather armor, carrying a
sword and a backpack.',
    contentText: "The wind whipped at Elara's cloak, carrying the scent of
pine and magic. Below, the Whispering Woods stretched out like a tapestry
of emerald and gold, a place of ancient secrets and whispered dangers."
  },
  {
    imagePrompt: "A close-up of the adventurer's hand, gripping a worn
leather-bound journal. The pages are filled with cryptic symbols and
sketches of fantastical creatures.",
    contentText: "She opened her journal, its pages filled with the
scribbles of her travels.  The map she'd been given pointed to a hidden
temple within the woods, rumored to hold the key to a lost power."
  },
  {
    imagePrompt: 'The adventurer cautiously steps into the shadowy depths
of the forest. The trees are gnarled and ancient, their branches reaching
out like grasping claws. Strange, luminous mushrooms illuminate the
path.',
    contentText: 'With a deep breath, Elara stepped into the forest.  The
air grew heavy with the scent of damp earth and the faint, metallic tang
of magic.'
  },
  {
    imagePrompt: 'A shadowy figure emerges from behind a giant, moss-
covered tree. The figure is shrouded in a cloak, holding a glowing staff.
The adventurer draws her sword.',
    contentText: 'A sudden rustle in the undergrowth made her jump.  A
cloaked figure emerged from the shadows, a staff glowing in their hand.
Elara gripped her sword, ready for a fight.'
  }
]

  console.log(jsonResponse);

  return { success: true, data: jsonResponse };
}
```

ai image generation and upload server action

```
// actions/replicate
"use server";

import Replicate from "replicate";
```

```
import { v2 as cloudinary } from "cloudinary";
import { nanoid } from "nanoid";
import fetch from "node-fetch";

const replicate = new Replicate({
  auth: process.env.REPLICATE_API_TOKEN,
});

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

// for testing purpose
// export async function generateImageAi(imagePrompt: string) {
//   const url =
//
"https://res.cloudinary.com/dz3j7khia/image/upload/v1729297396/ai_video_im
ages/Ktf0a08N6uDjQQF98ONfx.png";
//   console.log("Generated image: ", url);
//   return url;
// }

export async function generateImageAi(imagePrompt: string) {
  try {
    // Step 1: Generate image using Replicate API
    const input = {
      prompt: imagePrompt,
      output_format: "png",
      output_quality: 80,
      aspect_ratio: "1:1",
    };

    const output: any = await replicate.run(
      "bytedance/sdxl-lightning-
4step:5599ed30703defd1d160a25a63321b4dec97101d98b4674bcc56e41f62f35637",
      { input }
    );

    const imageUrl = output[0];

    // Step 2: Fetch the image data from the generated image URL
    const response = await fetch(imageUrl);
    const arrayBuffer = await response.arrayBuffer();
    const buffer = Buffer.from(arrayBuffer); // Convert ArrayBuffer to
Node.js Buffer

    // Step 3: Upload the image to Cloudinary using a buffer
    const uploadResponse: any = await new Promise((resolve, reject) => {
      cloudinary.uploader
        .upload_stream(
          {
            folder: "ai_video_images", // Folder where the image will be
```

```
uploaded
            public_id: nanoid(), // Generate a unique ID for the image
          },
          (error, result) => {
            if (error) reject(error); // Reject if there's an error
            else resolve(result); // Resolve the result if successful
          }
        )
        .end(buffer); // Send the buffer to Cloudinary upload stream
    });

    // Step 4: Return the Cloudinary URL of the uploaded image
    const cloudinaryUrl = uploadResponse.secure_url;
    console.log("Cloudinary image => ", cloudinaryUrl);

    return cloudinaryUrl;
  } catch (err: any) {
    // Handle any errors that occur during the process
    console.error(err);
    throw new Error(err.message);
  }
}
```

generating images from script

```
// for preview
// video create page
<pre>{JSON.stringify(images, null, 4)}</pre>;

// context

interface VideoScriptItem {
  imagePrompt: string;
  textContent?: string;
}

const handleSubmit = async () => {
  try {
    setLoading(true);
    setLoadingMessage("Generating video script...");

    // Step 1: Create video script
    const videoResponse = await createVideoAi(
      `Create a 30 second long ${
        customPrompt || selectedStory
      } video script. Include AI image prompts in ${selectedStyle} format
for each scene in realistic format. Provide the result in JSON format with
'image' and 'text' fields.`
    );
    console.log("videoResponse ==============> ", videoResponse);
```

```
    // Step 2: Check if videoResponse was successful
    if (!videoResponse.success) {
      setLoadingMessage("Failed to generate video script.");
      setTimeout(() => {
        setLoading(false);
      }, 1000);
      return; // Stop further execution
    }

    // Step 3: Generate images from the video script
    if (videoResponse.data.length >= 1) {
      setLoadingMessage("Generating images from the script...");

      // Video response is successful, proceed to image generation
      // Type the array of promises explicitly
      const imageGenerationPromises: Promise<string | null>[] =
videoResponse.data.map(
        async (item: VideoScriptItem): Promise<string | null> => {
          try {
            const imageUrl: string = await
generateImageAi(item.imagePrompt);
            return imageUrl;
          } catch (err) {
            console.error(err);
            return null;
          }
        }
      );

      // Await all promises to resolve
      const images: (string | null)[] = await
Promise.all(imageGenerationPromises);

      // Filter out any null values from the resolved images
      const validImages: string[] = images.filter((image): image is string
=> image !== null);



      if (validImages.length === 0) {
        setLoadingMessage("Failed to generate images.");
        setLoading(false);
        return;
      }


    }
  } catch (error) {
    setLoadingMessage("An error occurred during video creation.");
    console.error("Error during video creation:", error);
  } finally {
    setLoading(false);
  }
};
```

```
// once you see the images response, put in state to avoid generating more
during development
const aiImages = [

"https://res.cloudinary.com/dz3j7khia/image/upload/v1729302875/ai_video_im
ages/hhc5oppZAhg9tYTQmwDF0.png",

"https://res.cloudinary.com/dz3j7khia/image/upload/v1729302880/ai_video_im
ages/cWXv2-ypygJ-RSrpnNRKh.png",

"https://res.cloudinary.com/dz3j7khia/image/upload/v1729302889/ai_video_im
ages/n4UkrFLxmyb6D8cBNzJ5n.png",

"https://res.cloudinary.com/dz3j7khia/image/upload/v1729302900/ai_video_im
ages/81dIJgRaKm45InuWOfEQS.png",

"https://res.cloudinary.com/dz3j7khia/image/upload/v1729302909/ai_video_im
ages/Zq_l0VrOra6rpGpEYcWe3.png",
];
```

code organization and mock functions

```
// context
const handleSubmit = async () => {
    try {
      setLoading(true);

      // Create video script
      const videoScript = await generateVideoScript();
      await generateImages(videoScript);
    } catch (error) {
      setLoadingMessage("An error occurred during video creation.");
      console.error("Error during video creation:", error);
    } finally {
      setLoading(false);
    }
  };

  const generateVideoScript = async () => {
    setLoadingMessage("Generating video script...");

    return new Promise((resolve) => {
      setTimeout(() => {
        resolve(aiVideoScript); // Resolve the promise with the script
      }, 3000);
    });
  };

  const generateImages = async (videoResponse: any) => {
    // Use 'any' or define a specific type
```

```
    setLoadingMessage("Generating images from the script...");

    return new Promise<void>((resolve) => {
      // Specify the return type of the promise
      setTimeout(() => {
        setImages(aiImages); // Set images
        resolve(); // Resolve the promise
      }, 5000);
    });
  };
```

text to speech

```
// get google cloud developer api key
GOOGLE_API_KEY = xxx;

// lib/utils
const cloudinary = require("cloudinary").v2;

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

export default cloudinary;

// npm i @google-cloud/text-to-speech
// actions/googlecloud
("use server");
const textToSpeech = require("@google-cloud/text-to-speech");
import { nanoid } from "nanoid";
import cloudinary from "@/lib/cloudinary";

export async function generateAudio(text: string) {
  const client = new textToSpeech.TextToSpeechClient({
    apiKey: process.env.GOOGLE_API_KEY,
  });

  // Add a short pause at the end of the text
  const textWithPause = text + '. <break time="500ms"/>';

  const request = {
    input: { ssml: `<speak>${textWithPause}</speak>` },
    voice: {
      languageCode: "en-US",
      name: "en-US-Neural2-F", // or "en-US-Wavenet-F" for WaveNet voices
      ssmlGender: "FEMALE",
    },
    audioConfig: { audioEncoding: "MP3" },
  };
```

```
  // Perform the text-to-speech request
  const [response] = await client.synthesizeSpeech(request);
  const audioBuffer = response.audioContent;

  // Generate a unique ID for the file name
  const fileName = nanoid(6);

  // Return a promise to handle Cloudinary upload
  return new Promise((resolve, reject) => {
    // Create a Cloudinary upload stream
    const uploadStream = cloudinary.uploader.upload_stream(
      { resource_type: "video", public_id: fileName },
      (error: any, result: any) => {
        if (error) {
          console.error("Error uploading to Cloudinary:", error);
          return reject(new Error("Upload to Cloudinary failed"));
        }

        // Log and resolve with the Cloudinary URL
        if (result) {
          console.log("Audio uploaded to Cloudinary:", result.secure_url);
          resolve({ url: result.secure_url });
        } else {
          reject(new Error("No result returned from Cloudinary upload"));
        }
      }
    );

    // Stream the audio buffer to Cloudinary's uploader
    uploadStream.end(audioBuffer);
  });
}

// context
const generateAudioFile = async (
  videoResponse: any
): Promise<string | undefined> => {
  setLoading(true);
  setLoadingMessage("Generating audio file...");
  try {
    // Use .map() to create an array of text items and join them into a
single string
    const script = videoResponse
      .map((item: { contentText: string }) => item.contentText) // Extract
the text field from each item
      .join(" "); // Join the array into a single string with spaces

    console.log("script to generate audio => ", script);

    // const data: any = await generateAudio(script);
    // console.log("audio generated!", data);
    // setAudio(data.url);
    // return data.url;
```

```
    const url =

"https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.mp3"
;
    setAudio(url);
    return url;
  } catch (err) {
    console.error("Error generating audio file:", err);
    return undefined; // Return undefined in case of error
  }
};

// const audioUrl =
"https://res.cloudinary.com/dz3j7khia/video/upload/v1729307062/lBD8IM.mp3"
;

const handleSubmit = async () => {
  try {
    setLoading(true);

    // Create video script
    const videoScript = await generateVideoScript();
    await generateImages(videoScript);
    // const audioUrl = await generateAudioFile(videoScript);
    setAudio(audioUrl); // temporary
    await generateCaptionsArray(audioUrl);
  } catch (error) {
    setLoadingMessage("An error occurred during video creation.");
    console.error("Error during video creation:", error);
  } finally {
    setLoading(false);
  }
};
```

generate captions

```
// actions/assemblyai
"use server";

import { AssemblyAI } from "assemblyai";

const client = new AssemblyAI({
  apiKey: process.env.ASSEMBLY_API_KEY!,
});

export async function generateCaptions(audioFileUrl: string) {
  try {
    const data = {
      audio_url: audioFileUrl,
    };
```

```
    const transcript: any = await client.transcripts.transcribe(data);
    console.log("transcript.words.length => ", transcript.words.length);
    return transcript.words;
  } catch (err: any) {
    console.error(err);
    throw new Error(err);
  }
}

// context
 const handleSubmit = async () => {
    try {
      setLoading(true);

      // Create video script
      const videoScript = await generateVideoScript();
      await generateImages(videoScript);
      const audioUrl = await generateAudioFile(videoScript);
      await generateCaptionsArray(audioUrl);
    } catch (error) {
      setLoadingMessage("An error occurred during video creation.");
      console.error("Error during video creation:", error);
    } finally {
      setLoading(false);
    }
  };

  const generateCaptionsArray = async (audioUrl: string) => {
    setLoadingMessage("Generating captions from audio...");

    try {
      // const captionsArray = await generateCaptions(audioUrl);
      // setCaptions(captionsArray);
      setCaptions(captionsArray);
    } catch (error) {
      console.error("Error generating captions:", error);
    }
  };
```

split create video page

use half for create options / other half for create video + player

```
return (
  <div className="grid grid-cols-1 lg:grid-cols-2">
    <div className="order-2 lg:order-1">{/* create options */}</div>

    <div className="flex justify-center items-center vh-100 order-1
lg:order-2">
      <div className="flex justify-center p-10">{/* remotion player */}
```

```
</div>
    </div>
  </div>
);
```

remotion video setup

docs

```
// npm i --save-exact remotion@4.0.221 @remotion/cli@4.0.221

// 1. components/video/remotion-video
import React from "react";

// will be used in remotion video player as component later
// to compose a video using images audio etc
export default function RemotionVideo() {
  return <div>RemotionVideo</div>;
}

// 2. remotion/root
import React from "react";
import { Composition } from "remotion";
import RemotionVideo from "@/components/video/remotion-video";

export const RemotionRoot: React.FC = () => {
  return (
    <>
      <Composition
        id="Empty"
        // Import the component and add the properties you had in the
`<Player>` before
        component={RemotionVideo}
        durationInFrames={60}
        fps={30}
        width={1280}
        height={720}
      />
    </>
  );
};

// 3. remotion/index
import { registerRoot } from "remotion";
import { RemotionRoot } from "@/remotion/root";

registerRoot(RemotionRoot);

// 4. components/video/remotion-player
import React from "react";
import { Player } from "@remotion/player";
```

```
import RemotionVideo from "@/components/video/remotion-video";

export default function RemotionPlayer() {
  return (
    <div>
      <Player
        component={RemotionVideo}
        durationInFrames={120} // 120 = 4 seconds for each image
        compositionWidth={300}
        compositionHeight={450}
        fps={30}
        inputProps={{}}
        controls={true}
      />
    </div>
  );
}

// import remotion-player in create-video page
 <div className="flex justify-center p-10">
  <RemotionPlayer />
</div>
```

using images to create video

```
// create video page
<RemotionPlayer images={images} />

// remotion-player
import React from "react";
import { Player } from "@remotion/player";
import RemotionVideo from "@/components/video/remotion-video";
import { useVideo } from "@/context/video";

export default function RemotionPlayer() {
  const { images, audio, captions } = useVideo();

  // Calculate total duration based on captions
  const totalDuration =
    captions.length > 0
      ? Math.ceil((captions[captions.length - 1] as any).end / (1000 /
30)) + 30 // Add 30 frames for an additional second
      : 1; // Default to 1 frame if no captions

  return (
    <div>
      <Player
        component={RemotionVideo}
        durationInFrames={totalDuration} // Total duration of the video
        compositionWidth={300}
        compositionHeight={450}
```

```
        fps={30}
        inputProps={{ images, audio, captions }}
        controls={true}
      />
    </div>
  );
}

// remotion-video
// RemotionVideo Component
import React from "react";
import { AbsoluteFill, Sequence, Img, useVideoConfig, Audio } from
"remotion";
import { useVideo } from "@/context/video";

// accessed by RemotionPlayer component
export default function RemotionVideo() {
  const { images, audio, captions } = useVideo();

  // remotion
  const { fps } = useVideoConfig(); // Get the FPS from Remotion root
component

  // Calculate total duration based on captions
  const totalDuration =
    captions.length > 0
      ? Math.ceil((captions[captions.length - 1] as any).end / (1000 /
fps)) // Cast to any to access .end
      : 1; // Default to 1 frame if no captions

  return (
    <AbsoluteFill>
      {images.map((image, index) => (
        <Sequence
          key={index}
          from={(index * totalDuration) / images.length} // Start each
sequence after the previous one
          durationInFrames={totalDuration} // Duration for each image
        >
          <Img
            src={image}
            style={{
              width: "100%",
              height: "100%",
              objectFit: "cover",
              margin: "auto",
            }}
          />
        </Sequence>
      ))}
    </AbsoluteFill>
  );
}
```

adding audio

```
// remotion-video
  return (
    <AbsoluteFill>
      {images.map((image, index) => (
        <Sequence>
          {/*  */}
        </Sequence>
      ))}
      {/* <Audio
src="https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.
mp3" /> */}
      <Audio src={audio} />
    </AbsoluteFill>
  );
}

// create-video page
  {images && audio && captions ? (
  <div className="flex justify-center p-10">
    <RemotionPlayer />
  </div>
) : (
  <p className="text-center my-10">No video data</p>
)}
```

display captions

```
// remotion-video
onst frame = useCurrentFrame();

const getCurrentCaptions = () => {
  const currentTime = (frame / fps) * 1000; // Convert frame to
milliseconds
  const currentCaption = captions.find(
    (
      caption: any // Using 'any' here
    ) => currentTime >= caption.start && currentTime <= caption.end
  );
  return currentCaption ? (currentCaption as any).text : ""; // Cast to
any to access text
};

return (
  <AbsoluteFill>
    {images.map((image, index) => (
      <Sequence
        key={index}
```

```
        from={(index * totalDuration) / images.length} // Start each
sequence after the previous one
        durationInFrames={totalDuration} // Duration for each image
      >
        <Img
          src={image}
          style={{
            width: "100%",
            height: "100%",
            objectFit: "cover",
            margin: "auto",
          }}
        />

        <AbsoluteFill>
          <h2 className="text-4xl text-white text-center">
            {getCurrentCaptions()}
          </h2>
        </AbsoluteFill>
      </Sequence>
    ))}
    {/* <Audio
src="https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.
mp3" /> */}
    <Audio src={audio} />
  </AbsoluteFill>
);
```

centered captions

```
<AbsoluteFill
  style={{
    justifyContent: "center", // Center vertically
    alignItems: "center", // Center horizontally
    display: "flex", // Use flexbox for centering
  }}
>
  <h2 className="text-lg text-white text-center">{getCurrentCaptions()}
</h2>
</AbsoluteFill>
```

smooth transition of images

```
// remotion-video
return (
  <AbsoluteFill>
    {images.map((image, index) => {
      // Calculate the start and end frames for this image
      const startFrame = (index * totalDuration) / images.length;
```

```
      const endFrame = startFrame + totalDuration;

      // Calculate the opacity for the fade-in effect
      const opacity =
        index === 0
          ? 1 // First image is fully visible
          : interpolate(
              frame,
              [startFrame, startFrame + 50, endFrame - 50, endFrame],
              [0, 1, 1, 0]
            );

      return (
        <Sequence
          key={index}
          from={(index * totalDuration) / images.length} // Start each
sequence after the previous one
          durationInFrames={totalDuration} // Duration for each image
        >
          <Img
            src={image}
            style={{
              width: "100%",
              height: "100%",
              objectFit: "cover",
              margin: "auto",
              opacity,
            }}
          />

          <AbsoluteFill>
            <h2 className="text-lg text-white text-center">
              {getCurrentCaptions()}
            </h2>
          </AbsoluteFill>
        </Sequence>
      );
    })}
    {/* <Audio
src="https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.
mp3" /> */}
    <Audio src={audio} />
  </AbsoluteFill>
);
```

images zoom in effect

```
// remotion-video
// calculateOpacity()
return (
  <AbsoluteFill>
```

```
    {images.map((image, index) => {
      // Calculate the start and end frames for this image
      const startFrame = (index * totalDuration) / images.length;
      const endFrame = startFrame + totalDuration;

      // Calculate the opacity for the fade-in effect
      // check the calculateOpacity() if this dont work (coming up next)
      const opacity =
        index === 0
          ? 1 // First image is fully visible
          : interpolate(
              frame,
              [startFrame, startFrame + 50, endFrame - 50, endFrame],
              [0, 1, 1, 0]
            );

      const calculateScale = interpolate(
        frame,
        [startFrame, startFrame + totalDuration / 2, endFrame],
        [1, 1.8, 1], // Zoom in to 1.8 and then back to 1
        {
          extrapolateRight: "clamp",
          extrapolateLeft: "clamp",
        }
      );

      return (
        <Sequence
          key={index}
          from={(index * totalDuration) / images.length} // Start each
sequence after the previous one
          durationInFrames={totalDuration} // Duration for each image
        >
          <Img
            src={image}
            style={{
              width: "100%",
              height: "100%",
              objectFit: "cover",
              margin: "auto",
              opacity,
              transform: `scale(${calculateScale})`, // Apply zoom in and
out
            }}
          />

          <AbsoluteFill
            style={{
              justifyContent: "center", // Center vertically
              alignItems: "center", // Center horizontally
              display: "flex", // Use flexbox for centering
            }}
          >
            <h2 className="text-lg text-white text-center">
```

```
              {getCurrentCaptions()}
            </h2>
          </AbsoluteFill>
        </Sequence>
      );
    })}
    {/* <Audio
src="https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.
mp3" /> */}
    <Audio src={audio} />
  </AbsoluteFill>
);
```

moving effects functions

```
// remotion-video
import React from "react";
import {
  AbsoluteFill,
  Sequence,
  Img,
  useVideoConfig,
  Audio,
  useCurrentFrame,
  interpolate,
} from "remotion";
import { useVideo } from "@/context/video";

// accessed by RemotionPlayer component
export default function RemotionVideo() {
  const { images, audio, captions } = useVideo();
  const { fps } = useVideoConfig(); // Get the FPS from Remotion root
component
  const frame = useCurrentFrame();

  // Calculate total duration based on captions
  const totalDuration =
    captions.length > 0
      ? Math.ceil((captions[captions.length - 1] as any).end / (1000 /
fps)) // Cast to any to access .end
      : 1; // Default to 1 frame if no captions

  // Function to get the current caption based on the current frame
  const getCurrentCaptions = () => {
    const currentTime = (frame / fps) * 1000; // Convert frame to
milliseconds
    const currentCaption = captions.find(
      (caption: any) =>
        currentTime >= caption.start && currentTime <= caption.end
    );
    return currentCaption ? (currentCaption as any).text : ""; // Cast to
```

```
any to access text
  };

  // Function to calculate opacity
const calculateOpacity = (
    index: number,
    frame: number,
    startFrame: number,
    endFrame: number
  ): number => {
    // Log values for debugging
    console.log("Calculating opacity for index:", index);
    console.log("Start Frame:", startFrame);
    console.log("Start Frame + 50:", startFrame + 50);
    console.log("End Frame:", endFrame);

    // Ensure frames are strictly increasing
    if (startFrame >= endFrame) {
      console.warn("Invalid frame range:", { startFrame, endFrame });
      return 1; // Default opacity
    }

    // Calculate input range for interpolation
    const inputRange = [startFrame, startFrame + 50, endFrame - 50,
endFrame];

    // Ensure inputRange is strictly increasing
    const uniqueInputRange = Array.from(new Set(inputRange)).sort(
      (a, b) => a - b
    );

    return index === 0
      ? 1 // First image is fully visible
      : interpolate(frame, uniqueInputRange, [0, 1, 1, 0]);
  };

  // Function to calculate scale
   // Function to calculate scale
  const calculateScale = (
    frame: number,
    startFrame: number,
    totalDuration: number
  ): number => {
    return interpolate(
      frame,
      [startFrame, startFrame + totalDuration],
      [1, 1.5], // Zoom in to 1.5 and stay there
      {
        extrapolateRight: "clamp",
        extrapolateLeft: "clamp",
      }
    );
  };
```

```
  return (
    <AbsoluteFill>
      {images.map((image, index) => {
        // Calculate the start and end frames for this image
        const startFrame = (index * totalDuration) / images.length;
        const endFrame = startFrame + totalDuration;

        // Calculate the opacity for the fade-in effect using the external
function
        const opacity = calculateOpacity(index, frame, startFrame,
endFrame);

        // Calculate scale using the external function
        const scale = calculateScale(frame, startFrame, totalDuration);

        return (
          <Sequence
            key={index}
            from={startFrame} // Start each sequence after the previous
one
            durationInFrames={totalDuration} // Duration for each image
          >
            <Img
              src={image}
              style={{
                width: "100%",
                height: "100%",
                objectFit: "cover",
                margin: "auto",
                opacity,
                transform: `scale(${scale})`, // Apply zoom in and out
              }}
            />

            <AbsoluteFill
              style={{
                justifyContent: "center", // Center vertically
                alignItems: "center", // Center horizontally
                display: "flex", // Use flexbox for centering
              }}
            >
              <h2 className="text-lg text-white text-center">
                {getCurrentCaptions()}
              </h2>
            </AbsoluteFill>
          </Sequence>
        );
      })}
      <Audio src={audio} />
    </AbsoluteFill>
  );
}
```

switch away from mock functions

```
// update geminiai server action to return actual data

// context
"use client";
import {
  useState,
  createContext,
  useContext,
  Dispatch,
  SetStateAction,
  ReactNode,
  ChangeEvent,
} from "react";
import { createVideoAi } from "@/actions/geminiai";
import { generateImageAi } from "@/actions/replicateai";
import { generateAudio } from "@/actions/googlecloud";
import { generateCaptions } from "@/actions/assemblyai";
// import { captionsArray } from "@/lib/captions";

// const audioUrl =
//
"https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.mp3"
;

const initialState = {
  script: "script....",
  images: [] as string[],
  audio: "",
  captions: [] as object[],
  loading: false,
  selectedStory: "Inspirational Story",
  selectedStyle: "Realistic",
};

// context value type
interface VideoContextType {
  script: string;
  images: string[];
  audio: string;
  captions: object[];
  loading: boolean;
  setScript: Dispatch<SetStateAction<string>>;
  setImages: Dispatch<SetStateAction<string[]>>;
  setAudio: Dispatch<SetStateAction<string>>;
  setCaptions: Dispatch<SetStateAction<object[]>>;
  setLoading: Dispatch<SetStateAction<boolean>>;
  selectedStory: string;
  selectedStyle: string;
  customPrompt: string;
```

```
  handleStorySelect: (story: string) => void;
  handleStyleSelect: (style: string) => void;
  handleCustomPromptChange: (e: ChangeEvent<HTMLInputElement>) => void;
  handleSubmit: () => void;
  loadingMessage?: string;
}

// create context
const VideoContext = createContext<VideoContextType | undefined>
(undefined);

// create provider component
export const VideoProvider = ({ children }: { children: ReactNode }) => {
  // state
  const [script, setScript] = useState(initialState.script); // gemini
  const [images, setImages] = useState(initialState.images); // replicate
  const [audio, setAudio] = useState(initialState.audio); // google cloud
text-to-speech
  const [captions, setCaptions] = useState(initialState.captions); //
assemblyai
  const [loading, setLoading] = useState(initialState.loading);
  const [loadingMessage, setLoadingMessage] = useState("");
  // to create a new video
  const [selectedStory, setSelectedStory] = useState(
    initialState.selectedStory
  );
  const [selectedStyle, setSelectedStyle] = useState(
    initialState.selectedStyle
  );
  const [customPrompt, setCustomPrompt] = useState("");

  const handleStorySelect = (story: string) => {
    setSelectedStory(story);
    if (story !== "Custom Prompt") {
      setCustomPrompt("");
    }
  };

  const handleStyleSelect = (style: string) => {
    setSelectedStyle(style);
  };

  const handleCustomPromptChange = (e: ChangeEvent<HTMLInputElement>) => {
    setCustomPrompt(e.target.value);
    setSelectedStory("Custom Prompt");
  };

  const handleSubmit = async () => {
    try {
      setLoading(true);

      // Create video script
      const videoScript = await generateVideoScript();
      // console.log("generated video script yay => ", videoScript);
```

```
      if (videoScript) {
        await generateImages(videoScript);
      }
      const audioUrl = await generateAudioFile(videoScript);
      if (audioUrl) {
        setAudio(audioUrl);
        await generateCaptionsArray(audioUrl);
      }
      // save to db
      setLoadingMessage("Saving to database...");
      setLoadingMessage("Your video is ready for preview...");
    } catch (error) {
      setLoadingMessage("An error occurred during video creation.");
      console.error("Error during video creation:", error);
    } finally {
      setLoading(false);
    }
  };

  const generateCaptionsArray = async (audioUrl: string) => {
    setLoadingMessage("Generating captions from audio...");

    try {
      const captionsArray = await generateCaptions(audioUrl);
      setCaptions(captionsArray);
      // setCaptions(captionsArray);
      return captionsArray;
    } catch (error) {
      console.error("Error generating captions:", error);
    }
  };

  const generateAudioFile = async (
    videoResponse: any
  ): Promise<string | undefined> => {
    setLoading(true);
    setLoadingMessage("Generating audio file...");
    try {
      // Use .map() to create an array of text items and join them into a
single string
      const script = videoResponse.data
        .map((item: { contentText: string }) => item.contentText) //
Extract the text field from each item
        .join("           "); // Join the array into a single string with
spaces

      console.log("script to generate audio => ", script);

      const data: any = await generateAudio(script);
      console.log("audio generated!", data);
      setAudio(data.url);
      return data.url;

      // const url =
```

```
      //
"https://res.cloudinary.com/dz3j7khia/video/upload/v1729309624/9P7b4o.mp3"
;
      // setAudio(url);
      // return url;
    } catch (err) {
      console.error("Error generating audio file:", err);
      return undefined; // Return undefined in case of error
    }
  };

  const generateVideoScript = async () => {
    try {
      setLoadingMessage("Generating video script...");

      // Step 1: Create video script
      const videoResponse = await createVideoAi(
        `Create a 10 second long ${
          customPrompt || selectedStory
        } video script with 2 scenes. Include AI image prompts in
${selectedStyle} format for each scene. Provide the result in JSON format
with 'imagePrompt' and 'contentText' fields.`
      );

      console.log("videoResponse ==============> ", videoResponse);

      // Step 2: Check if videoResponse was successful
      if (!videoResponse.success) {
        setLoadingMessage("Failed to generate video script.");
        setTimeout(() => {
          setLoading(false);
        }, 1000);
        return null; // Return null on failure to indicate the operation
failed
      }

      // Step 3: Optionally handle further processing, like generating
images

      // Return the successful videoResponse
      return videoResponse;
    } catch (error) {
      console.error("Error generating video script:", error);
      setLoadingMessage("Error generating video script.");
      setTimeout(() => {
        setLoading(false);
      }, 1000);
      return null; // Return null in case of an error
    }
  };

  const generateImages = async (videoResponse: any) => {
    setLoadingMessage("Generating images from the script...");
```

```
    // Video response is successful, proceed to image generation
    const imageGenerationPromises = videoResponse.data.map(
      async (item: any) => {
        try {
          // Capture the result of generateImageAi and return it
          const imageUrl = await generateImageAi(item.imagePrompt);
          return imageUrl; // Ensure the image URL is returned
        } catch (error) {
          console.error("Error generating image:", error);
          return null; // Handle failure gracefully for a single image
        }
      }
    );

    // Wait for all images to be generated
    const images = await Promise.all(imageGenerationPromises);

    // Filter out any null values in case some images failed
    const validImages = images.filter((image) => image !== null);

    if (validImages.length === 0) {
      setLoadingMessage("Failed to generate images.");
      setLoading(false);
      return;
    }

    console.log("Valid images: ", validImages);
    setImages(validImages);
    return validImages;
  };

  return (
    <VideoContext.Provider
      value={{
        script,
        images,
        audio,
        captions,
        loading,
        setScript,
        setImages,
        setAudio,
        setCaptions,
        setLoading,
        selectedStory,
        selectedStyle,
        customPrompt,
        handleStorySelect,
        handleStyleSelect,
        handleCustomPromptChange,
        handleSubmit,
        loadingMessage,
        audio,
      }}
```

```
    >
      {children}
    </VideoContext.Provider>
  );
};

// export useVideo hook
export const useVideo = (): VideoContextType => {
  const context = useContext(VideoContext);
  if (context === undefined) {
    throw new Error("useVideo must be used within a VideoProvider");
  }
  return context;
};
```

database setup

```
// utils/db
// .env.local
// DATABASE=mongodb://localhost:27017/ai_video

import mongoose from "mongoose";

export default async function db() {
  if (mongoose.connection.readyState >= 1) {
    return;
  }

  try {
    await mongoose.connect(process.env.DATABASE as string);
    console.log("✅ Database connection successful");
  } catch (error) {
    console.error("❌ Database connection error:", error);
  }
}

// models/video
import mongoose from "mongoose";

const VideoSchema = new mongoose.Schema(
  {
    userEmail: { type: String, required: true, index: true },
    userName: String,
    videoScript: Array,
    images: Array,
    audioUrl: String,
    captions: Array,
  },
  { timestamps: true }
);
```

```
const Video = mongoose.models.Video || mongoose.model("Video",
VideoSchema);
export default Video;
```

save to db

```
// actions/mongodb
"use server";
import Video from "@/models/video";
import db from "@/lib/db";
import { currentUser } from "@clerk/nextjs/server";

export const saveVideoToDatabase = async (data: any) => {
  try {
    await db();

    const user = await currentUser();
    const userEmail = user?.emailAddresses[0]?.emailAddress;
    const userName = user?.fullName;

    await new Video({
      ...data,
      userEmail,
      userName,
    }).save();
  } catch (error) {
    console.error("Error saving video to database:", error);
  }
};

// use in context
const handleSubmit = async () => {
  try {
    setLoading(true);

    const videoScript = await generateVideoScript();
    const images = await generateImages(videoScript);
    const audioUrl = await generateAudioFile(videoScript);
    const captions = await generateCaptionsArray(audioUrl);

    // save to db
    if (videoScript && images && audioUrl && captions) {
      console.log("videoScript => ", videoScript);
      console.log("images => ", images);
      console.log("audioUrl => ", audioUrl);
      console.log("captions => ", captions);

      setLoadingMessage("Saving to database...");
      await saveVideoToDatabase({ videoScript, images, audioUrl, captions
});
      setLoadingMessage("Video creation successful!");
```

```
    }
    setLoadingMessage("Your video is ready for preview...");
  } catch (error) {
    setLoadingMessage("An error occurred during video creation.");
    console.error("Error during video creation:", error);
  } finally {
    setLoading(false);
  }
};
```

display videos in dashboard

```
// actions/mongodb
// get all videos of the current user from the database
export const getUserVideosFromDatabase = async () => {
  try {
    await db();

    const user = await currentUser();
    const userEmail = user?.emailAddresses[0]?.emailAddress;

    const videos = await Video.find({ userEmail }).sort({ createdAt: -1
});

    return JSON.parse(JSON.stringify(videos));
  } catch (error) {
    console.error("Error getting videos from database:", error);
  }
};
```

```
// dashboard/page
// pages/dashboard.tsx
import React from "react";
import { getUserVideosFromDatabase } from "@/actions/mongodb";
import DashboardVideoPlayer from "@/components/video/dashboard-video-
player";

export default async function DashboardPage() {
  const videos: any = await getUserVideosFromDatabase();

  return (
    <div className="p-10">
      <h1 className="text-2xl font-bold mb-5">Dashboard</h1>
      <div className="grid grid-cols-3 gap-5">
        {videos.map((video: any) => (
          <div key={video._id}>
            <DashboardVideoPlayer video={video} />
          </div>
        ))}
```

```
      </div>
    </div>
  );
}

//  components/video/dashboard-video-player
"use client";

import React from "react";
import { Player } from "@remotion/player";
import RemotionVideo from "@/components/video/remotion-video";

export default function DashboardVideoPlayer({ video }: { video: any }) {
  const { images, audioUrl, captions } = video;

  const totalDuration =
    captions.length > 0
      ? Math.ceil((captions[captions.length - 1] as any).end / (1000 /
30))
      : 1;

  return (
    <Player
      component={RemotionVideo}
      durationInFrames={totalDuration}
      compositionWidth={300}
      compositionHeight={450}
      fps={30}
      inputProps={{ images, audio: audioUrl, captions }}
      controls={true}
    />
  );
}

// components/video/remotion-video
// update props
// accessed by RemotionPlayer component
export default function RemotionVideo({
  images = [],
  audio = "",
  captions = [],
}: any) {
  // const { images, audio, captions } = useVideo();
  const {
    images: videoImages,
    audio: videoAudio,
    captions: videoCaptions,
  } = useVideo();
  //
}
```

create card on dashboard

```
// dashboard/page
return (
  <div className="p-4 md:p-10">
    <div className="p-4 md:p-10">
      <h1 className="text-2xl font-bold mb-5">Dashboard</h1>

      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-
5">
        <Link href="/dashboard/create-video">
          <div className="flex flex-col items-center justify-center">
            <Button
              variant="outline"
              className="h-[450px] w-[300px] flex flex-col items-center
justify-center text-2xl"
            >
              <Image src="/logo.png" alt="upload" width={100} height={100}
/>
              <span>Create</span>
            </Button>
          </div>
        </Link>
        {videos.length > 0 &&
          videos.map((video: any) => (
            <div key={video._id} className="flex flex-col items-center">
              <DashboardVideoPlayer video={video} />
            </div>
          ))}
      </div>
    </div>
);
```

credit page and component

```
// components/nav/credits
"use client";
import React from "react";
import { useUser } from "@clerk/nextjs";
import { Coins } from "lucide-react";

export default function CreditsComponent() {
  const [total, setTotal] = React.useState(0);

  return (
    <div>
      <Coins className="h-10 w-10 text-[#6a5acd]" />
    </div>
  );
}

// import and use in top nav
```

```
// top nav
import { Bot } from "lucide-react";

{
  user && (
    <MenubarMenu>
      <Link href="/buy-credits">
        <Credits />
      </Link>
    </MenubarMenu>
  );
}

// app/buy-credits/page
// add to middleware
```

paypal context for credits

```
// npm i @paypal/react-paypal-js

// context/paypal
"use client";
import { PayPalScriptProvider } from "@paypal/react-paypal-js";

// get clientId from paypal developer sandbox
const initialOptions = {
  clientId: process.env.NEXT_PUBLIC_PAYPAL_CLIENT_ID!,
  currency: "AUD",
  locale: "en_AU", // not needed to use default US
  intent: "capture",
};

// create account at https://developer.paypal.com/developer/applications
// get clientid fro api section (sidebar menu) from paypal developer
sandbox, merchant app
export default function PayPalProvider({
  children,
}: Readonly<{ children: React.ReactNode }>) {
  return (
    <PayPalScriptProvider options={initialOptions}>
      {children}
    </PayPalScriptProvider>
  );
}
// import and use in layout
```

loader component

```
// components/loader
import React from "react";
import { Loader2Icon } from "lucide-react";

export default function Loading() {
  return (
    <div className="flex justify-center items-center h-screen">
      <div className="absolute inset-0 flex items-center justify-center
bg-opacity-50 bg-white">
        <Loader2Icon className="animate-spin" size={64} />
      </div>
    </div>
  );
}
```

buy credits page with paypal

```
// buy-credits page
"use client";
import React, { useState } from "react";
import { PayPalButtons, usePayPalScriptReducer } from "@paypal/react-
paypal-js";
import Loader from "@/components/loader";
import { Card, CardContent, CardHeader, CardTitle } from
"@/components/ui/card";
import { Button } from "@/components/ui/button"; // Assuming you have a
Button component
import toast from "react-hot-toast";

export default function CreditsPage() {
  const [{ isPending }] = usePayPalScriptReducer();
  const [selected, setSelected] = useState({ credits: 10, amount: 5.0 });

  // Define mapping of credits to dollar amount
  const creditOptions = [
    { credits: 10, amount: 5.0 },
    { credits: 20, amount: 10.0 },
    { credits: 50, amount: 20.0 },
  ];

  const handleSuccess = async (details: any) => {
    console.log(details);
  };

  const handleError = (error: any) => {
    console.error("Payment error:", error);
    toast.error("Payment failed. Please try again.");
  };

  if (isPending) {
```

```
    return <Loader />;
  }

  return (
    <div className="container mx-auto p-4">
      <Card className="w-full max-w-md mx-auto">
        <CardHeader>
          <CardTitle className="text-2xl font-bold text-center">
            Buy Credits
          </CardTitle>
        </CardHeader>

        <CardContent>
          <div className="flex flex-col gap-2 justify-between mb-6">
            {creditOptions.map((option) => (
              <Button
                key={option.credits}
                onClick={() => setSelected(option)}
                variant={
                  selected.credits === option.credits ? "default" :
"outline"
                }
                className="h-10"
              >
                {option.credits} Credits - ${option.amount.toFixed(2)}
              </Button>
            ))}
          </div>

          <div className="relative z-0">
            <PayPalButtons
              key={selected.credits} // IMPORTANT! Re-render the buttons
when the selected credits change
              createOrder={(data, actions: any) => {
                const price = selected.amount.toFixed(2); // 20
                const credits = selected.credits.toString(); // 50

                return actions.order.create({
                  purchase_units: [
                    {
                      amount: {
                        currency_code: "AUD",
                        value: price,
                      },
                      custom_id: credits,
                    },
                  ],
                });
              }}
              onApprove={async (data, actions) => {
                const details = await actions?.order?.capture();
                handleSuccess(details);
              }}
              onError={handleError}
```

```
            />
          </div>
        </CardContent>
      </Card>
    </div>
  );
}
```

credit model

```
// models/credit
import mongoose from "mongoose";

const CreditSchema = new mongoose.Schema(
  {
    userEmail: { type: String, required: true, index: true },
    credits: Number,
    amount: Number,
  },
  { timestamps: true }
);

const Credit = mongoose.models.Credit || mongoose.model("Credit",
CreditSchema);

export default Credit;
```

credits on database server action

```
// actions/credit
"use server";
import Credit from "@/models/credit";
import db from "@/utils/db";
import { currentUser } from "@clerk/nextjs/server";

export const saveCreditToDb = async (amount: number, credits: number) => {
  try {
    await db();
    const user = await currentUser();
    const userEmail = user?.emailAddresses[0]?.emailAddress;

    // Check if the user already has a credit record
    const existingCredit = await Credit.findOne({ userEmail });

    if (existingCredit) {
      // If a credit record exists, update the amount
      existingCredit.amount += amount;
      existingCredit.credits += credits;
      await existingCredit.save();
```

```
      return JSON.parse(JSON.stringify(existingCredit));
    } else {
      // If no credit record exists, create a new one
      const newCredit = new Credit({
        userEmail,
        amount,
        credits,
      });
      await newCredit.save();

      return JSON.parse(JSON.stringify(newCredit));
    }
  } catch (err: any) {
    throw new Error(err);
  }
};


// use in credits page
const handleSuccess = async (details: any) => {
  console.log(details);
  const amount = parseFloat(details.purchase_units[0].amount.value); //
Convert to number
  const credits = parseInt(details.purchase_units[0].custom_id, 10); //
Convert to number

  // save credits in db
  try {
    await saveCreditToDb(amount, credits);
    toast.success(
      `Transaction completed by ${details.payer.name.given_name}`
    );
  } catch (err) {
    toast.error("Error saving credits. Please contact support.");
  }
};
```

get credits server action

```
// actions/credit
export const getUserCreditsDb = async () => {
  try {
    await db();
    const user = await currentUser();
    const userEmail = user?.emailAddresses[0]?.emailAddress;

    const credit = await Credit.find({ userEmail });

    return JSON.parse(JSON.stringify(credit[0])); // Return the first
record
```

```
  } catch (err: any) {
    throw new Error(err);
  }
};

// use store the credits in context, so can be accessed globally
```

display credits

credits in context

```
// context/video

//
import { getUserCreditsDb } from "@/actions/credit";

// context value type
interface VideoContextType {
  //
  credits: number;
  setCredits: React.Dispatch<React.SetStateAction<number>>;
}

export const VideoProvider = ({ children }: { children: React.ReactNode })
=> {
  // state
  // ..
  const [credits, setCredits] = React.useState(0);

  // ...

  // fetch user credits
  React.useEffect(() => {
    getUserCredits();
  }, []);

  const getUserCredits = () => {
    getUserCreditsDb().then((credit) => {
      setCredits(credit.credits);
    });
  };

  return (
    <VideoContext.Provider
      value={{
        //
        credits,
        setCredits,
      }}
    >
```

```
      {children}
    </VideoContext.Provider>
  );
};

// access context in nav/credits
("use client");
import React from "react";
import { Coins } from "lucide-react";
import { useVideo } from "@/context/video";

export default function CreditsComponent() {
  const { credits } = useVideo();

  return (
    <div className="relative inline-block">
      <Coins className="h-8 w-8 text-[#6a5acd]" /> {credits}
    </div>
  );
}
```

display user credits

```
// nav/credits
"use client";
import React from "react";
import { Coins } from "lucide-react";
import { useVideo } from "@/context/video";

export default function CreditsComponent() {
  const { credits } = useVideo();

  const displayCredits = credits > 99 ? "99+" : credits.toString();
  const badgeColor = credits < 10 ? "bg-red-500" : "bg-green-500";

  return (
    <div className="flex items-center">
      <div className="relative inline-block">
        <Coins className="h-8 w-8 text-[#6a5acd]" />
        <span
          className={`
          absolute -top-1 -right-2
          ${badgeColor}
          text-white text-[10px] font-bold rounded-full
          min-w-[18px] h-[18px] flex items-center justify-center px-1
        `}
        >
          {displayCredits}
        </span>
      </div>
    </div>
```

```
  );
}

// disiplay total credits on buy-credits component
import { useVideo } from "@/context/video";

const { credits } = useVideo();

<CardHeader>
  <CardTitle className="text-3xl font-bold text-center mb-5">
    Buy Credits
  </CardTitle>
  <p className="text-center bg-green-500 p-2 rounded-md">
    You currently have <span className="font-bold text-primary">{credits}
</span>{" "}
    credits
  </p>
</CardHeader>;
```

credits on user signup

```
// actions/credit
// check if credit record exists for userEmail, if not create one
export const checkCreditRecordDb = async () => {
  try {
    await db();
    const user = await currentUser();
    const userEmail = user?.emailAddresses[0]?.emailAddress;

    const credit = await Credit.findOne({ userEmail });

    if (!credit) {
      const newCredit = new Credit({
        userEmail,
        amount: 0,
        credits: 5,
      });
      await newCredit.save();
    }
  } catch (err: any) {
    throw new Error(err);
  }
};


// use in context
// check if credit record exists for user
React.useEffect(() => {
  checkCreditRecordDb();
}, []);
```

spend credit on each video generation

```
// actions/mongodb
export const saveVideoToDatabase = async (data: any) => {
  auth().protect();
  try {
    await db();

    const user = await currentUser();
    const userEmail = user?.emailAddresses[0]?.emailAddress;
    const userName = user?.fullName;

    // 1. Check if user has enough credits
    const userCredit = await Credit.findOne({ userEmail });
    if (!userCredit || userCredit.credits < 1) {
      // throw new Error("Insufficient credits");
      return { success: false, _id: null, credits: userCredit.credits };
    }

    // Reduce 1 credit
    userCredit.credits -= 1;
    await userCredit.save();

    console.log("Saving video to database:", data);
    const { videoScript, images, audioUrl, captions } = data;
    console.log({ videoScript, images, audioUrl, captions });

    const newVideo = await new Video({
      videoScript,
      images,
      audioUrl,
      captions,
      userEmail,
      userName,
    });
    newVideo.save();

    return {
      success: true,
      credits: userCredit.credits,
    };
  } catch (error) {
    console.error("Error saving video to database:", error);
  }
};

// update handleSubmit() in context
const handleSubmit = async () => {
  try {
    setLoading(true);

    const videoScript = await generateVideoScript();
```

```
    const images = await generateImages(videoScript);
    const audioUrl = await generateAudioFile(videoScript);
    const captions = await generateCaptionsArray(audioUrl);

    // Save to database
    if (videoScript && images && audioUrl && captions) {
      setLoadingMessage("Saving to database...");
      const { success, _id, credits }: any = await saveVideoToDatabase({
        videoScript,
        images,
        audioUrl,
        captions,
      });
      setLoadingMessage("Video creation successful!");

      if (success) {
        setCredits(credits);
        toast.success("🎉 Video generated");
      } else {
        setCredits(credits);
        toast.error(
          "Insufficient credits. Please buy more credits to generate
videos"
        );
      }
    }
    setLoadingMessage("Your video is ready for preview...");
  } catch (error) {
    setLoadingMessage("An error occurred during video creation.");
    console.error("Error during video creation:", error);
  } finally {
    setLoading(false);
  }
};
```

buy credits update client

```
// buy-credits
// access from context then use

import { useImage } from "@/context/image";

const { credits, getUserCredits } = useImage();

const handleSuccess = async (details: any) => {
  // ..
  // save credits in db
  try {
    await saveCreditToDb(amount, credits);
    getUserCredits();
    toast.success(`Transaction completed by
```

```
${details.payer.name.given_name}`);
  } catch (err) {
    toast.error("Error saving credits. Please contact support.");
  }
};
```

landing page with V0

inspired by vid.ai use V0 to get starter code

```
import Hero from "@/components/landing-page/hero";
import Parallax from "@/components/landing-page/parallax";
import Features from "@/components/landing-page/features";
import Pricing from "@/components/landing-page/pricing";
import Reviews from "@/components/landing-page/reviews";
import FAQ from "@/components/landing-page/faq";
import CTA from "@/components/landing-page/cta";
import Footer from "@/components/landing-page/footer";
import HowItWorks from "@/components/landing-page/how-it-works";

export default function LandingPage() {
  return (
    <div className="min-h-screen bg-gray-900 text-gray-100">
      <Hero />

      <Parallax
        url="/dashboard.jpg"
        paddingTop="pt-5"
        paddingBottom="pb-5"
      ></Parallax>

      <Features />

      <Parallax url="/dashboard.jpg"></Parallax>

      <HowItWorks />

      <Parallax url="/create-video.jpg"></Parallax>

      <Pricing />

      <Parallax
        url="/uhd-laser.jpg"
        paddingTop="pt-5"
        paddingBottom="pb-5"
      ></Parallax>

      <Reviews />

      <Parallax
        url="/create-video.jpg"
```

```
        paddingTop="pt-20"
        paddingBottom="pb-20"
      ></Parallax>

      <FAQ />

      <Parallax
        url="/create-video.jpg"
        paddingTop="pt-5"
        paddingBottom="pb-5"
      ></Parallax>

      <CTA />

      <Footer />
    </div>
  );
}
```

hero section with animated titles and background image

```
// components/landing-page/hero
"use client";
import React from "react";
import { Button } from "@/components/ui/button";
import Link from "next/link";

export default function Hero() {
  const [titles, setTitles] = React.useState([
    "AI VIDEOS",
    "YOUTUBE SHORTS",
    "TIKTOK VIDEOS",
    "INSTAGRAM REELS",
    "SHORT VIDEOS",
  ]);

  const [currentTitleIndex, setCurrentTitleIndex] = React.useState(0);

  React.useEffect(() => {
    const intervalId = setInterval(() => {
      setCurrentTitleIndex((prevIndex) => (prevIndex + 1) %
titles.length);
    }, 1000);

    return () => clearInterval(intervalId); // Cleanup on component
unmount
  }, [titles]);

  return (
    <div className="min-h-screen bg-gray-900 text-gray-100">
      {/* Hero Section */}
```

```
      <section className="relative h-screen flex items-center justify-
center overflow-hidden">
        <div className="absolute inset-0 z-0">
          <img
            src="/uhd-laser.jpg"
            alt="AI Video Creation"
            className="w-full h-full object-cover opacity-30"
          />
          <div className="absolute inset-0 bg-gradient-to-b from-
transparent to-gray-900"></div>
        </div>
        <div className="container mx-auto px-4 text-center relative z-10">
          <h1 className="text-5xl md:text-7xl font-bold mb-6">
            <small className="animate-bounce bg-clip-text text-transparent
bg-gradient-to-r from-purple-400 to-pink-600 block">
              {titles[currentTitleIndex]}
            </small>
            <span className="bg-clip-text text-transparent bg-gradient-to-
r from-purple-400 to-pink-600">
              GENERATOR
            </span>
          </h1>
          <p className="text-xl md:text-2xl mb-8 text-gray-300 max-w-3xl
mx-auto">
            Why pay more when you can generate stunning videos for free?
          </p>

          <Link href="/dashboard/create-video">
            <Button
              type="submit"
              className="w-1/2 bg-gradient-to-r from-purple-500 to-pink-
500 hover:from-purple-600 hover:to-pink-600"
            >
              Get Started
            </Button>
          </Link>
        </div>
      </section>
    </div>
  );
}
```

parallax image

```
// components/landing-page/parallax
export default function Parallax({
  url,
  children,
  paddingTop = "pt-52", // equivalent to 200px
  paddingBottom = "pb-52", // equivalent to 200px
}: any) {
```

```
  return (
    <div className={`relative ${paddingTop} ${paddingBottom} overflow-
hidden`}>
      <div
        className="absolute inset-0 bg-cover bg-center bg-no-repeat bg-
fixed"
        style={{ backgroundImage: `url(${url})` }}
      ></div>
      <div className="absolute inset-0 bg-gradient-to-b from-transparent
to-gray-900 opacity-90"></div>
      <div className="relative z-10">{children}</div>
    </div>
  );
}
```

features section

```
// components/landing-page/features
import React from "react";
import { Button } from "@/components/ui/button";

const features = [
  "Prompt to Video",
  "Video Script Generator",
  "Script to Video",
  "AI Voices",
  "AI Text to Speech",
  "AI Audio to Text",
  "Short Videos",
  "AI Image Generator",
  "AI Animated Image Generator",
  "AI Art Image Generator",
  "AI Fantasy Image Generator",
  "AI Audio Subtitle Generator",
];

export default function Features() {
  return (
    <section className="py-20 bg-gradient-to-b from-gray-900 to-gray-800">
      <div className="container mx-auto px-4">
        <h2 className="text-4xl md:text-5xl font-bold mb-12 text-center
bg-clip-text text-transparent bg-gradient-to-r from-purple-400 to-pink-
600">
          Powerful AI Features
        </h2>

        <div className="flex flex-wrap gap-4 p-4 justify-center">
          {features.map((feature, index) => (
            <Button
              key={index}
              variant="outline"
```

```
              className="h-auto py-3 px-4 flex items-center justify-start
whitespace-nowrap font-sans font-bold rounded-full transition-all
duration-300 ease-in-out hover:bg-gradient-to-r hover:from-purple-400
hover:to-pink-600 hover:text-white"
            >
              <span className="text-sm">{feature}</span>
            </Button>
          ))}
        </div>
      </div>
    </section>
  );
}
```

how it works section with image icons

```
// components/landing-page/how-it-works
import React from "react";

const works = [
  {
    image: "/icons/ideas.png",
    title: "1. Script Idea",
    description: "Start with your video concept or choose one",
  },
  {
    image: "/icons/generation.png",
    title: "2. AI Generation",
    description: "Our AI creates script, visuals, and audio",
  },
  {
    image: "/icons/preview.png",
    title: "3. Preview",
    description: "Review and tweak your AI-generated video",
  },
  {
    image: "/icons/publish.png",
    title: "4. Publish",
    description: "Share your video to your favorite platforms",
  },
];

export default function HowItWorks() {
  return (
    <section className="py-20 bg-gradient-to-b from-gray-800 to-gray-900">
      <div className="container mx-auto px-4">
        <h2 className="text-4xl md:text-5xl font-bold mb-12 text-center
bg-clip-text text-transparent bg-gradient-to-r from-purple-400 to-pink-
600">
          How It Works
        </h2>
```

```
        <div className="grid md:grid-cols-4 gap-8">
          {works.map((step, index) => (
            <div key={index} className="flex flex-col items-center text-
center">
              <GradientIcon imageSrc={step.image} />
              <h3 className="text-xl font-semibold mb-2">{step.title}</h3>
              <p className="text-gray-400">{step.description}</p>
            </div>
          ))}
        </div>
      </div>
    </section>
  );
}

const GradientIcon = ({ imageSrc }: { imageSrc: string }) => (
  <div className="relative w-16 h-16 mb-4">
    <img
      src={imageSrc}
      alt="Step icon"
      className="w-full h-full object-contain"
    />
  </div>
);
```

pricing cards

```
// components/landing-page/pricing
import React from "react";
import { Check } from "lucide-react";
import Link from "next/link";
import { Button } from "@/components/ui/button";

const pricePlans = [
  { name: "Starter", credits: 10, price: 5.0 },
  { name: "Pro", credits: 20, price: 10.0, popular: true },
  { name: "Enterprise", credits: 50, price: 20.0 },
];

export default function Pricing() {
  return (
    <section className="py-20 bg-gradient-to-b from-gray-900 to-gray-800">
      <div className="container mx-auto px-4">
        <h2 className="text-4xl md:text-5xl font-bold mb-12 text-center
bg-clip-text text-transparent bg-gradient-to-r from-purple-400 to-pink-
600">
          Simple Pricing
        </h2>
        <div className="grid md:grid-cols-3 gap-8">
          {pricePlans.map((plan, index) => (
            <PriceCard index={index} plan={plan} />
```

```
          ))}
        </div>
      </div>
    </section>
  );
}

const PriceCard = ({ index, plan }: any) => (
  <div
    key={index}
    className={`bg-gray-800 p-6 rounded-lg border ${
      plan.popular ? "border-purple-500" : "border-gray-700"
    } hover:border-purple-500 transition-colors`}
  >
    {plan.popular && (
      <div className="text-purple-400 text-sm mb-2">Most Popular</div>
    )}
    <h3 className="text-2xl font-bold mb-4">{plan.name}</h3>
    <div className="text-4xl font-bold mb-4 bg-clip-text text-transparent
bg-gradient-to-r from-purple-400 to-pink-600">
      ${plan.price}
      <span className="text-xl text-gray-400">/mo</span>
    </div>
    <ul className="mb-6 space-y-2">
      <li className="flex items-center">
        <Check className="w-5 h-5 mr-2 text-green-500" />
        {plan.credits} AI-generated videos
      </li>
      <li className="flex items-center">
        <Check className="w-5 h-5 mr-2 text-green-500" />
        Access to all features
      </li>
      <li className="flex items-center">
        <Check className="w-5 h-5 mr-2 text-green-500" />
        24/7 support
      </li>
    </ul>
    <Link href="/buy-credits">
      <Button className="w-full bg-gradient-to-r from-purple-500 to-pink-
500 hover:from-purple-600 hover:to-pink-600">
        {plan.popular ? "Get Started" : "Choose Plan"}
      </Button>
    </Link>
  </div>
);
```

reviews section

```
// components/landing-page/reviews
import React from "react";
```

```
const reviews = [
  {
    name: "Alex Johnson",
    role: "Content Creator",
    quote:
      "AiVid has revolutionized my content creation process. I can now
produce high-quality shorts in minutes!",
  },
  {
    name: "Sarah Lee",
    role: "Marketing Manager",
    quote:
      "The AI-generated scripts and visuals are incredibly creative. It's
like having a whole production team at my fingertips.",
  },
  {
    name: "Mike Brown",
    role: "Small Business Owner",
    quote:
      "Affordable and easy to use. AiVid has helped me boost my social
media presence without breaking the bank.",
  },
];

export default function Reviews() {
  return (
    <section className="py-20 bg-gradient-to-b from-gray-800 to-gray-900">
      <div className="container mx-auto px-4">
        <h2 className="text-4xl md:text-5xl font-bold mb-12 text-center
bg-clip-text text-transparent bg-gradient-to-r from-purple-400 to-pink-
600">
          What Our Users Say
        </h2>
        <div className="grid md:grid-cols-3 gap-8">
          {reviews.map((testimonial, index) => (
            <div
              key={index}
              className="bg-gray-800 p-6 rounded-lg border border-gray-700
hover:border-purple-500 transition-colors"
            >
              <p className="mb-4 italic text-gray-300">"
{testimonial.quote}"</p>
              <div className="font-semibold">{testimonial.name}</div>
              <div className="text-gray-400">{testimonial.role}</div>
            </div>
          ))}
        </div>
      </div>
    </section>
  );
}
```

faq section with accordians

```
// components/landing-page/faq
import React from "react";
import {
  Accordion,
  AccordionContent,
  AccordionItem,
  AccordionTrigger,
} from "@/components/ui/accordion";

const faq = [
  {
    question: "How does the credit system work?",
    answer:
      "Each video generation costs a 1 credit. You can purchase credit
packs anytime or top-up to your account.",
  },
  {
    question: "Can I decide what the AI will generate?",
    answer:
      "Yes, you can choose from the existing template options or you can
enter your own custom prompts for generating video scripts, images and
subtitles.",
  },
  {
    question: "What can these videos be used for?",
    answer:
      "These AI generated short-form video formats can be used to publish
videos with platforms like TikTok, Instagram Reels and YouTube Shorts.",
  },
  {
    question: "Is there a free trial available?",
    answer:
      "Yes, new users can sign up and start generating AI videos for free.
Free credits will apply to your account as soon as you signin. No credit
card required.",
  },
];

export default function FAQ() {
  return (
    <section className="py-20 bg-gradient-to-b from-gray-900 to-gray-800">
      <div className="container mx-auto px-4">
        <h2 className="text-4xl md:text-5xl font-bold mb-12 text-center
bg-clip-text text-transparent bg-gradient-to-r from-purple-400 to-pink-
600">
          Frequently Asked Questions
        </h2>
        <Accordion type="single" collapsible className="max-w-2xl mx-
auto">
          {faq.map((faq, index) => (
```

```
            <AccordionItem
              key={index}
              value={`item-${index}`}
              className="border-b border-gray-700"
            >
              <AccordionTrigger className="text-left">
                {faq.question}
              </AccordionTrigger>
              <AccordionContent className="text-gray-400">
                {faq.answer}
              </AccordionContent>
            </AccordionItem>
          ))}
        </Accordion>
      </div>
    </section>
  );
}
```

call to action section

```
// components/landing-page/cta
import React from "react";
import Link from "next/link";
import { ChevronRight } from "lucide-react";
import { Button } from "@/components/ui/button";

export default function CTA() {
  return (
    <section className="py-20 bg-gradient-to-b from-gray-800 to-gray-900">
      <div className="container mx-auto px-4 text-center">
        <h2 className="text-4xl md:text-5xl font-bold mb-6 bg-clip-text
text-transparent bg-gradient-to-r from-purple-400 to-pink-600">
          Ready to Create Amazing Videos?
        </h2>
        <p className="text-xl mb-8 text-gray-300 max-w-2xl mx-auto">
          Join thousands of content creators who are already using AiVid
to
          produce stunning short-form videos with AI.
        </p>
        <Link href="/dashboard/create-video">
          <Button className="bg-gradient-to-r from-purple-500 to-pink-500
hover:from-purple-600 hover:to-pink-600 text-lg px-8 py-3">
            Get Started Now <ChevronRight className="ml-2" />
          </Button>
        </Link>
      </div>
    </section>
  );
}
```

footer

```
// components/landing-page/footer
import React from "react";
import { Copyright } from "lucide-react";

export default function Footer() {
  return (
    <footer className="bg-gray-900 py-12">
      <div className="container mx-auto px-4">
        <div className="mt-8 pt-8 border-t border-gray-800 text-center
text-gray-400">
          <div className="flex justify-center gap-2">
            <Copyright /> {new Date().getFullYear()} AiVid. All rights
reserved.
          </div>
        </div>
      </div>
    </footer>
  );
}
```

deploy to vercel

if video generation fails due to timeout error:

- you need to upgrade the vercel plan - https://vercel.com/guides/what-can-i-do-about-vercel-serverless-functions-timing-out
- or you can deploy to digital ocean or similar services - REFERAL/DISCOUNT LINK $200 credit for 60 days: https://m.do.co/c/90857e82d426

🎉 contrats! You did it!