Spooky 🎃

Hello Ghouls and Goblins! This is my submission to the ACM Halloween Dev Contest.

What on Earth is Spooky?!

Have you ever been in the mood for a fun horror story? Ever needed to break the ice at a Halloween party? Feeling **Spooky** 👻 ?

Spooky has your back!

Spooky is a web application that uses OpenAI text and voice generation to create a fun, short scary story in 4 sentences, or your money back! Pass in any idea or image, and let Spooky handle the rest.

Examples:

> Cat in a mech suit.
> In a town where cats roamed free, one curious feline stumbled upon a shiny mech suit abandoned in a junkyard. Thrilled by the chance to play, she jumped inside, her tiny paws deftly pressing buttons, turning her into a towering terror. The townsfolk watched in awe and dread as she meowed, "I've discovered the purr-fect way to rule the night!" With a playful flick of her tail, she sent the moonlit shadows dancing, leaving the villagers laughing and shrieking in delight.

What's with the Pomodoro Timer?

I wanted to find a way to have spooky lofi music in the app, so why not have a little study session and make it 👻 **SPOOKY** 👻 ?

Let's Take a Deep Dive into How This Was Made!

Tech Stack

- **Next.js**: A React framework that is feature-rich and the most popular frontend framework for developing web applications with TypeScript.
- **Tailwind CSS**: A CSS framework that provides predefined classes with a utility-first approach.

- **AWS S3**: A cloud storage service from Amazon Web Services (AWS) that allows users to store and retrieve data from anywhere.
- **OpenAI SDK**: For making API calls and creating content with the power of AI 🤖 .

Tools

- **V0**: Quickly generate UI with AI to get a prototype up in no time!
- **Vercel**: Vercel allows us to host our web applications in just a few clicks.

Application Breakdown

Main Page (page.tsx)

On this page, we keep track of the two main components of the application: the Story Generator and the Pomodoro Timer.

```
const [showTimer, setShowTimer] = useState(false);
const [showStoryGenerator, setShowStoryGenerator] = useState(false);
```

When clicking the "Tell me a scary story" button, we hide the Pomodoro if it's visible and set `showStoryGenerator` to true.

```
<button
  onClick={() => {
    setShowStoryGenerator(!showStoryGenerator);
    setShowTimer(false);
  }}
  className="bg-orange-500 hover:bg-orange-600 text-white px-4 py-2 rounded"
>
  Tell me a scary story
</button>
```

We use the same logic for the "Study with me" button:

```
<button
  onClick={() => {
    setShowTimer(!showTimer);
    setShowStoryGenerator(false);
  }}
  className="bg-orange-500 hover:bg-orange-600 text-white px-4 py-2 rounded"
>
  Study with me
</button>
```

When `showStoryGenerator` is true, the `&&` expression evaluates to render the `<div>` and its contents. When `showStoryGenerator` is false, the `&&` expression evaluates to false, so React renders nothing.

```
{
  showStoryGenerator && (
    <div className="mb-8 w-full max-w-md">
      <StoryGenerator />
    </div>
  );
}
```

The same logic applies to the Pomodoro section, but the difference is that we've added an extra piece for the music player.

```
{
  showTimer && (
    <div className="mb-8 w-full max-w-md">
      <PomodoroTimer />
      <div className="mt-8 bg-gray-800 p-4 rounded-lg text-center">
        <h3 className="text-xl mb-4 text-white">Spooky Study Music</h3>
        <iframe
          width="100%"
          height="300"
          allow="autoplay"
          src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/playli
          className="rounded-lg"
        ></iframe>
        <div className="text-xs text-gray-400 mt-2 break-words">
          <a
            href="https://soundcloud.com/user-203264501"
            title="chaoticlove"
            target="_blank"
            className="text-gray-400 hover:text-gray-300"
          >
            chaoticlove
          </a>{" "}
          ·{" "}
          <a
            href="https://soundcloud.com/user-203264501/sets/spooky-halloween-lofi"
            title="Spooky Halloween Lofi"
            target="_blank"
            className="text-gray-400 hover:text-gray-300"
          >
            Spooky Halloween Lofi
          </a>
        </div>
      </div>
    </div>
  );
}
```

Components

The main component we will review is `StoryGenerator.tsx`. This is a form that sends a POST request to our API handler to generate spooky content.

`api/spooky-story/route.ts`

`route.ts` is a server function that handles API requests for server-side operations. You would want to use this type of API route for:

- Database interactions
- Handling sensitive data that shouldn't be exposed to clients

Here’s how we use it:

We create an S3 object to store audio files to return to users for playback. Originally, I wanted to use blob storage on the client, but S3 made it easier to deliver the audio files to the page. Once we have the object, we create an upload function to send the MP3 files we generate from OpenAI.

```
// object creation
const s3 = new AWS.S3({
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
  region: process.env.AWS_REGION,
});

// S3 upload functionality
async function uploadToS3(buffer: Buffer, key: string): Promise<string> {
    const params = {
      Bucket: process.env.S3_BUCKET_NAME!,
      Key: key,
      Body: buffer,
      ContentType: "audio/mpeg",
    };

    await s3.upload(params).promise();
    return `https://${process.env.S3_BUCKET_NAME}.s3.${process.env.AWS_REGION}.amazonaw
  }
```

The `encodeImageToBase64` function takes an image file as input, converts it to a base64-encoded string, and returns that string. You have to do this before you send a local file to the API endpoint.

```
const encodeImageToBase64 = async (image: File): Promise<string> => {
  const buffer = Buffer.from(await image.arrayBuffer());
  return buffer.toString("base64");
};
```

Here are the functions that call OpenAI and return AI-generated content. They are similar, with only slight differences in their input and output.

```
// creates a story based on the prompt we pass in the text input
async function generateStoryFromText(storyIdea: string): Promise<string | null> {
  const openAI = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
  });

  const response = await openAI.chat.completions.create({
    model: "gpt-4o-mini",
    messages: [
      {
        role: "user",
        content: `I want you to create a 4-sentence short scary story that is whimsical
      },
    ],
  });

  return response.choices[0].message.content || null;
}

// creates a story based on the image we pass in from the form
async function generateStoryFromImage(base64Img: string): Promise<string | null> {
  const openAI = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
  });

  const response = await openAI.chat.completions.create({
    model: "gpt-4o-mini",
    messages: [
      {
        role: "user",
        content: [
          {
            type: "text",
            text: `Take these key items from the image: the person, the environment, an
          },
          {
            type: "image_url",
            image_url: {
              url: `data:image/jpeg;base64,${base64Img}`,
            },
          },
        ],
      },
    ],
  });

  return response.choices[0].message.content || null;
}

// generates an mp3 file from the story text
async function generateVoice(story: string): Promise<Buffer> {
  const openAI = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY,
  });

  const mp3 = await openAI.audio.speech.create({
    model: "tts-1",
    voice: "fable",
    input: story,
  });

  return Buffer.from(await mp3.arrayBuffer());
```

POST Function

In this code, the `POST` function is

an API route handler that processes incoming POST requests to generate a story (based on text input or an image), converts it to audio, and stores the audio in AWS S3. This is broken down into 5 steps:

```
export async function POST(request: Request) {
// Step 1 – Extract data from the request
  const formData = await request.formData();
  const storyIdea = formData.get("storyIdea") as string;
  const image = formData.get("image") as File | null;

  try {
    // Step 2 – Process the information and generate a scary story from the text or ima
    let story: string | null;

    if (image) {
      const base64Img = await encodeImageToBase64(image);
      story = await generateStoryFromImage(base64Img);
    } else if (storyIdea) {
      story = await generateStoryFromText(storyIdea);
    } else {
      throw new Error("No story idea or image provided");
    }

    if (!story) throw new Error("Story generation failed");

    // Step 3 – Generate audio from the story text
    const audioBuffer = await generateVoice(story);

    // Step 4 – Upload the audio file to S3
    const audioKey = `spooky_story_${Date.now()}.mp3`;
    const audioUrl = await uploadToS3(audioBuffer, audioKey);

    // Step 5 – Return the response to be used in the StoryGenerator component
    return NextResponse.json({ story, audioUrl });
  } catch (error) {
    console.error("Error in SpookyStoryGenerator:", error);
    return NextResponse.json({ error: "Failed to generate story and audio" }, { status:
  }
}
```

Thank you for reading, and Happy Halloween! 🎃 👻