On April 2, 2020 one of my favorite sites, dev.to, announced a Twilio hackathon. The link is found here\(^1\). In this hackathon, participants had to make anything with the Twilio API and submit it before April 30, 2020. I decided to make a simple video app with Gatsby and the Twilio API. One of the main categories of this hackathon was Covid-19 communications, so it will be simple enough for anyone to use, including elderly people.

I got help from a YouTube tutorial in order to build this\(^2\) web app. You need a Twilio account for this project. We will go through the whole process, from setting up the account to writing Twilio functions for the video chat app, in this chapter.

The Setup

In a new folder called SimpleVideoApp, create a new Gatsby project with the default starter, using the gatsby new command. The commands are shown in Listing 7-1.

**Listing 7-1.** Creating a New Gatsby Project

```bash
mkdir SimpleVideoApp
cd SimpleVideoApp
gatsby new .
```

Once the installation is done, it’s time to install twilio and twilio-video. Use the command in Listing 7-2 from the terminal to do this.

---

\(^1\)https://dev.to/devteam/announcing-the-twilio-hackathon-on-dev-2lh8

\(^2\)https://www.youtube.com/embed/K02SnxY6c_0

© Nabendu Biswas 2021

N. Biswas, *Foundation Gatsby Projects*, https://doi.org/10.1007/978-1-4842-6558-1_7
Listing 7-2. Twilio Install

`npm i twilio`

`npm i twilio-video`

It’s time to create a Twilio account.

Creating a Twilio Account

Go to [https://www.twilio.com/try-twilio](https://www.twilio.com/try-twilio) to create an account. Enter the details on this page and click the Start Your Free Trial button (see Figure 7-1).

![Figure 7-1. Twilio account](image)

You will get the usual verification email on the next page (see Figure 7-2).
Upon checking my mailbox, I found the mail. Click the Confirm Your Email link to continue (see Figure 7-3).

**Figure 7-2. Verification email**

Upon checking my mailbox, I found the mail. Click the Confirm Your Email link to continue (see Figure 7-3).
After that, you have to do mobile number verification. Provide a valid mobile number and click the check box; then click the Verify button (see Figure 7-4).

**Figure 7-3. Confirm your email**

After that, you have to do mobile number verification. Provide a valid mobile number and click the check box; then click the Verify button (see Figure 7-4).
We will get a verification code on our mobile phone, which you need to enter here and then click Submit (see Figure 7-5).
Figure 7-5. Verification code

After that, you’ll see the screen in Figure 7-6, in which you have to state whether you code. I chose Yes.
In the next screen we have to choose the framework. We need to choose NodeJS here (see Figure 7-7).

**Figure 7-6. Choose yes when asked if you write code**

In the next screen we have to choose the framework. We need to choose NodeJS here (see Figure 7-7).
Next, click the Skip to Dashboard button, as shown in Figure 7-8.

**Figure 7-7. Choose NodeJS**
This will take you to the dashboard (see Figure 7-9).

**Figure 7-8. Skip to the dashboard**

This will take you to the dashboard (see Figure 7-9).
Working in the Dashboard

There are a lot of settings in the dashboard, including creating a Twilio function. We logged into the Twilio dashboard earlier. Click the three dots on the left menu, as shown in Figure 7-10.
Next, click Programmable Video from the menu (see Figure 7-11).

**Figure 7-10. Click the three dots**

Next, click Programmable Video from the menu (see Figure 7-11).
On the next screen, click the Show API Credentials link (see Figure 7-12).

**Figure 7-11. Choose Programmable Video**

On the next screen, click the Show API Credentials link (see Figure 7-12).
Figure 7-12. Click Show API Credentials

On the next page, note your Account SID and Auth Token (see Figure 7-13).
It’s time to put these secrets in an .env file. Create an .env file in the root directory and put the account SID and auth token into the TWILIO_ACC_SID and TWILIO_AUTH_TOKEN variables, respectively. The content is shown in Listing 7-3.

Listing 7-3. The Environment File

```
TWILIO_ACC_SID=XXXXXXXXXXXX
TWILIO_AUTH_TOKEN=XXXXXXXXXX
```

Then choose Tools from the left menu (see Figure 7-14).
After that, click the API Keys submenu link. Next, we have to click the Create New API Key button (see Figure 7-15).
On the next screen, we have to give the new API key a name and then click the Create API Key button (see Figure 7-16).

**Figure 7-15. Create a new API key**

On the next screen, we have to give the new API key a name and then click the Create API Key button (see Figure 7-16).
On the next screen, we will be shown the SID and the SECRET. We need to note them both (see Figure 7-17).

**Figure 7-16. Provide a name here**

On the next screen, we will be shown the SID and the SECRET. We need to note them both (see Figure 7-17).
Figure 7-17. *Note the SID and secret*

Then click the check box and the Done button (see Figure 7-18).
On the next screen, the secret won’t be visible. This means our API keys have been saved (see Figure 7-19).

**Figure 7-18.** Check the box to indicate you saved the information

On the next screen, the secret won’t be visible. This means our API keys have been saved (see Figure 7-19).
Again, go to the .env file and save the two variables in Listing 7-4.

**Listing 7-4.** Environment File with Additional Keys

```
TWILIO_ACC_SID=XXXXXXXXXXXXX
TWILIO_AUTH_TOKEN=XXXXXXXXXXX
TWILIO_API_SID=XXXXXXXXXXXXX
TWILIO_API_SECRET=XXXXXXXXXXXX
```

The settings of the API keys took a lot of time, so we will start with Twilio functions in the next section.

**Creating Twilio Functions**

We will finally start creating Twilio functions. First, click the three dots on the left side of the Twilio dashboard (see Figure 7-20).

*Figure 7-19. The API keys have been saved*
Figure 7-20. The Twilio dashboard

Next, if you scroll down a bit, you will find the Functions menu; click it (see Figure 7-21).
Figure 7-21. Functions

On the next screen, click the Create a Function button (see Figure 7-22).
A popup will appear in which you need to click Blank and then click the Create button (see Figure 7-23).
On the next page, we need to give our function a name and a path. I named them Create Token and /create-token, respectively (see Figure 7-24).

Figure 7-23. Create a blank function

On the next page, we need to give our function a name and a path. I named them Create Token and /create-token, respectively (see Figure 7-24).
Next, we need to remove the check box and then remove everything inside the function (see Figure 7-25).

**Figure 7-24. Create a token**
Next, we will write some code in the function. This code will use built-in Twilio variables and get a new access token from our stored variables. The code is shown in Listing 7-5.

**Listing 7-5. Twilio Functions**

```javascript
exports.handler = function(context, event, callback) {
  let accessToken = Twilio.jwt.AccessToken;
  let videoGrant = accessToken.VideoGrant;
  let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);
};
```

After writing these three lines of code, click the Save button. After that, click the Configure link in the left menu. It will open the page shown in Figure 7-26.

**Figure 7-25. Remove everything inside the function**

Next, we will write some code in the function. This code will use built-in Twilio variables and get a new access token from our stored variables. The code is shown in Listing 7-5.

**Listing 7-5. Twilio Functions**

```javascript
exports.handler = function(context, event, callback) {
  let accessToken = Twilio.jwt.AccessToken;
  let videoGrant = accessToken.VideoGrant;
  let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);
};
```

After writing these three lines of code, click the Save button. After that, click the Configure link in the left menu. It will open the page shown in Figure 7-26.
We need to check the Enable ACCOUNT_SID and AUTH_TOKEN check box. After that, click the + button next to KEY twice and enter API_KEY and API_SECRET in the Key column.

You get their values from TWILIO_API_SID and TWILIO_API_SECRET, which will be saved in the previous .env file (see Figure 7-27).

**Figure 7-26. Configuration page**

We need to check the Enable ACCOUNT_SID and AUTH_TOKEN check box. After that, click the + button next to KEY twice and enter API_KEY and API_SECRET in the Key column.

You get their values from TWILIO_API_SID and TWILIO_API_SECRET, which will be saved in the previous .env file (see Figure 7-27).
Figure 7-27. *Enable the API keys*

After clicking the Save button, click the Manage link (see Figure 7-28).
We need to click the function name (such as Create Token) to go to the edit page. Next, we add the four lines in Figure 7-29 to the function. We are doing a callback with the token with JWT.

**Figure 7-28. Choose the Manage option**

We need to click the function name (such as Create Token) to go to the edit page. Next, we add the four lines in Figure 7-29 to the function. We are doing a callback with the token with JWT.
Adding the Code

We will do some coding now, but first start the project by moving to the directory and running the `gatsby develop` command.

Basic Setup

The command to start the server is shown in Listing 7-6.

Listing 7-6. Server Start Command

cd SimpleVideoApp
gatsby develop

If we go to `http://localhost:8000/`, we will get the default page (see Figure 7-30).

Figure 7-29. Configuring the function

Adding the Code

We will do some coding now, but first start the project by moving to the directory and running the `gatsby develop` command.

Basic Setup

The command to start the server is shown in Listing 7-6.

Listing 7-6. Server Start Command

cd SimpleVideoApp
gatsby develop

If we go to `http://localhost:8000/`, we will get the default page (see Figure 7-30).
Time to change the default starter, so open the `index.js` file and add the code in Listing 7-7.

**Listing 7-7. The index.js File**

```javascript
import React from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"

const IndexPage = () => {
  return (
    <Layout>
      <SEO title="Home" />
    </Layout>
  )
}

export default IndexPage
```

**Figure 7-30. Default starter**

Time to change the default starter, so open the `index.js` file and add the code in Listing 7-7.
We will remove all the unnecessary things. Remove the page-2.js file, as we don’t need it.

It’s also time to change some things in this Gatsby starter. Open the layout.js file in the components folder and change the footer text. The updated code is marked in bold in Listing 7-8.

Listing 7-8. The layout.js File

```jsx
... ...
return (
  <Header siteTitle={data.site.siteMetadata.title} />
  <div
    style={{
      margin: `0 auto`,
      maxWidth: 960,
      padding: `0 1.0875rem 1.45rem`,
    }}
  >
    <main>{children}</main>
    <footer>
      Copyright © <a href="https://thewebdev.tech">SimpleVideoApp</a>,
      {new Date().getFullYear()},
      All rights reserved
    </footer>
  </div>
)
...
```

Also, let’s change the site’s title. Head over to the gatsby-config.js file and change it. The updated code is marked in bold in Listing 7-9.
**Listing 7-9.** Site Metadata Change in gatsby-config.js

```javascript
module.exports = {
  siteMetadata: {
    title: `Simple Video App`,
    description: `A simple video app, created using gatsby and twilio, for twilio hackathon on dev.to`,
    author: `Nabendu Biswas`,
  },
  plugins: [
    ...
  ],
}
```

When we go to http://localhost:8000/,\(^4\) we will see the updated app (see Figure 7-31).

![Simple Video App](image)

Copyright © SimpleVideoApp, 2020. All rights reserved

**Figure 7-31.** Updated app

### Create a Login Form

Let’s create a simple functional login form. Create a file called `login-form.js` inside the components folder and put the code from Listing 7-10 in it. It is a simple form with a text field that allows you to enter the name and a button to submit the form.

\(^4\)http://localhost:8000/
Listing 7-10. The login-form.js File

```javascript
import React, { useState } from "react"

const LoginForm = () => {
  const [name, setName] = useState(""

  return (
    <form>
      <label htmlFor="name">
        Display Name: <br />
        <input
          type="text"
          id="name"
          name="name"
          value={name}
          onChange={e => setName(e.target.value)}
        />
      </label>
      <br />
      <button type="submit">Join Video Chat</button>
    </form>
  )
}

export default LoginForm
```

Next, let’s show this component in the index.js file. Import it and use it. The updated code is marked in bold in Listing 7-11.

Listing 7-11. LoginForm in index.js

```javascript
import React from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"
import LoginForm from "../components/login-form"

const IndexPage = () => {
  return (党内
    <form>
      <label htmlFor="name">
        Display Name: <br />
        <input
          type="text"
          id="name"
          name="name"
          value={name}
          onChange={e => setName(e.target.value)}
        />
      </label>
      <br />
      <button type="submit">Join Video Chat</button>
    </form>
  )
}
```

382
<Layout>
  <SEO title="Home" />
  <LoginForm />
</Layout>

export default IndexPage

The web app now shows our not-so-beautiful login form (see Figure 7-32).

![Simple video app](image)

**Figure 7-32. Simple login form**

Let's style this web app a bit, as I like web apps to look nice. First update all the primary styles in `layout.css`. Remove all the content and replace it with Listing 7-12.

**Listing 7-12. The New layout.css File**

```css
@import url("https://fonts.googleapis.com/css?family=Quicksand&display=swap");

* {
  box-sizing: border-box;
  margin: 0;
}

:root {
  --primaryColor:#243e36;
  --mainGrey: #F9F9FA;
  --mainWhite: #fff;
  --mainBlack: #0A0A0A;
  --darkGrey: #8e8e8e;
```

383
Next, let’s update the `header.js` file to use a new color scheme. I had updated the background and also removed the line `margin: 0 auto` from `maxWidth: 960`. The updated code is marked in bold in Listing 7-13.

**Listing 7-13. The Updated header.js File**

```javascript
import { Link } from "gatsby"
import PropTypes from "prop-types"
import React from "react"

const Header = ({ siteTitle }) => {
  return (
    <header
      style={{
        background: "#243e36",
        marginBottom: "1.45rem",
        color: var(--mainBlack),
        font-size: 18px,
        overflow-x: hidden;
      }}
    >
      
    </header>
  );
}
```

384
Next, create a file called `login.module.css` in the `components` folder and put the code in Listing 7-14 into it.

**Listing 7-14.** The `login.module.css` File

```css
.contact {
  padding: 4rem 0;
}
.center {
  width: 80vw;
  margin: 0 auto;
}
@media screen and (min-width: 992px) {
  .center {
    width: 50vw;
    margin: 0 auto;
  }
}
```
Update the login-form.js file to include these styles. The updated code is marked in bold in Listing 7-15.

**Listing 7-15.** Styles in login-form.js

```javascript
import React, { useState } from "react"
import styles from ".login.module.css"

const LoginForm = () => {
  const [name, setName] = useState(""

  return (  
    <section className={styles.contact}>
      <h3>Login</h3>
      <div className={styles.center}>
        <form>
          <div>
            <label htmlFor="name">Display Name</label>
            <input  
              type="text"  
              id="name"  
              name="name"  
              value={name}  
              className={styles.formControl}  
              onChange={e => setName(e.target.value)}
            />
          </div>
          <button type="submit" className={styles.submit}>Join Video Chat</button>
        </form>
      </div>
    </section>
  )
}

export default LoginForm
```

The web app now looks perfect, as shown in Figure 7-33.
Connect the App to Twilio

We need to do an API call to the Twilio endpoint, which contains our function. For this we will install Axios first.

Stop your gatsby develop and install Axios using the npm i axios command. Don’t forget to restart your development server by re-running the gatsby develop command.

We will now use Axios to send the form data to our Twilio endpoint. Open the login-form.js file and create a form called onSubmit. It will call a function called handleSubmit and use the Twilio endpoint URL, and the data will be the name. The updated code is marked in bold in Listing 7-16.

Listing 7-16. Axios in login-form.js

```javascript
import React, { useState } from "react"
import styles from ".//login.module.css"
import axios from "axios"

const LoginForm = () => {
  const [name, setName] = useState(""
```
### Chapter 7  Creating a Video Chat Site

```javascript
const handleSubmit = async event => {
  event.preventDefault()
  const result = await axios({
    method: "POST",
    url: "https://tan-cat-7689.twil.io/create-token",
    data: {
      identity: name,
    },
  })
  console.log(result);
}

return (
  <section className={styles.contact}>
    <h3>Login</h3>
    <div className={styles.center}>
      <form onSubmit={handleSubmit}>
        <div>
          ...
          ...
        </div>
        <button type="submit" className={styles.submit}>Join Video Chat</button>
      </form>
    </div>
  </section>
)

export default LoginForm
```

We get the Twilio endpoint in the code from the function page in Twilio (see Figure 7-34).
We also updated the code as per Listing 7-17.

**Listing 7-17. Updated Twilio Function**

```javascript
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY, process.env.API_SECRET);
    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);
    callback(null, token.JWT);
};
```

Go back to the web app and open the console. After that, provide a name and log in. We are getting a CORS error and are not able to log in (see Figure 7-35).
There is a very good way to check the actual error from inside the Twilio dashboard. Click the bug icon on the top-right side. After that, click the Go to the Debugger link (see Figure 7-36).

Figure 7-35. Logging in

Figure 7-36. You can check an error from the Twilio dashboard
The screen in Figure 7-37 will appear and will show the error. We need to click the highlighted error in this screen.

![Screenshot of Debugger Events]

**Figure 7-37. Click the highlighted error**

It will show us the real error (see Figure 7-38).

---

392
After reviewing my function again, I realized that the four lines had not been saved from earlier. I added those lines again and clicked the Save button (see Figure 7-39).

**Figure 7-38. The real error**

After reviewing my function again, I realized that the four lines had not been saved from earlier. I added those lines again and clicked the Save button (see Figure 7-39).
After logging in again, I get the same error. By watching the YouTube video, I realize that we need to use the function as in the Twilio docs link (see Figure 7-40).

Figure 7-39. Added lines, added again

After logging in again, I get the same error. By watching the YouTube video, I realize that we need to use the function as in the Twilio docs link (see Figure 7-40).

---

5https://www.youtube.com/watch?v=K02SnxY6c_0
6https://www.twilio.com/docs/runtime/functions/faq?code-sample=code-set-multiple-http-headers-in-a-response-5&codelanguage=Node.js&code-sdk-version=default
Updating your function accordingly. Also, don’t forget to click the Save button. The updated code is marked in bold in Listing 7-18.

Listing 7-18. Twilio Function Updated Again

```javascript
exports.handler = function(context, event, callback) {
    let accessToken = Twilio.jwt.AccessToken;
    let videoGrant = accessToken.VideoGrant;
    let token = new accessToken(process.env.ACCOUNT_SID, process.env.API_KEY,
                              process.env.API_SECRET);
    token.identity = event.identity;
    const grant = new videoGrant();
    token.addGrant(grant);

    let response = new Twilio.Response();

    // Build list of headers
    let headers = {
        "Access-Control-Allow-Origin": "*",
        "Access-Control-Allow-Methods": "GET,PUT,POST",
    }
    // Set headers in response
    response.set(headers);
    callback(null, response);
}
```

Figure 7-40. Twilio docs
"Access-Control-Allow-Headers": "Content-Type"
};

// Set headers in response
response.setHeaders(headers);
response.setBody(JSON.stringify(token.toJwt()));
callback(null, response);

When we submit again, we will get the JWT back successfully (see Figure 7-41).

Figure 7-41. JWT back

We can also check the validity of it by going to the site at https://jwt.io/ and pasting the returned JWT there. And, yes, it is valid (see Figure 7-42).
Implementing the Video

It’s time to store the result so that we can use it in the next section. Update the login-form.js file with a props storeToken and then save the JWT in it. The updated code is marked in bold in Listing 7-19.

Listing 7-19. JWT in login-form.js

```javascript
const LoginForm = ({ storeToken }) => {
  const [name, setName] = useState(""

  const handleSubmit = async event => {
    event.preventDefault()
    const result = await axios({
      method: "POST",
      url: "https://tan-cat-7689.twil.io/create-token",
    })
  }
```

Figure 7-42. Valid JWT
data: {
    identity: name,
},
})
console.log(result);
const jwt = result.data;
storeToken(jwt);
}

Next, we will update index.js with a new state token and pass the props storeToken in LoginForm.

We are using ternary logic to display the form if the user is not logged in or for the time being has token text. The updated code is marked in bold in Listing 7-20.

**Listing 7-20.** storeToken in index.js

import React, {useState} from "react"
import Layout from "../components/layout"
import SEO from "../components/seo"
import LoginForm from "../components/login-form";

const IndexPage = () => {
    const [token, setToken] = useState(false);

    return (
        <Layout>
            <SEO title="Home" />
            {!token ? <LoginForm storeToken={setToken} /> : <p>Has Token</p>}
        </Layout>
    )
}

export default IndexPage

It’s time to test our code in http://localhost:8000/. Upon opening it, we get the login form (see Figure 7-43).

---

8http://localhost:8000/
When we provide a name and click Join Video Chat, we are taken to the screen in Figure 7-44.

Create the Video Component

Next, we will create the Video component. Create a new file called video.js inside the components folder. We are using a useEffect hook, which will fire when the token changes. The token comes as a prop and is used to connect to a room, using the twilio-video built-in method. The code is shown in Listing 7-21.
CHAPTER 7  CREATING A VIDEO CHAT SITE

Listing 7-21. The video.js File

```javascript
import React, { useEffect } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
    useEffect(() => {
        TwilioVideo.connect(token, { video: true, audio: true, name: "SVA" }).then(
            result => {
                console.log("Successfully joined room", result)
            }
        ), [token])
    return (  
        <div>
            Video
        </div>
    )
}

export default Video;
```

Next, let's use this component in `index.js` when the user is authenticated. The updated code is shown in Listing 7-22.

Listing 7-22. Video in index.js

```javascript
import React, {useState} from "react"
import Layout from "./components/layout"
import SEO from "./components/seo"
import LoginForm from "./components/login-form";
import Video from "./components/video";

const IndexPage = () => {
    const [token, setToken] = useState(false);
```
We will again log in to our web app, by giving a display name and clicking the Join Video Chat button (see Figure 7-45).

![Simple Video App](image)

**Figure 7-45. Log in**

This will open a popup to ask for permission to use video and audio. We need to click Allow (see Figure 7-46).
Figure 7-46. Choose Allow to continue

It will successfully log you in and show Video (see Figure 7-47).

Figure 7-47. Video
We will now start showing the local webcam video. Open the video.js file and update the highlighted parts. Here, we are using useRef, as we want to attach the video to the div.

We are using a built-in createLocalVideoTrack() variable from Twilio to attach the video of the webcam to the div localVidRef. The updated code is shown in Listing 7-23.

**Listing 7-23.** Local Video in video.js

```jsx
import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
  const localVidRef = useRef()

  useEffect(() => {
    TwilioVideo.connect(token, { video: true, audio: true, name: "SVA" }).then(
      result => {
        TwilioVideo.createLocalVideoTrack().then(track => {
          localVidRef.current.appendChild(track.attach())
        })
      }
    )
  }, [token])

  return (
    <div>
      <div ref={localVidRef} />
    </div>
  )
}

export default Video;
```

Go to http://localhost:8000/ again and log in with any username (see Figure 7-48).
After that, allow the browser to use the camera and microphone (see Figure 7-49).

**Figure 7-48.** Log in again

After that, allow the browser to use the camera and microphone (see Figure 7-49).

**Figure 7-49.** Allow the browser to use the camera and microphone

You will be able to see yourself in the video (see Figure 7-50).
It’s time to attach the remote participants. Update the video.js file as shown in Listing 7-24. We are adding another ref `remoteVidRef` to a div.

We then take the result and loop through each participant. Each participant can have one or more tracks (cameras), so we are looping through them as well. We then append it to `remoteVidRef`. The updated code is shown in Listing 7-24.

**Listing 7-24. Remote Video in video.js**

```javascript
import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
  const localVidRef = useRef()
  const remoteVidRef = useRef()

```

**Figure 7-50. The video is working**

It’s time to attach the remote participants. Update the video.js file as shown in Listing 7-24. We are adding another ref `remoteVidRef` to a div.

We then take the result and loop through each participant. Each participant can have one or more tracks (cameras), so we are looping through them as well. We then append it to `remoteVidRef`. The updated code is shown in Listing 7-24.

**Listing 7-24. Remote Video in video.js**

```javascript
import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"

const Video = ({ token }) => {
  const localVidRef = useRef()
  const remoteVidRef = useRef()

```
useEffect(() => {
    TwilioVideo.connect(token, { video: true, audio: true, name: "SVA" }).then(result => {
        TwilioVideo.createLocalVideoTrack().then(track => {
            localVidRef.current.appendChild(track.attach())
        })

        const addParticipant = participant => {
            participant.tracks.forEach(publication => {
                if (publication.isSubscribed) {
                    const track = publication.track
                    remoteVidRef.current.appendChild(track.attach())
                }
            })
        }

        result.participants.forEach(addParticipant)
    }), [token])

    return (
        <div>
            <div ref={localVidRef} />
            <div ref={remoteVidRef} />
        </div>
    }
}

export default Video;

It seems to be working, as we are able to see two videos now (see Figure 7-51).
Deploying Netlify

It’s time to deploy to Netlify, so that you can test from two devices. Open your Netlify dashboard and click the New Site from Git button (see Figure 7-52).
Next, click GitHub (see Figure 7-53), as my code is in https://github.com/nabendu82/SimpleVideoApp.

Figure 7-52. Netlify

Next, click GitHub (see Figure 7-53), as my code is in https://github.com/nabendu82/SimpleVideoApp.\textsuperscript{10}
After that, I need to search the repo, as I have a lot of them. After getting the correct repo, click it (see Figure 7-54).
After that, keep all the default settings and click the Deploy Site button (see Figure 7-55).

**Figure 7-54. Searching for the correct repo**

After that, keep all the default settings and click the Deploy Site button (see Figure 7-55).

**Figure 7-55. Deploy the site**
On the next screen, click the Site Settings button quickly (see Figure 7-56).

Figure 7-56. Site settings

Next, scroll a bit and click the Change Site Name button (see Figure 7-57).
Figure 7-57. Change the site’s name

It will open a popup, in which you can change the random site name to something meaningful (see Figure 7-58).
Finally, the site is deployed (see Figure 7-59).

**Figure 7-58.** Choose a meaningful name for your site

Finally, the site is deployed (see Figure 7-59).

**Figure 7-59.** The site is deployed
The app is deployed and I logged in from two machines, but I was not able to see both videos. Per the YouTube video, I realized I missed a part. We need to add the updated part to the video.js file. The updated code is shown in Listing 7-25.

Listing 7-25. Fixes to the video.js File

```javascript
const Video = ({ token }) => {
  const localVidRef = useRef()
  const remoteVidRef = useRef()

  useEffect(() => {
    TwilioVideo.connect(token, { video: true, audio: true, name: "SVA" }).then(
      result => {
        TwilioVideo.createLocalVideoTrack().then(track => {
          localVidRef.current.appendChild(track.attach())
        })
      }
    )

    const addParticipant = participant => {
      participant.tracks.forEach(publication => {
        if (publication.isSubscribed) {
          const track = publication.track
          remoteVidRef.current.appendChild(track.attach())
        }
      })
      participant.on("trackSubscribed", track => {
        remoteVidRef.current.appendChild(track.attach())
      })
    }
    result.participants.forEach(addParticipant)
    result.on("participantConnected", addParticipant)
  }, [token])

  return {
    component: <div ref={localVidRef} />
  }
}
```

Chapter 7  Creating a Video Chat Site
I then push the code, which automatically deploys to Netlify. I am able to log in from multiple devices and see the videos (see Figure 7-60).

![Figure 7-60. Working!](image)

There is some CSS and some other changes remaining, before I can submit my app to the hackathon.

### Making CSS Changes

Our web app is almost finished; only some CSS remains, so let’s add it.

I also want to show the name of the organizer, so let’s get it from `login-form.js`. We will use a callback function called `storeName`, which is similar to `storeToken`, to send the name back. The updated code is shown in Listing 7-26.
Listing 7-26. storeName in login-form.js

... 
...
const LoginForm = ({ storeToken, storeName }) => {
  const [name, setName] = useState(""

  const handleSubmit = async event => {
    event.preventDefault()
    const result = await axios({
      method: "POST",
      url: "https://tan-cat-7689.twil.io/create-token",
      data: {
        identity: name,
      },
    })
    console.log(result);
    const jwt = result.data;
    storeToken(jwt);
    storeName(name);
  }

  ...
  ...

  Next, let’s update index.js and use the logic similar to that of token. We are passing the name to the Video component. The updated code is shown in Listing 7-27.

Listing 7-27. storeName in index.js

... 
...
const IndexPage = () => {
  const [token, setToken] = useState(false);
  const [name, setName] = useState(false);
return (  
  <Layout>  
    <SEO title="Home" />  
    {!token ? <LoginForm storeToken={setToken} storeName={setName} /> : <Video token={token} name={name} />}  
  </Layout>  
)  
export default IndexPage

Next, let's update the Video component in the video.js file. Here, we are importing a video.css file. We will make it soon. After that, we are destructuring the name prop.

After that, in the video.js file, make the following changes. Change the enclosing div to a fragment and add Organizer and use the name prop.

Also add an h2 for remote participants. Lastly, add a className for remoteVideRef div, which we are going to style next. The updated code is shown in Listing 7-28.

Listing 7-28. Styles in video.js

import React, { useEffect, useRef } from 'react'
import TwilioVideo from "twilio-video"
import './video.css'

const Video = ({ token, name }) => {
  ...
  ...
  return (  
    <h2>Organizer: {name}</h2>
    <div ref={localVidRef} />
    <h2>Remote Participants</h2>
    <div className="remoteVideo" ref={remoteVidRef} />
  )
}
export default Video;

Add a video.css file to the same folder and add the simple style shown in Listing 7-29.
Listing 7-29. The video.css File

```css
.remoteVideo{
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
    grid-gap: 10px;
    justify-items: center;
}

video{
    width: 100%;
    max-width: 240px
}

h2 {
    margin: 1rem 0;
}
```

All the changes are done, so it's time to test it in localhost. It is working fine and I checked it in three different browsers (see Figure 7-61).

![Simple Video App](image)

**Figure 7-61.** Tested in the localhost
Automatic Deployment

It’s time to push the code in GitHub, to deploy it automatically. After it was deployed, I logged in from three different devices (see Figure 7-62).

It is working fine but I found a bug where, for remote participants, it will show their name in place of the word Organizer. Let’s change the way it will look for each user. I am now showing the name in Connect, so it will be different for each user. The updated code is shown in Listing 7-30.

Listing 7-30. Bug Fix in video.js

```javascript
return (  
  <h2>Organizer</h2>  
  <div ref={localVidRef} />  
  <h2>Remote Participants</h2>  
  <div className="remoteVideo" ref={remoteVidRef} />  
  <p>Connected : {name}</p>  
)
```

Figure 7-62. Deployed

It is working fine but I found a bug where, for remote participants, it will show their name in place of the word Organizer. Let’s change the way it will look for each user. I am now showing the name in Connect, so it will be different for each user. The updated code is shown in Listing 7-30.
Let’s add the code in Listing 7-31 to the video.css file.

Listing 7-31. The video.css File

```css
p {
    margin-top: 1rem
}
```

It is now working fine for different users (see Figures 7-63 and 7-64).

![Simple Video App](image)

**Figure 7-63. Organizer**
I will redeploy now and check the web app. I am testing with all three devices—two laptops and one phone (see Figure 7-65).
My app is complete and deployed. As I mentioned earlier, this is a very simple video app, which can be created with ease and deployed in no time with Netlify.

Share the link with your friends and enjoy video conferencing. We get $15 worth of free credits from Twilio and I used $3.24 while making this project (see Figure 7-66).

**Figure 7-65. Working fine**
I am submitting the web app soon for the hackathon, so I had to add README and LICENSE files. I added both in my GitHub and joined the Twilio CodeExchange Community. Details about the submission rules are found here. You can also find the code for this project at this GitHub link.

**Summary**

I hope you liked the video chat app we created in this chapter. You can use it to create your own app. We covered the following topics in this chapter:

- Creating a video chat web app, using the awesome Twilio service
- Setting up the Twilio site
- Writing Twilio functions for the video chat app

---

11https://ahoy.twilio.com/code-exchange-community  
12https://dev.to/devteam/announcing-the-twilio-hackathon-on-dev-2lh8  
13https://github.com/nabendu82/SimpleVideoApp