CAREERFOUNDRY

Achievement 4: Verifying Your Meet App on Google

Completely Verifying Your Meet App

In the section “Verifying Your App,” you took the first steps towards verifying your Meet app; however, to completely verify your app (and get that warning removed when it’s opened), you’ll need to undergo a more robust verification process with a few more steps. This process can take three to five days and doesn’t affect your app in any way—you can still develop it and follow along with the course content. In fact, you don’t actually need to complete this verification for your app to work! The main reason you’d want to complete the verification is if you want to use this app in your portfolio, as the warning sign that pops up if you don’t could look unprofessional to potential employers

Let's walk through the steps now so you know what to do if you want to complete the verification!

1. First, add this HTML file to the “public” folder in your Meet app and run npm run deploy. This is a simple privacy policy to include in your app. You’re free to style if however you want to fit your app’s aesthetic.

2. Next, you’ll need to create a welcome screen for your app. This step is a little more involved. Create a new component file "WelcomeScreen.jsx" in "src" folder, then add the following code to it (Make sure to replace YOUR_GITHUB_USERNAME with your github username):

```
import React from "react";
import './WelcomeScreen.css';

function WelcomeScreen(props) {
  return props.showWelcomeScreen ?
    (
      <div className="WelcomeScreen">
        <h1>Welcome to the Meet app</h1>
        <h4>
          Log in to see upcoming events around the world for
          full-stack
          developers
      </h4>
        <div className="button_cont" align="center">
          <div class="google-btn">
            <div class="google-icon-wrapper">
              <img
                class="google-icon"
src="https://upload.wikimedia.org/wikipedia/commons/5/53/Google_%22G%22_Logo.svg"
                alt="Google sign-in"
              />
            </div>
            <button onClick={() => { props.getAccessToken() }}
              rel="nofollow noopener"
              class="btn-text"
            >
              <b>Sign in with google</b>
            </button>
```

```
          </div>
        </div>
        <a
          href="https://YOUR_GITHUB_USERNAME.github.io/meet/privacy.html"
          rel="nofollow noopener"
        >
          Privacy policy
      </a>
      </div>
    )
    : null
}

export default WelcomeScreen;
```

3. Create a new CSS file called “WelcomeScreen.css” in your “src” folder and add the following code to it:

```
.WelcomeScreen {
  position: fixed;
  width: 100%;
  height: 100%;
  top: 0;
  background: white;
}

.login-button {
  color: #494949;
  text-transform: uppercase;
  text-decoration: none;
  background: #ffffff;
  border: 4px solid #494949;
  display: inline-block;
  transition: all 0.4s ease 0s;
  min-width: 10rem;
  margin: 1rem;
}

.google-btn {
  width: 184px;
  height: 42px;
  margin: 1rem;
  background-color: #4285f4;
  border-radius: 2px;
  box-shadow: 0 3px 4px 0 rgba(0, 0, 0, 0.25);
}

.google-btn:hover {
  box-shadow: 0 0 6px #4285f4;
}
```

```
.google-btn:active {
  background: #1669f2;
}

.google-icon-wrapper {
  position: absolute;
  margin-top: 1px;
  margin-left: 1px;
  width: 40px;
  height: 40px;
  border-radius: 2px;
  background-color: #fff;
}

.google-icon {
  position: absolute;
  margin-top: 11px;
  margin-left: -8px;
  width: 18px;
  height: 18px;
}

.btn-text {
  float: right;
  color: #fff;
  font-size: 14px;
  letter-spacing: 0.2px;
  font-family: "Roboto";
  background-color: #4285f4;
  width: 80%;
  height: 100%;
  text-align: center;
}
```

4. Open the “src/api.js” file and make sure that checkToken() function is exported. There should be an export keyword like this:

```
export const checkToken = async (accessToken) => {…};
```

5. Next, import your new “WelcomeScreen.jsx” component into your “src/App.js” file. Also, import checkToken() and getAccessToken() from the “api.js” file:

```
import WelcomeScreen from './WelcomeScreen';
import { getEvents, extractLocations, checkToken, getAccessToken } from
'./api';
```

6. Add a new state, showWelcomeScreen, to the “src/App.js” file and set its default value to undefined. This state will be used as a flag to determine when to render the welcome screen as follows: true will mean “show the welcome screen,” false will mean “hide it to show the other components,” and undefined will be used to render an empty div until the state gets either true or false:

```
state = {
  events: [],
  locations: [],
  showWelcomeScreen: undefined
}
```

7. Add a conditional that renders an empty div element. Make this line the first one in the render():

```
render() {
   if (this.state.showWelcomeScreen === undefined) return <div
className="App" />

  return (
    ...
  );
}
```

8. Render the <WelcomeScreen .../> component in “App.js” at the end before the closing of the tag. You’ll be adding two props to the component. The first is assigned with the showWelcomeScreen state, and it’ll be used in the “WelcomeScreen.jsx” file to determine whether the Welcome Screen is going to be rendered or not. As for the second prop, it will be a function prop that calls getTokenAccess(). This function prop will be called when the “Continue with Google” button in the Welcome Screen component is clicked (check step 2 to see its UI code). The component must be at the bottom of all the components rendered in the “App.js” file so that it gets displayed on top of all the UI, for example:

```
render() {
   if (this.state.showWelcomeScreen === undefined) return <div
className="App" />

  return (
    <div className="App">
      {/* Other components such as CitySearch, EventList,...etc */}
      <WelcomeScreen showWelcomeScreen={this.state.showWelcomeScreen}
getAccessToken={() => { getAccessToken() }} />
    </div>
  );
}
```

9. Update componentDidMount() with the following code:

```
async componentDidMount() {
  this.mounted = true;
  const accessToken = localStorage.getItem('access_token');
  const isTokenValid = (await checkToken(accessToken)).error ? false :
true;
  const searchParams = new URLSearchParams(window.location.search);
```

```
  const code = searchParams.get("code");
  this.setState({ showWelcomeScreen: !(code || isTokenValid) });
  if ((code || isTokenValid) && this.mounted) {
    getEvents().then((events) => {
      if (this.mounted) {
        this.setState({ events, locations: extractLocations(events) });
      }
    });
  }
}
```

There are a few things happening here. First, you're trying to get the token from localStorage (it goes to localStorage when the getToken() function in "src/api.js" is called). Then, you're trying to verify it using another function from "src/api.js"—checkToken()—hence why you needed to export it from there. If there's an error in the object returned by checkToken(), the variable isTokenValid will be assigned with the value false; otherwise, it will be true.

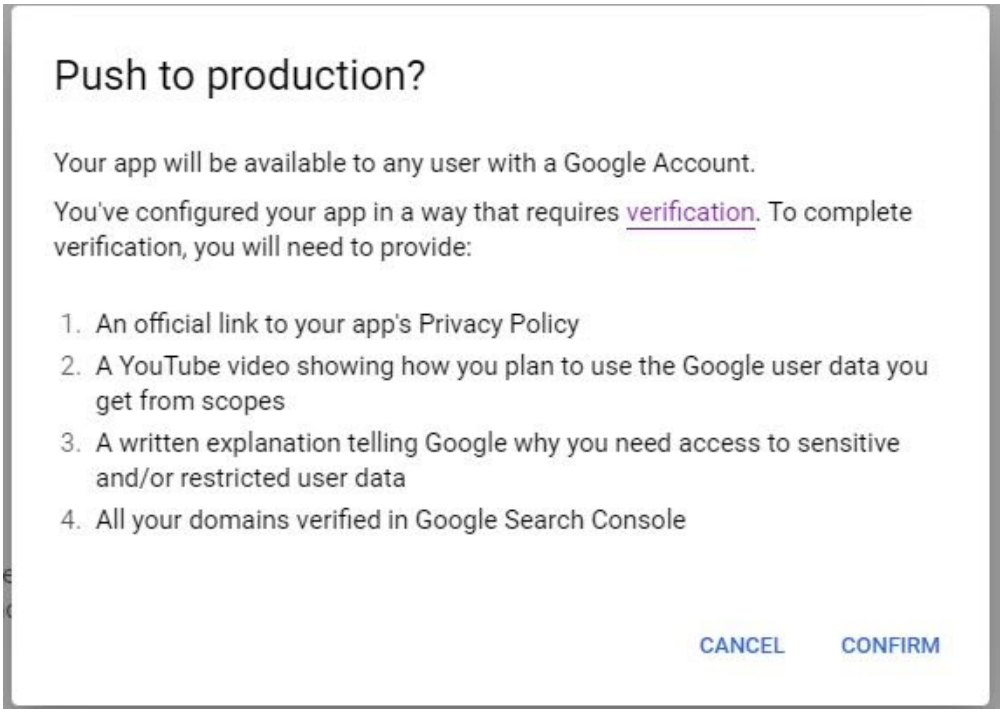
If there's no access token in the localStorage, or if the token in there isn't valid, users can still get access to the list of events by getting a new authorization code once they try to log in. This code will eventually be used to get a new access token after getEvents() is executed. This is why you'll find that the code is trying to extract the code query from the URL that appears when the user logs in and gets redirected back to your application. This means the application will be re-launched, only with the code parameter in the URL search field after your site's domain. Subsequently, this will lead to a new "mount" for the App component, resulting in componentDidMount being called once again.

10. Finally, run npm run deploy to ensure everything is working. Feel free to style this page and make it look a bit nicer. This a great opportunity to refresh your CSS learnings! You can also style your privacy page if you want. Moreover, don't forget to push your code changes to your GitHub repository's main branch

Oauth Consent Screen Settings

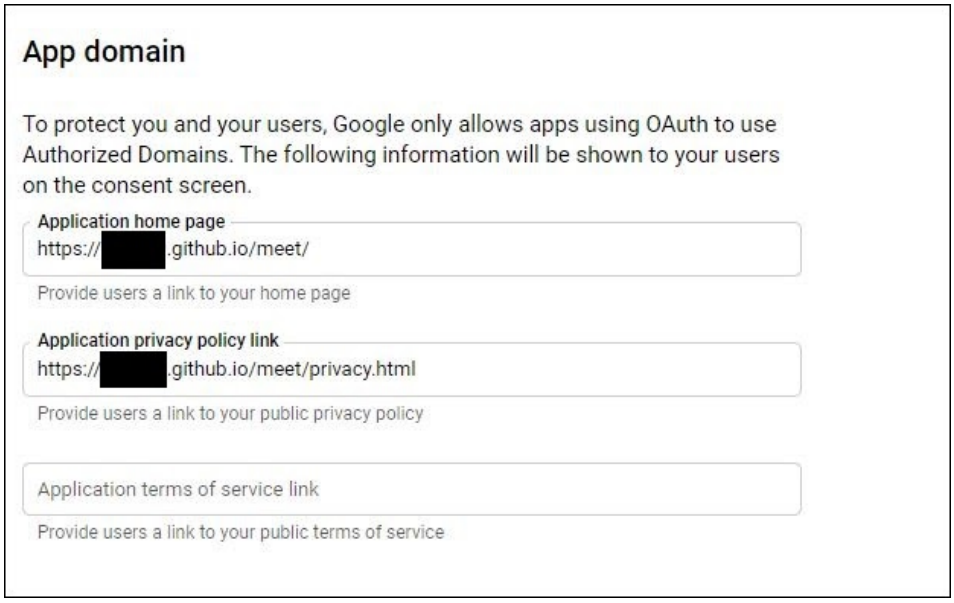
That's it on the coding side. Now, visit the [Oauth Consent Screen Settings](https://console.cloud.google.com/apis/credentials/consent) and follow these steps to submit a verification request.

1. Click the **PUBLISH APP** button you find at the top of the page under the **Publishing Status** section. A popup will appear with a set of instructions to follow in order to verify your app. Click **Confirm**.



2. You'll notice that the OAuth consent screen page will change a bit—there will be a new **PREPARE FOR VERIFICATION** button. Go ahead and click it. The next page should contain information you entered previously when first creating the project.

3. Fill in the fields below, then click **Save and Continue:**
    a. **App Logo:** Add a logo, making sure that it matches any specified requirements written below the **App Logo** input field.
    b. **App Domain:** Add a link to your privacy page in the **Application Privacy Policy Link** input field. It should look like this (replace [github_username] with your own username): "https://[github_username].github.io/meet/privacy.html"



4. In the **Scopes**, scroll down to the **Your sensitive scopes** section. You'll find the scope you added earlier, along with a question asking "How will the scopes be used?" In this box, add the following:

   The Meet app is using https://www.googleapis.com/auth/calendar.events.readonly for educational purposes only. The calendar accessed isn't the user's but a calendar provided by CareerFoundry as part of their Full-Stack Immersion course. No user information is saved or used within the application, and personal calendars aren't accessed.

5. At the end of the screen, under **Demo Video**, you need to submit a video that satisfies the following conditions:
   a. The video must be publicly accessible on YouTube.
   b. It must show all details on the consent screen once you go through the login process. Don't crop anything related to the consent/login screen.
   c. It must show all the screens in your application. You can, for example, enter something in the search field or show how the calendar events will be filtered once you submit a search query.
   d. Talking aloud as you walk through the process in the video in clear and simple English is also recommended.

6. Once you hit **Save and Continue**, you'll be taken to the **Optional Info** screen. You can keep everything as is or fill out more information about you and your application (which can help in securing your application's verification). Then, click **Save and Continue** for a final preview. If everything looks good, confirm the verification preparation request.