Angad Srivastav Follow

May 23, 2021 · 4 min read

React Google Login and Node.js implementation

Social login is a great way to get new users started on your platform as it takes away all the hassle of filling up those tedious signup forms. In this blog, I will show you how you can implement google social login for your react application the right way via backend token verification. I am going to assume that you have already set up your react frontend and node.js back-end for this tutorial. So without any further ado let's get started.

For this tutorial, you will need the following packages

- Backend
  * Google Auth Library (https://www.npmjs.com/package/google-auth-library)
- Frontend

On your front end firstly we will import all the packages that we will require for this project.

```
import React, { useState, useEffect } from "react";
import { GoogleLogin } from "react-google-login";
import axios from "axios";
import Confetti from "react-confetti";
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import "./App.css";
```

After importing all the packages first, we will focus on the GoogleLogin package. Google Login package allows us to get an OAuth token which will be used in our backend to verify the legitimacy of social login. For using this plugin you will require a ***client-id*** which you can get from your google console developer account. For this tutorial, I will assume that you already have a client-id. The code of your Google Login Package should look something like this.

```
<div>
  <GoogleLogin
    clientId="INSERT_YOUR_CLIENT_ID_HERE"
    buttonText="Login"
    onSuccess={googleResponse}
    onFailure={onFailure}
  />
</div>
```

This package requires two functions. One function will run when it will be able to successfully get a client-id. The other will run in case of any errors. In case of success, we will get an OAuth Token which we will send to our API for verification and login. The function of success will look something like this.

```
// If successfull return of data from google we run this function:
const googleResponse = async (response) => {      You, 4 months ago • React Google Social Login
  // Check if a token was recieved and send it to our API:
  if (response.tokenId) {
    const googleResponse = await axios.post(
      "http://localhost:3001/api/v1/user-auth",
      { token: response.tokenId }
    );
    // Check if we have some result:
    if (Object.keys(googleResponse.data.payload).length !== 0) {
      /*
        Get the following user details from our API and set them in the state:
        User Account Name
        User Email
        User Profile Picture for Google
      */
      const { name, email, picture } = googleResponse.data.payload;
      setState({
        ...state,
        name,
        email,
        picture,
        profile_loaded: true,
        confetti: true,
      });
      // Show a toast to the user letting them know that thelogin was successfull:
      toast.success("You have logged into your google account!", {
        position: "top-right",
        autoClose: 5000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: false,
        draggable: true,
        progress: undefined,
      });
    }
  }
};
```

The steps for success are as follows:

1. First, we will check if we have received a token
2. Once we receive a token we will send it to our API
3. Our API will provide us with the user details like name, email, and even picture
4. Once successfully logged in we will show users a toast notifying them that the login attempt was a success.

Also just to make it even better we will throw confetti on the screen for that celebration effect. So our final frontend code would look something like this.

```
  return (
    <div className="app">
      {/* The Toast container */}
      <ToastContainer
        position="top-right"
        autoClose={5000}
        hideProgressBar={false}
        newestOnTop={false}
        closeOnClick
        rtl={false}
        pauseOnFocusLoss
        draggable
        pauseOnHover={false}
      />
      {/* Show login button when user not logged in */}
      {!state.profile_loaded ? (
        <div>
          <GoogleLogin
            clientId="INSERT_YOUR_CLIENT_ID_HERE"
            buttonText="Login"
            onSuccess={googleResponse}
            onFailure={onFailure}
          />
        </div>
      ) : (
        // Show User details when logged in:
        <div className="user-details">
          {state.confetti ? (
            // Confetti Component:
            <Confetti width={window.innerWidth} height={window.innerHeight} />
          ) : null}
          <img
            src={state.picture}
            alt="profilePicture"
            className="profile-picture"
          />
          <h3>{state.name}</h3>
          <h3>{state.email}</h3>      You, 4 months ago • React Google Social Login
        </div>
      )}
    </div>
  );
};

export default App;
```

Our render code logic.

```
// This will turn off the confetti raining down on the screen after 5 seconds of successfull login:
useEffect(() => {
  setTimeout(() => {
    setState({
      ...state,
      confetti: false,
    });
  }, 5000);
```

```
// States for the component:
const [state, setState] = useState({
  name: "",
  email: "",      You, 4 months ago • React Google Social Login
  picture: "",
  profile_loaded: false,
  confetti: false,
});

// On Failur of google login we get the reason for failur in an alert:
const onFailure = (error) => {
  alert(error);
};
```

Our state logic and failure code of google login.

This was the implementation of our front end. Now let's take a look at our backend.

On your backend first of all import the library that we previously installed.

```
const { OAuth2Client } = require("google-auth-library");
```

Once we have imported the library we will now need to create a client instance from this. To create a client instance we will once again need our client-id which we can get from console developers google.

```
// Client ID for Google Login:
const client = new OAuth2Client(
  "INSERT_YOUR_CLIENT_ID_HERE"
);
```

After that, we will create our endpoint which will handle our login logic. Our login logic will constitute of the following steps

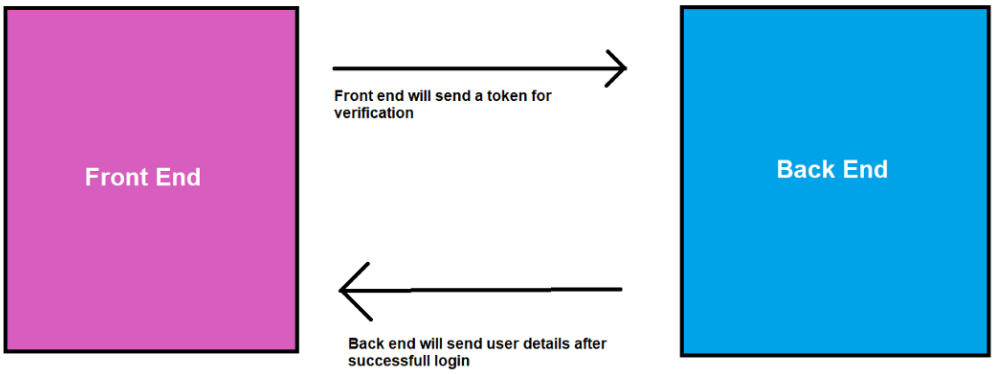
1. First, get the token from the body of the request.

4. Finally, send the data back to the front end.

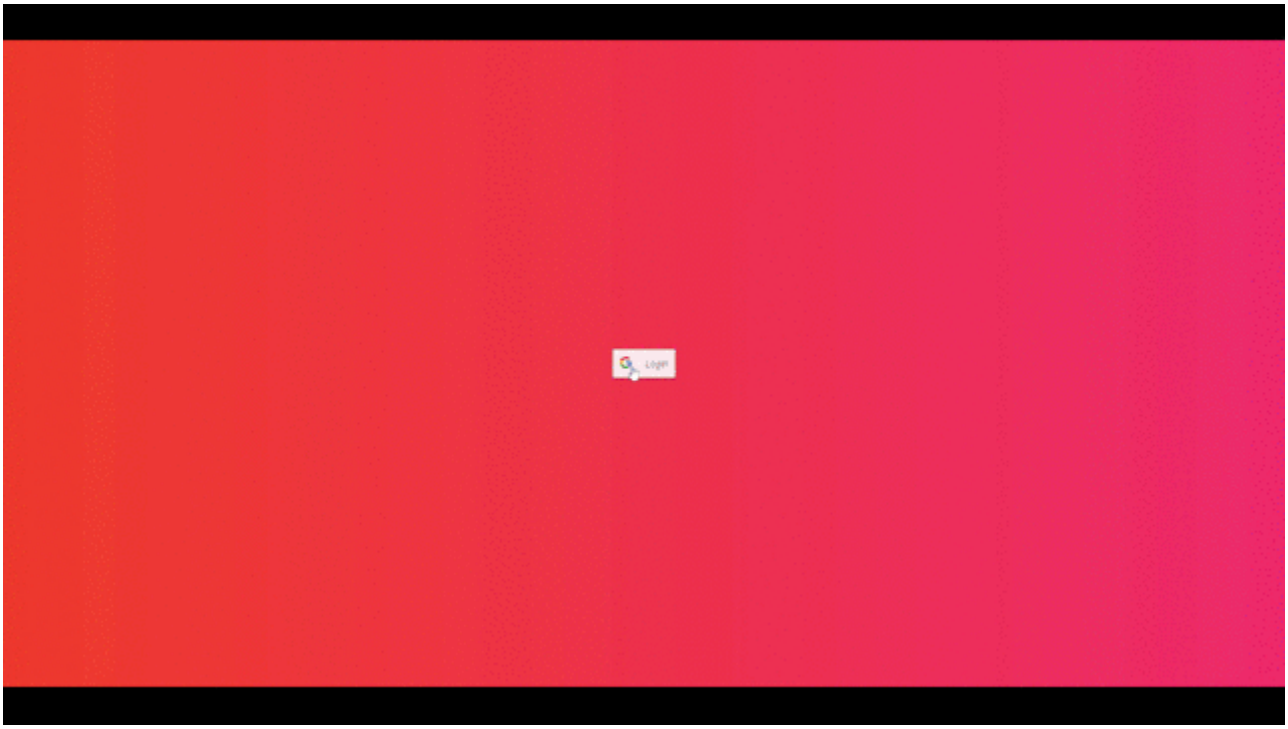
In this tutorial I have not performed any DB operations as DB operations on logic can vary from platform to platform. I have simply retrieved the data to show you how you can get legitimate login data. Our final code should look something like this.

```
// Get the token from the body of the request:
const { token } = req.body;
// Verify the token and then get User details from the payload if verified:
try {      You, 4 months ago • React Google Social Login
  const ticket = await client.verifyIdToken({
    idToken: token,
    audience:
      "INSERT_YOUR_CLIENT_ID_HERE",
  });
  const payload = ticket.getPayload();

  /*******
  ......
  Perform Database insertion or
  retrieval operations here ...
  ......
  *******/

  // Send the payload
  res.send({
    payload,
    isSuccess: true,
  });
} catch (error) {
  console.log(error);
  // If error then send an empty payload:
  res.send({
    payload: {},
    isSuccess: false,
  });
}
```

So our final interaction between front-end and back-end would be as follows.



And our social login should work as follows:



So this is one of the ways on how you can implement social login in your react applications. The link for the code can be found in my git repo.