CSC 321 Mastery Extension Assignment

Topic: CSRF Tokens

Joey Marenin

December 1, 2022

GitHub: https://github.com/jmarenin/CSC321_Mastery

Table of Contents

Premise

Out of all the topics in security, why did I choose to direct my efforts towards CSRF tokens? Throughout my time as a computer scientist, I have realized that my true passion is software engineering. Ever since I developed my first web application in my intro to computer science class, I realized that web application development was something that I wanted to pursue as a career, so web-based security topics have always been more interesting to me than others.

For even more context, I was able to get a software engineering internship with IBM the summer after taking CSC307, and pretty much the entire summer I was working on developing a full stack web application from scratch. One part of this web application involved IoT devices sending temperature and humidity data to the backend to be processed, and I encountered a lot of issues because the backend was rejecting all the device requests. At the time, I had no background in security, and did not understand the backend error messages: "No CSRF token found". Because of my lack of understanding of their importance and my desire to continue developing the application as quickly as possible, I found a way to disable CSRF tokens for the backend and was able to continue developing the web application.

When we were presented with the idea of the mastery extension, CSRF tokens immediately came to mind due to my struggles with them during my internship. I figured that I would explore and demonstrate the behavior of a backend application when trying to make requests with and without tokens.

CSRF Token Background

When considering web security, there are many things that can be exploited and/or go wrong for users. One such vulnerability is known as cross forgery, or CSRF, which is an exploit that gets users to use legitimate cookies to perform malicious actions unintentionally. Although these kinds of attacks were much more common in the past, CSRF attacks are still possible under certain conditions. Even if the web browser being used adopts SameSite by default policy, state-changing requests can attack users through an HTTP GET request.

The consequences of a CSRF attack are variable and entirely reliant on the kind of attack being done. Things such as changing the email on a user's account, changing grades in a school system, or transferring bank funds are all exploits that can be done through a CSRF attack.

For a CSRF attack to be performed, there are several conditions that must be met:

1. There is some sort of action that the attacker has a reason to exploit. There must be some sort of target that the attacker is aiming to execute through the CSRF attack.
2. The parameters of the request must be known by the attacker. If the attacker does not know the parameters of the request, the target action is not going to be vulnerable to any CSRF attack.
3. Lastly, the application that is being targeted only uses session cookies to identify the source of the request. Including other mechanisms to validate where the request is coming from will prevent CSRF attacks from happening.

Considering how harmful the consequences of CSRF attacks are, defenses against them were made, the primary one being CSRF tokens. A CSRF token is "a unique, secret, unpredictable value that is generated by the server-side application and transmitted to the client in such a way that it is included in any subsequent HTTP request by the client." As stated above, to perform a CSRF attack, the attacker needs to know all the parameters of the request, and since CSRF tokens should be unpredictable, it makes it nearly impossible to perform the attack. Without the CSRF token header in the request, the server will dismiss it as a fraudulent request, preventing the exploit from occurring.

Work Done

Overview

The goal of my project is to demonstrate the interaction between the server and a client with and without a CSRF token. I demonstrated this by developing a simple backend with the Django framework and a frontend in JS react. Together, the front end and back end count the number of clicks that have occurred since the backend started running. Users can attempt to increase the number of clicks by pressing a button, but the backend server will not allow them to increase the number of clicks without a valid CSRF token.

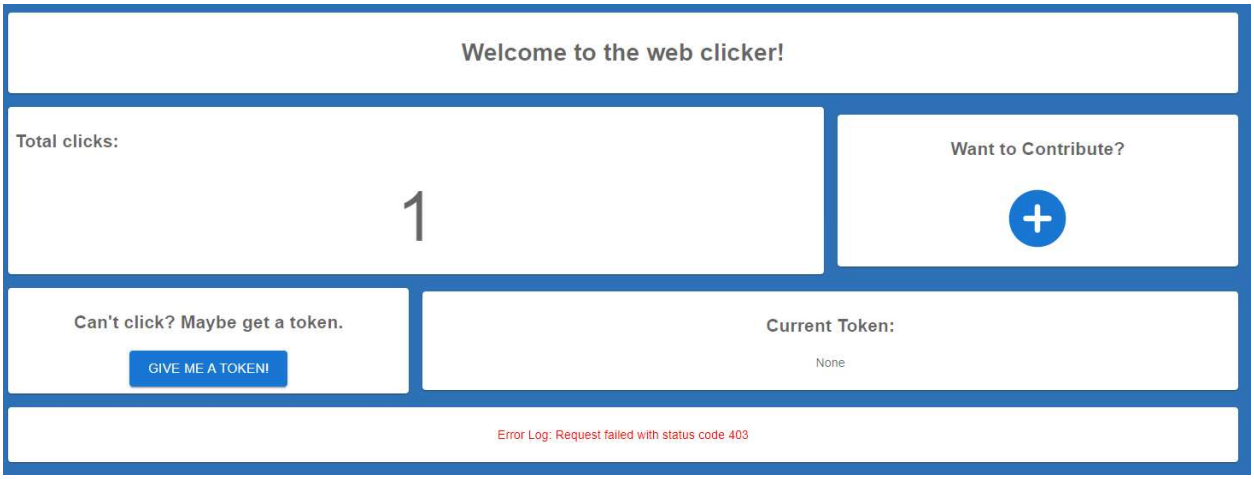
Front End

Interface

When users type the address of the web application into their browser, they are presented with following interface:

On this screen, there is a welcome message, a widget to display the number of clicks, a button to increase the number of clicks, a button to request a token, a widget to display the current token (if any), and an error log.

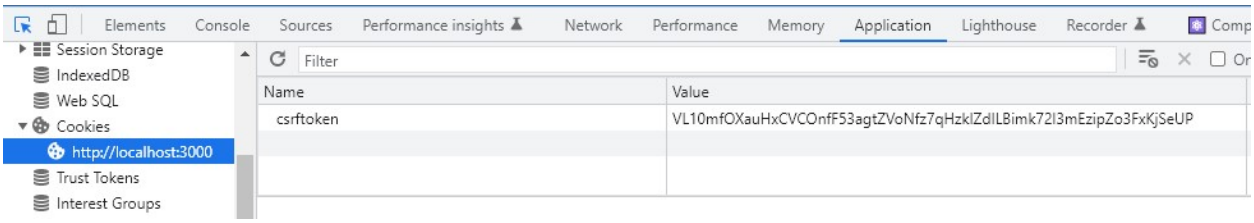
If users try to immediately click on the plus button under the text "Want to Contribute?", they are going to encounter an error. The total clicks is not going to change, and the error log is going to display "Request failed with status code 403".



An HTTP response with code 403 means that a client is forbidden from accessing whatever resource they are trying to get. This error is occurring because the client (the front end in this case), does not have a CSRF token, therefore preventing them from performing the PATCH request to the backend that they are attempting.

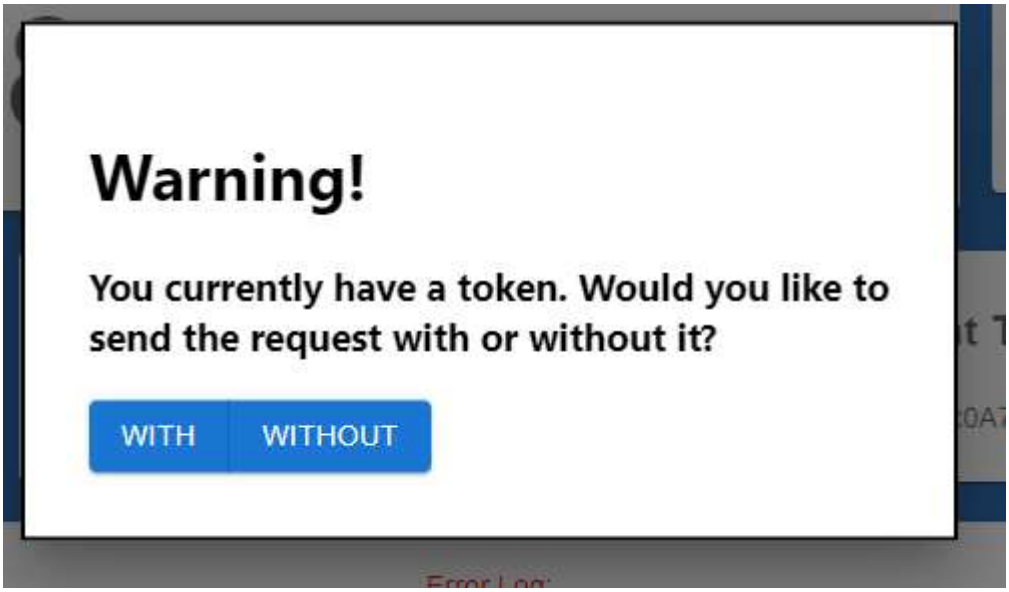
However, this can be easily fixed by pressing the "GIVE ME A TOKEN" button. Upon pressing this, the token value will be displayed in the item box to the right of that button.

This token is also stored as a cookie and can be seen through the application console on your browser.



Now that the user has a token, they can successfully contribute to the server's click count!

When the user then goes to click the button, a modal window pops up to prompt the user about whether or not they want to include the CSRF token in the request.



If the user presses the "WITH" button, the user will send the token in the HTTP header and the count on the screen will go up! Otherwise, if the user presses without, no CSRF token will be included in the header and the user will continue to receive the 403 error.

Code

To make requests to the backend, I used the axios library for react. Axios provides a lot of APIs that can be made to perform asynchronous HTTP requests. These APIs were used for all the communication between the front end and the back end.

```
axios.defaults.xsrfHeaderName = "X-CSRFTOKEN";
axios.defaults.xsrfCookieName = "csrftoken";
axios.defaults.withCredentials = true;
```

This segment of code sets the default CSRF header name and cookie name and setting the withCredentials setting to true guarantees that cross site access control requests are going to be made with credentials, in this case, a CSRF token.

This means that for any of the requests called from the functions below, the X-CSRFTOKEN header is going to be included in every call, even if the client has not received a CSRF token from the server yet.

```
async function get_token() {
    try {
      const result = await axios.get(`${HOST}/csrf/`);
      return result.data.token;
    } catch (error) {
      console.log(error);
```

```
    }
  }
```

```
async function request_for_token() {
    setTokenIsLoading(true);
    get_token().then(result => {
      if (result) {
        setCookie('csrftoken', result);
        setToken(result);
        setTokenIsLoading(false);
        setErrorLog("");
      }
    })
  }
```

These functions are responsible for requesting a token from the backend. It is important to know that this is not a secure method to receive a CSRF token, since this method is prone to man in the middle attacks. Instead, the method of performing a GET request to get a token is merely for simplicity so the project can demonstrate the differences in server behavior when the client has or does not have a token.

```
async function post_count() {
    try {
      const result = await axios.patch(`${HOST}/count/`);
      setCount(result.data.count);
      setErrorLog("");
    } catch (error) {
      setErrorLog(error.message);
    }
  }
```

This function is responsible for requesting that the click count should increase. This request is only going to succeed if a CSRF token is present. Remember, the token does not need to be explicitly called in the API call because of the settings that were changed in previous code segments.

Backend

The backend is responsible for token generation, and reporting the total amount of clicks. To configure Django with cross site origin enabled as well as CSRF security, the following installed apps and middleware need to be included:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

These variables also need to be set.

```
CORS_ORIGIN_ALLOW_ALL = True
CSRF_TRUSTED_ORIGINS = ['http://localhost:3000']
CSRF_COOKIE_SECURE = False
CSRF_COOKIE_NAME = "csrftoken"
CSRF_COOKIE_HTTPONLY = False
CORS_EXPOSE_HEADERS = ["Content-Type", "HTTP_X_CSRFTOKEN"]
CORS_ALLOW_CREDENTIALS = True
```

Some Variable explanations:

- ***CORS_ORIGIN_ALLOW_ALL***: this is set to true to allow requests from any origin
- ***CSRF_TRUSTED_ORIGINS***: origins that have CSRF tokens that are trusted. In this case, only the localhost where the react is being hosted is trusted, but can be changed if ever put to production.
- ***CSRF_COOKIE_SECURE***: this is set to false because development servers only work with HTTP requests but should be changed to true so the production server can use HTTPS.
- ***CORS_EXPOSE_HEADERS***: allows the python scripts to see the CSRF token header

The actual functions of the request are quite simple, the first one being a request to get a CSRF token.

```
@require_GET
def get_csrf(request: HttpRequest) -> JsonResponse:
    """
```

```
    Function to retrieve CSRF token

    Parameters:
        request (HttpRequest): get request to retrieve token

    Returns:
        response (JsonResponse): response with token in it
    """
    response = JsonResponse({
        'token': get_token(request)
    })

    return response
```

This is a simple function that takes advantage of the django.middleware.csrf.get_token function to easily generate a CSRF token to send to the client. Again, this is not the best way of transferring a CSRF token, but done for simplicity.

The other function deals with requests for count, which are either GET or PATCH requests.

```
count = 1

@require_http_methods(['GET', 'PATCH'])
def count_request(request: HttpRequest) -> JsonResponse:
    """
    Function to retrieve current count

    Parameters:
        request (HttpRequest): get request to retrieve count

    Returns:
        response (JsonResponse): response with count
    """
    global count

    print(json.dumps(dict(request.headers), indent=2))

    if request.method == 'PATCH':
        count += 1

    return JsonResponse({
        'count': count
    })
```

First, for debugging and demonstration, the function prints the headers of the request to the console. Then, it checks to see what kind of method the function is. If it is a PATCH request, it will first increment the count variable, and the function ends by sending back the current count value. If this function is called through a PATCH request without a CSRF token, the following will output to the terminal:

```
Forbidden (CSRF cookie not set.): /count/
[01/Dec/2022 13:36:49] "PATCH /count/ HTTP/1.1" 403 2870
```

However, with a CSRF token, this is what the output of the terminal will be:

Conclusion

In conclusion, this project explains a little bit about what CSRF tokens are, and how they might be used to verify a client trying to interact with a server. I definitely got a lot out of this project, and I am happy that I was able to explore a topic that had originally perplexed me earlier this year.

Appendix

settings.py

```
"""
Django settings for backend project.

Generated by 'django-admin startproject' using Django 4.1.

For more information on this file, see
https://docs.djangoproject.com/en/4.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-g0#e4(10!&k3+k$njsoa*h1m7)c^4f-#bxpc)+c!wn%2ley3!4'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders'
]
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'backend.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

from corsheaders.defaults import default_headers

CORS_ORIGIN_ALLOW_ALL = True
CSRF_TRUSTED_ORIGINS = ['http://localhost:3000']
CSRF_COOKIE_SECURE = False
CSRF_COOKIE_NAME = "csrftoken"
CSRF_COOKIE_HTTPONLY = False
CORS_EXPOSE_HEADERS = ["Content-Type", "HTTP_X_CSRFTOKEN"]
CORS_ALLOW_CREDENTIALS = True

WSGI_APPLICATION = 'backend.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/
```

```
STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

views.py

```
from django.http import *
from django.middleware.csrf import get_token
from django.views.decorators.http import *
from django.views.decorators.csrf import csrf_exempt
import json

@require_GET
def get_csrf(request: HttpRequest) -> JsonResponse:
    """
    Function to retrieve CSRF token

    Parameters:
        request (HttpRequest): get request to retrieve token

    Returns:
        response (JsonResponse): response with token in it
    """
    response = JsonResponse({
        'token': get_token(request)
    })

    return response

count = 1

@require_http_methods(['GET', 'PATCH'])
def count_request(request: HttpRequest) -> JsonResponse:
    """
    Function to retrieve current count

    Parameters:
        request (HttpRequest): get request to retrieve count

    Returns:
```

```
        response (JsonResponse): response with count
    """
    global count

    print(json.dumps(dict(request.headers), indent=2))

    if request.method == 'PATCH':
        count += 1

    return JsonResponse({
        'count': count
    })
```

App.js

```
import './App.css';
import React from 'react';

import { useState, useEffect } from 'react';
import axios from 'axios';
import { useCookies } from 'react-cookie';

import CircularProgress from '@mui/material/CircularProgress';
import {
  Grid,
  Paper,
  IconButton,
  Button,
  Modal,
  Box,
  ButtonGroup
} from '@mui/material';
import { styled } from '@mui/material/styles';
import AddCircleRoundedIcon from '@mui/icons-material/AddCircleRounded';

const HOST = "http://localhost:8000";
axios.defaults.xsrfHeaderName = "X-CSRFTOKEN";
axios.defaults.xsrfCookieName = "csrftoken";
axios.defaults.withCredentials = true;

const Item = styled(Paper)(({ theme }) => ({
  backgroundColor: theme.palette.mode === 'dark' ? '#1A2027' : '#fff',
  ...theme.typography.body2,
```

```
  padding: theme.spacing(1),
  textAlign: 'center',
  color: theme.palette.text.secondary,
}));

const style = {
  position: 'absolute',
  top: '50%',
  left: '50%',
  transform: 'translate(-50%, -50%)',
  width: 400,
  bgcolor: 'background.paper',
  border: '2px solid #000',
  boxShadow: 24,
  p: 4,
};



function App() {
  const [countIsLoading, setCountIsLoading] = useState(true);
  const [count, setCount] = useState(undefined);
  const [errorLog, setErrorLog] = useState("");
  const [token, setToken] = useState('None');
  const [cookies, setCookie, removeCookie] = useCookies(['cookie-name']);
  const [modalOpen, setModalOpen] = useState(false);

  async function get_count() {
    try {
      const result = await axios.get(`${HOST}/count/`);
      return result.data.count;
    } catch (error) {
      console.log(error);
    }
  }

  async function post_count() {
    try {
      const result = await axios.patch(`${HOST}/count/`);
      setCount(result.data.count);
      setErrorLog("");
    } catch (error) {
      setErrorLog(error.message);
    }
  }
```

```
  async function get_token() {
    try {
      const result = await axios.get(`${HOST}/csrf/`);
      return result.data.token;
    } catch (error) {
      console.log(error);
    }
  }

  async function request_for_token() {
    get_token().then(result => {
      if (result) {
        setCookie('csrftoken', result);
        setToken(result);
        setErrorLog("");
      }
    })
  }

  useEffect(() => {
    removeCookie('csrftoken');
  }, []);

  useEffect(() => {
    get_count().then(result => {
      if (result) {
        setCount(result);
        setCountIsLoading(false);
      }
    })
  }, [])

  return (
    <div className="App">
      <Modal
        open={modalOpen}
        onClose={() => setModalOpen(false)}
        aria-labelledby="modal-modal-title"
        aria-describedby="modal-modal-description"
      >
        <Box sx={style}>
          <h1>Warning!</h1>
          <h3>You currently have a token. Would you like to send the request with
or without it?</h3>
```

```
          <ButtonGroup variant="contained">
            <Button onClick={() => {
              axios.defaults.withCredentials = true;
              post_count();
              setModalOpen(false);
            }}>
              With
            </Button>
            <Button onClick={() => {
              axios.defaults.withCredentials = false;
              post_count();
              setModalOpen(false);
            }}>Without</Button>
          </ButtonGroup>
        </Box>

      </Modal>
      <div style={{'width':'75%', 'marginTop': '15px', 'marginLeft': 'auto',
'marginRight': 'auto'}}>
        <Grid container
          spacing={2}
          justifyContent='center'
          alignItems='center'
        >
          <Grid item xs={12}>
            <Item>
              <h1>Welcome to the web clicker!</h1>
            </Item>
          </Grid>
          <Grid item xs={8}>
            <Item>
              <div style={{'textAlign': 'left'}}>
                <h2>Total clicks:</h2>
              </div>
              {countIsLoading && <CircularProgress />}
              {!countIsLoading && <p style={{fontSize: '80px', margin:
0}}>{count}</p>}
            </Item>
          </Grid>
          <Grid item xs={4}>
            <Item>
              <div>
                <h2>Want to Contribute?</h2>
              </div>
              <div style={{display: 'flex',
```

```
                alignItems: 'center',
                justifyContent: 'center'}}>
                <IconButton sx={{display: 'flex', alignItems: 'center'}}
onClick={() => {
                  if (token === 'None')
                    post_count()
                  else
                    setModalOpen(true);
                }}>
                    <AddCircleRoundedIcon color='primary' sx={{fontSize:
'80px'}}/>
                </IconButton>
              </div>
            </Item>
          </Grid>
          <Grid item xs={4}>
            <Item>
              <div>
                <h2>Can't click? Maybe get a token.</h2>
              </div>
              <div style={{display: 'flex',
                alignItems: 'center',
                justifyContent: 'center'}}>
                <Button onClick={request_for_token} size='large'
variant='contained'>Give me a token!</Button>
              </div>
            </Item>
          </Grid>
          <Grid item xs={8}>
            <Item>
              <h2>Current Token:</h2>
              <p>{token}</p>
            </Item>
          </Grid>
          <Grid item xs={12}>
            <Item>
              <p style={{'color': 'red'}}>Error Log: {errorLog}</p>
            </Item>
          </Grid>
        </Grid>
      </div>
    </div>
  );
}
```

```
export default App;
```