Member-only story

How to Implement Google ReCaptcha v3 in Next.js?

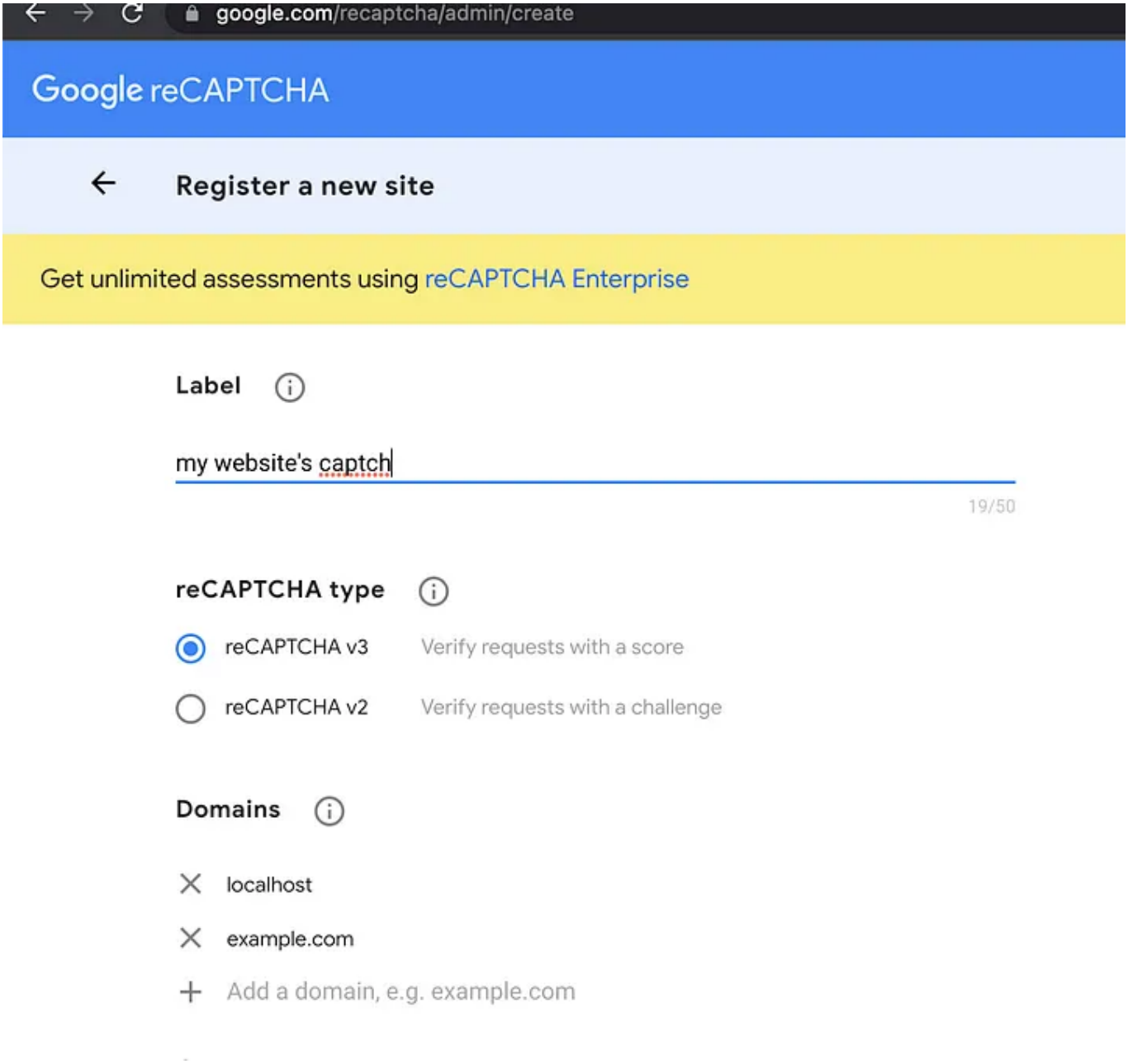
Meta Collective · Follow

Published in JavaScript in Plain English · 5 min read · Apr 13

In this post, I will quickly go over the steps needed to implement Google's ReCaptcha in a NextJS application.

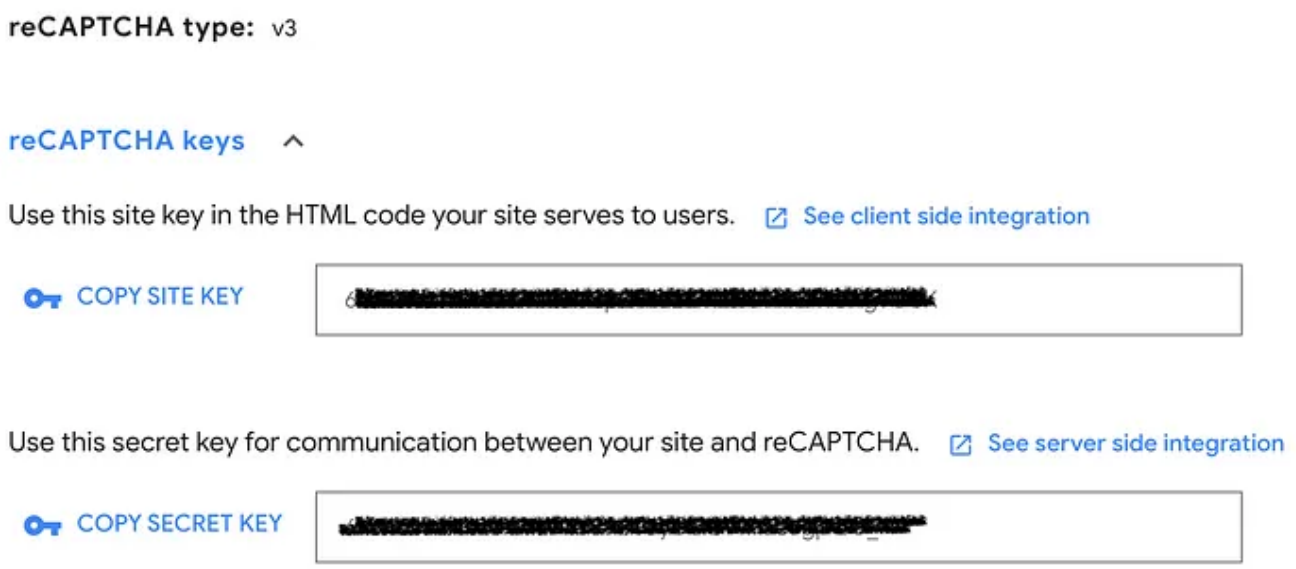
How to get started?

Head over to the official website of Google ReCaptcha and register your website for ReCaptcha V3. you can also add localhost in case you are testing it locally



ReCaptcha V3 setup

Once you have registered, you will get the site key and secret key. We will use them in our application to make API calls for implementing ReCaptcha



How to implement in NextJs?

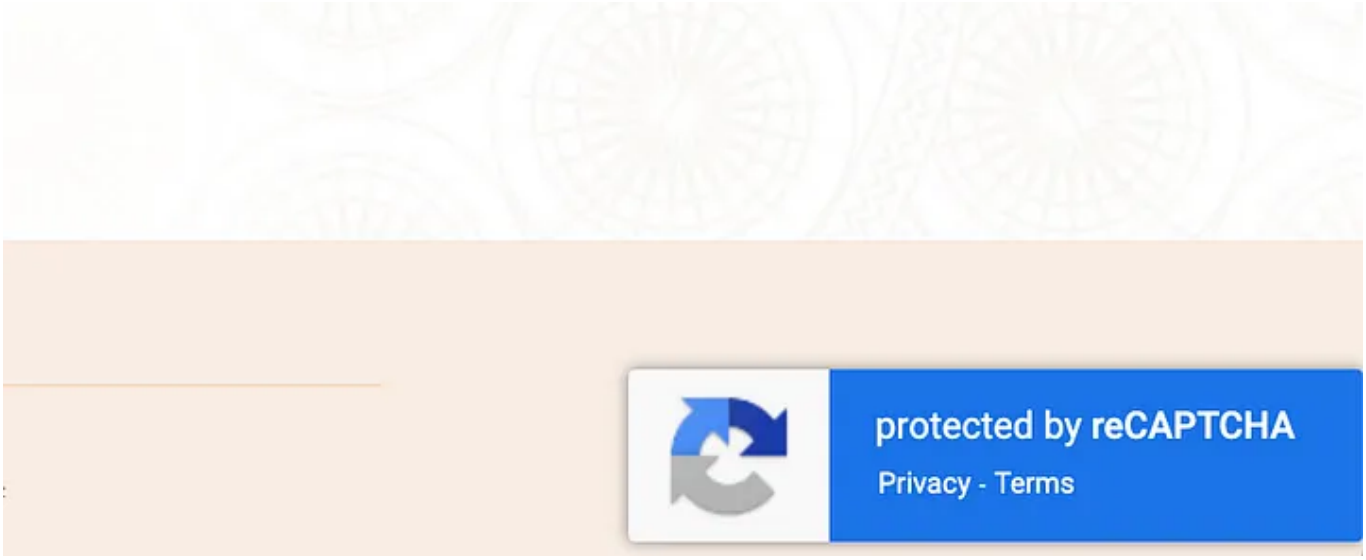
Install `react-google-recaptcha-v3` package. This is a React library for integrating Google ReCaptcha V3 into your application.

Then, head over to `_app.tsx` and implement `GoogleReCaptchaProvider` like this

```
import React from "react";
import { GoogleReCaptchaProvider } from "react-google-recaptcha-v3";

const App = ({ Component, pageProps }) => {

  return (
    <>
      <GoogleReCaptchaProvider
      reCaptchaKey=process.env.RECAPTCH_SITE_KEY
      scriptProps={{
        async: false,
        defer: false,
        appendTo: "head",
        nonce: undefined,
      }}
      >
```

```
          <Component {...pageProps} />
        </GoogleReCaptchaProvider>
      </>
    );
};

export default App;
```

This will inject a ReCaptcha on your site like this



ReCaptcha V3 added to the website

Now, we have to validate the token with the form submission and make sure that it uses Google ReCaptcha.

Create a new API file in your `/pages/api` folder and name it whatever you want it to be. I called mine `contactUs.ts` and it looks something like this -

```
import axios from "axios";
import type { NextApiRequest, NextApiResponse } from "next";

const verifyRecaptcha = async (token:string) => {
  const secretKey = process.env.RECAPTHA_SECRET_KEY;
```

```
  var verificationUrl =
    "https://www.google.com/recaptcha/api/siteverify?secret=" +
    secretKey +
    "&response=" +
    token;

  return await axios.post(verificationUrl);
};

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse<any>
) {
  try {
   const token = req.body.gRecaptchaToken;

    // Recaptcha response
    const response = await verifyRecaptcha(token);

    // Checking if the reponse sent by reCaptcha success or not and if the score
    // In ReCaptcha v3, a score sent which tells if the data sent from front end
    // For more info check, https://developers.google.com/recaptcha/docs/v3
    // ReCaptcha v3 response, {
    //     "success": true|false,      // whether this request was a valid reCAP
    //     "score": number             // the score for this request (0.0 - 1.0)
    //     "action": string            // the action name for this request (impo
    //     "challenge_ts": timestamp,  // timestamp of the challenge load (ISO f
    //     "hostname": string,         // the hostname of the site where the reC
    //     "error-codes": [...]        // optional
    //   }
    if (response.data.success && response.data.score >= 0.5) {
      //INSERT API/LOGIC for saving data once the validation is complete
    } else {
      return res.json({
        status: "error",
        message: "Something went wrong, please try again!!!",
      });
    }
  } catch (error) {
    console.log("ERRRRROr", error);
    res.json({
      status: "error",
      message: "Something went wrong, please try again!!!",
    });
  }
}
```

Here we are simply passing the token from our frontend application to Google for verification. It will return a response with a score in it. As per the current standard guidelines, a score above 0.5 is considered safe and recognised as not Bot. After that, you can process your data as you, please. In my case, I simply passed it over to a backend API to further process the form data.

And to bring all this together, you need to tie it up with your form component like this (This is straight from one of my hobby projects therefore it has a lot of content that may be out of context, but you can check the `handleSumitForm` function for implementation)-

```
import { markdownify } from "@lib/utils/textConverter";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faPhone, faLocationPin } from "@fortawesome/free-solid-svg-icons";
import { useCallback, useState } from "react";
import { useGoogleReCaptcha } from "react-google-recaptcha-v3";
import Alert from "./Alert";

const ContactUs = ({ contact_us, theme }: any) => {
  const { executeRecaptcha } = useGoogleReCaptcha();

  const [name, setName] = useState<string>('');
  const [email, setEmail] = useState<string>('');
  const [message, setMessage] = useState<string>('');
  const [subject, setSubject] = useState<string>('');
  const [notification, setNotification] = useState<string>('');
  const [notificationType, setNotificationType] = useState<string>('');

  const handleSumitForm = useCallback(
    (e: any) => {
      e.preventDefault();
      if (!executeRecaptcha) {
        console.log("Execute recaptcha not yet available");
        return;
      }
      executeRecaptcha("enquiryFormSubmit").then((gReCaptchaToken) => {
        console.log(gReCaptchaToken, "response Google reCaptcha server");
        submitEnquiryForm(gReCaptchaToken);
      });
```

```
    },
    [executeRecaptcha,name,message,email,subject]
  );

  const submitEnquiryForm = async (gReCaptchaToken: any) => {
    fetch("/api/contactUs", {
      method: "POST",
      headers: {
        Accept: "application/json, text/plain, */*",
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        fromName:name,
        to: email,
        from: "someone@example.com",
        message,
        subject,
        gRecaptchaToken: gReCaptchaToken,
      }),
    })
      .then((res) => res.json())
      .then((res) => {
        console.log(res, "response from backend");
        if (res?.status === "success") {
          setNotification(res?.message);
          setNotificationType(res?.status);
        } else {
          setNotification(res?.message);
          setNotificationType(res?.status);
        }
      });
  };

  return (
    <div className="container">
      {markdownify(contact_us.title, "h1", "text-center font-normal")}
      {markdownify(contact_us.description, "p", "mt-4 text-center")}
      <p className="mt-3 text-center">
        <FontAwesomeIcon
          icon={faPhone}
          style={{
            fontSize: 20,
            color: `${theme.colors.default.theme_color.primary}`,
            marginRight: 10,
          }}
        />
        {contact_us.contacts.phone}
      </p>
      <p className="mt-3 text-center">
        <FontAwesomeIcon
```

```
              icon={faLocationPin}
              style={{
                fontSize: 20,
                color: `${theme.colors.default.theme_color.primary}`,
                marginRight: 10,
              }}
            />
            {contact_us.contacts.address}
          </p>

          { notification &&
            <div className="text-center mt-2">
              <Alert message={notification} type={notificationType}  />
            </div>
          }
          <div className="section row pb-0">
            <div className="col-12 md:col-6 lg:col-7">
              <form
                className="contact-form"
                method="POST"
                onSubmit={handleSumitForm}
              >
                <div className="mb-3">
                  <input
                    className="form-input w-full rounded"
                    name="name"
                    value={name}
                    onChange={(e) => setName(e?.target?.value)}
                    type="text"
                    placeholder="Name"
                    required
                  />
                </div>
                <div className="mb-3">
                  <input
                    className="form-input w-full rounded"
                    name="email"
                    value={email}
                    onChange={(e) => setEmail(e?.target?.value)}
                    type="email"
                    placeholder="Your email"
                    required
                  />
                </div>
                <div className="mb-3">
                  <input
                    className="form-input w-full rounded"
                    name="subject"
                    value={subject}
                    onChange={(e) => setSubject(e?.target?.value)}
```

```
                type="text"
                placeholder="Subject"
                required
              />
            </div>
            <div className="mb-3">
              <textarea
                className="form-textarea w-full rounded-md"
                rows={12}
                name="message"
                value={message}
                onChange={(e) => setMessage(e?.target?.value)}
                placeholder="Your message"
              />
            </div>
            <button type="submit" className="btn btn-primary">
              Send Now
            </button>
          </form>
        </div>

      </div>
    </div>
  );
};

export default ContactUs;
```

The `handleSumitForm` function implements `useCallback` hook of ReactJS and calls our `contactUs` API for validation and further processing

. . .