



Web de mecenazgos

20/06/2022

—

Iván Triguero Curado
Grupo Studium

| | |
|--|-----------|
| Planteamiento del Proyecto | 1 |
| Estudio preliminar | 1 |
| Recorrido rápido por la aplicación | 2 |
| Usuario Donante | 3 |
| Usuario ONG | 3 |
| Objetivos finales | 4 |
| Objetivos futuros | 5 |
| Estimación inicial del coste del sistema | 5 |
| Recursos humanos | 5 |
| Recursos Hardware | 6 |
| Recursos Software | 6 |
| Análisis y diseño de la aplicación | 6 |
| Esquema relacional | 6 |
| Esquema entidad-relación | 7 |
| Esquema MySQL Workbench | 7 |
| NextJS y React | 8 |
| React | 8 |
| Next.js | 8 |
| Estructura de la aplicación | 10 |
| Estructura de la carpeta pages | 10 |
| admin | 11 |
| ONGPage | 11 |
| userPage | 11 |
| recuperarPass | 11 |
| _app.js | 11 |
| confirmarEmail y errorEmail | 11 |
| HomePage | 12 |
| index | 12 |
| ONGS y sobreNosotros | 12 |
| Login | 12 |
| Register | 12 |
| RegisterONG | 12 |
| RegisterPage | 12 |
| Explicación de react para la creación de páginas | 13 |
| Funciones nextjs | 14 |
| Dependencias usadas en mi proyecto | 16 |

| | |
|--|-----------|
| NextJS backend | 18 |
| Api NextJS | 18 |
| API de mi aplicación web | 19 |
| Envío de emails | 21 |
| API de registro | 24 |
| Confirmar email | 24 |
| Recuperar contraseña | 24 |
| Planes de pruebas | 25 |
| Plan de pruebas caja negra | 26 |
| Informe pruebas caja negra | 27 |
| Registrar usuario | 27 |
| Confirma usuario por correo | 27 |
| Al confirmar ya podremos acceder. | 28 |
| Registrar proyecto como ONG | 28 |
| Editar, eliminar proyecto como ONG | 28 |
| Entramos en la página de administración para las ong y creamos, eliminamos y editamos un proyecto. | 28 |
| Crear, editar eliminar registro como administrador | 28 |
| Recuperar contraseña por correo | 28 |
| Donar como donante | 30 |
| Pruebas caja negra | 31 |
| Manual de despliegue | 31 |
| Bibliografía | 32 |

Planteamiento del Proyecto

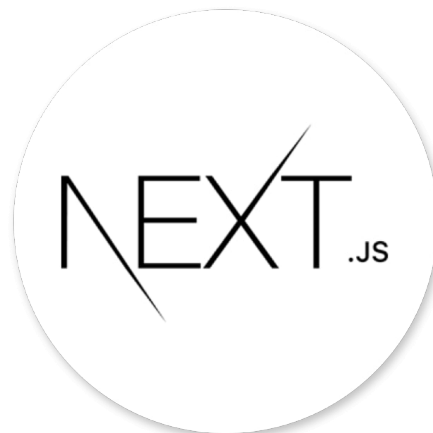
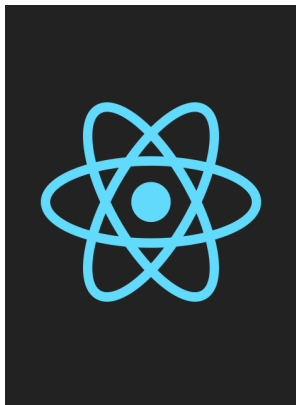
Estudio preliminar

La propuesta inicial de este proyecto es crear una aplicación web, donde existen tres tipos de usuario.

- Usuario administrador. Permite gestionar los registros de distintas tablas, además de visualizar, editar y administrar la información de donaciones necesarias.
- Usuario ONG. Puede realizar peticiones, estableciendo un título, descripción, fecha límite y otro parámetros para que la gente pueda realizar donaciones de la cantidad que considere necesaria para apoyar dicho proyecto.
- Usuario donante. Usuario que realiza donaciones a los proyectos que considere necesarios y/o convenientes, dichas donaciones están administradas por el túnel de ventas de PayPal.

Además de dichas funciones contará con confirmación del usuario por correo electrónico, recuperación de contraseña y demás añadidos.

Toda esta aplicación web será desarrollada en su totalidad en React, en este caso he optado por usar NextJs un framework de React que cuenta con algunos agregados interesantes que explicaré más adelante.



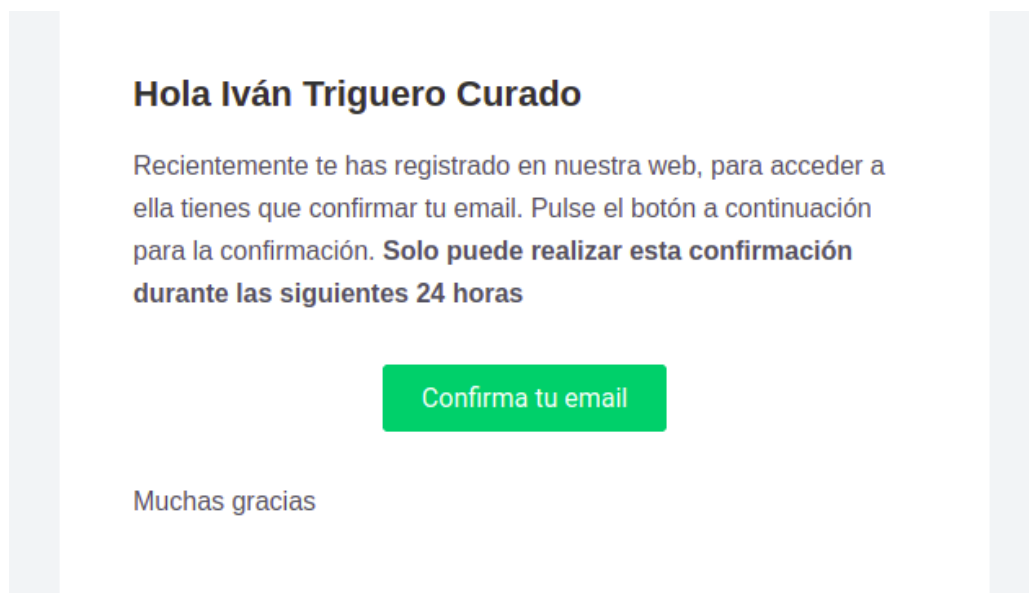
Recorrido rápido por la aplicación

Para entender un poco mejor el funcionamiento de esta aplicación, haremos un recorrido básico como si fuéramos un usuario que va a realizar una donación y un como un usuario ONG que realizará peticiones.

Para comenzar nada más entrar a la web nos encontraremos con algunas pestañas de información sobre cómo funciona, su funcionamiento, finalidad, objetivos de las ONGs, etc...

Si miramos un poco más encontraremos la sección para registrarnos. Podemos registrarnos tanto como usuario donante o como usuario ONG.


Para comenzar con el registro tendremos que rellenar los datos necesarios y se nos enviará un email para confirmar nuestro registro similar al siguiente.



No podremos iniciar sesión hasta que confirmemos nuestro usuario.

Cuando iniciemos sesión podremos ver el panel inicial según el tipo de usuario que ha iniciado sesión.

Usuario Donante



Cerrar sesión

Buscar Proyectos ...

proyecto3

Proyecto de prueba

0.0014-06-2022

Ver detalles

anwanfaw

awnawjf

0.0011-05-2022

Ver detalles

afwa

aw

0.0014-06-2022


Ver detalles

Proyecto de prueba

aa

Proyecto Prueba

Usuario ONG





Crear Proyecto

Cerrar sesión

Buscar Proyectos ...

Proyecto Prueba



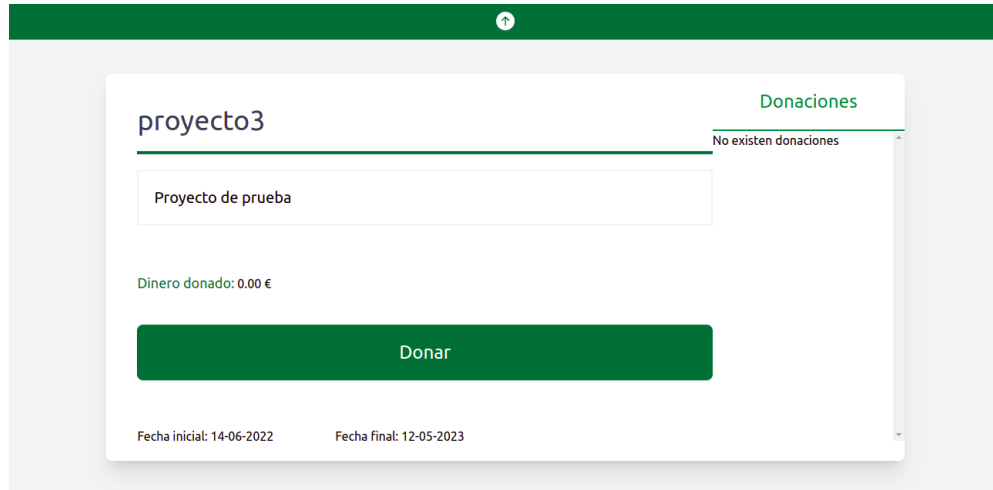
Proyecto Prueba

0.0031-05-2022

Ver detalles

El usuario donante podrá ver todas las donaciones y donar a la que crea conveniente. El usuario ONG solo verá sus peticiones que podrá editar, eliminar, ver detalles o crear nuevas.

Al ver los detalles podremos observar las donaciones realizadas a la derecha, la información referente a la donación y realizar una nueva donación.



Objetivos finales

El objetivo a alcanzar en la realización de este proyecto está dividido en dos secciones. La parte del administrador que es básicamente un CRUD con las tablas de la base de datos.

La parte de donantes y ONGs. Esta sección es la principal de esta aplicación web, donde se pueden realizar pagos a través de la pasarela de PayPal.

Durante este proyecto he aprendido e implementado varios módulos de npm para agregar funcionalidades como por ejemplo:

- axios. Para realizar peticiones ajax, en este caso a nuestra api.
- mysql2. Para realizar funciones mysql con un pool de conexiones y de manera asíncrona.
- framer-motion. Para realizar animación al renderizar o pasar de un componente a otro, para así hacer más estética la página.
- jsonwebtoken. Para la generación de tokens que se pueden descriptar.
- nodemailer. Para el envío de correos.

Según el alcance en estos meses de desarrollo de la aplicación podré agregar funcionalidades extras.

Objetivos futuros

De contar con más tiempo de desarrollo y con objetivo de desarrollar más en un futuro, estas son algunas de las funciones que se podrían agregar.

- Un chat para comunicarte con un administrador
- Un sistema de puntuación de proyectos.
- Un sistema de comentarios.
- Más personalización de usuario, como por ejemplo fotos de perfil.
- Decodificar el código, por ejemplo separando la parte html de la parte js.

Estimación inicial del coste del sistema

En este apartado veremos el presupuesto estimado que va tener el sistema inicialmente, comprendiendo los diferentes ámbitos como; recursos humanos, recursos de hardware y de software. Para calcular esto desglosaremos cada apartado durante el diseño, desarrollo e implantación del sistema, intentando estimar el tiempo que durará y el coste que tendrá dichas tareas.

Recursos humanos

En este apartado veremos el coste estimado que tendrá la intervención humana en nuestro proyecto durante su desarrollo completo.

| | Horas | Precio hora | Total |
|-------------------------------|-------|-------------|-----------|
| Análisis inicial | 7 | 7,00 € | 49,00 € |
| Planificación del sistema | 7 | 7,00 € | 49,00 € |
| Diseño de la aplicación | 10 | 7,00 € | 70,00 € |
| Desarrollo de la aplicación | 154 | 7,00 € | 1078,00 € |
| Despliegue de la aplicación | 7 | 7,00 € | 49,00 € |
| Pruebas en la aplicación | 8 | 7,00 € | 49,00 € |
| Solucionar posibles problemas | 9 | 7,00 € | 63,00 € |
| Mantenimiento mensual | | | 250,00 € |
| TOTAL | 202 | | 1658,00 € |

Recursos Hardware

En este apartado contemplaremos el coste total del proyecto para el ámbito del hardware.

1 x Maquina Servidor (Alojaremos nuestra API, BD y aplicación web) – 550€

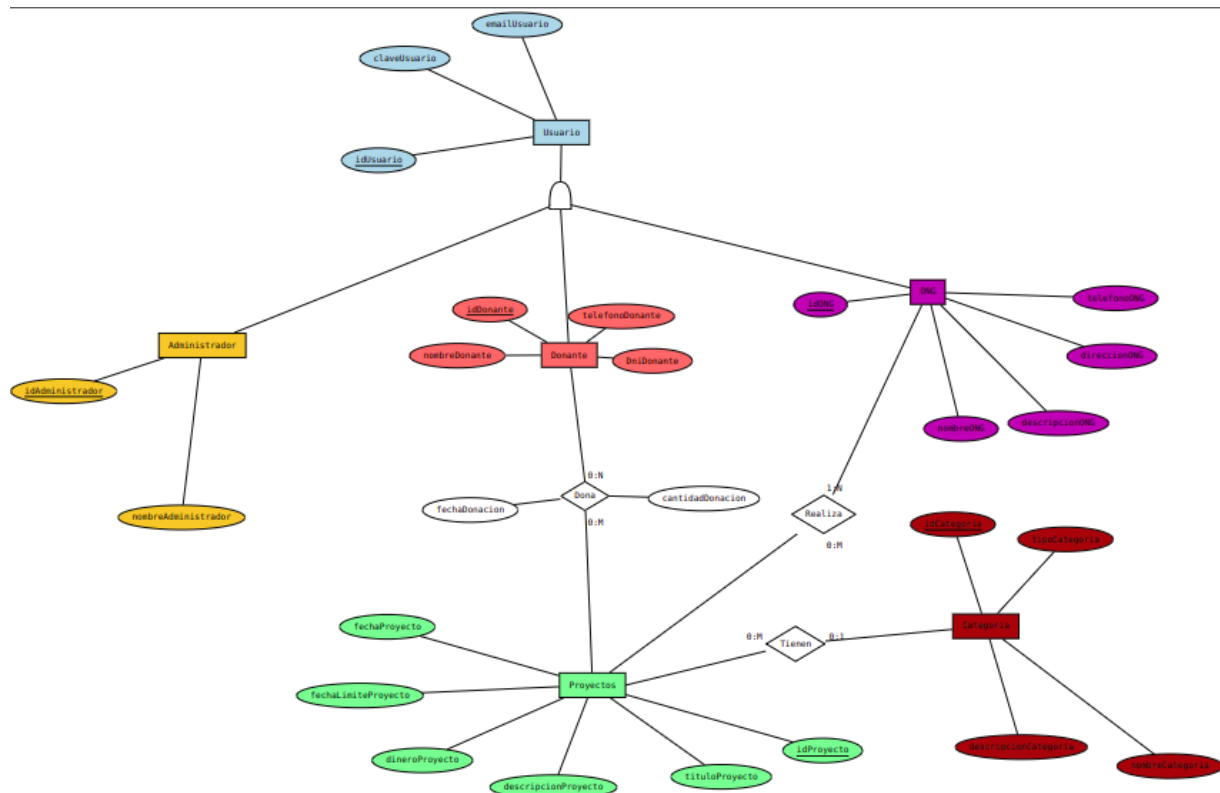
1 x Infraestructura de Red – 250€*

Recursos Software

Para la realización de este proyecto usaríamos software gratuito y frameworks como nextjs, por lo que no existiría un gran gasto en el apartado del software.

Análisis y diseño de la aplicación

Esquema relacional



Esquema entidad-relación

Usuario(idUsuario, emailUsuario, claveUsuario)

Administrador(idAdministrador, nombreAdministrador)

Donante(idDonante, nombreDonante, dniDonante, telefonoDonante)

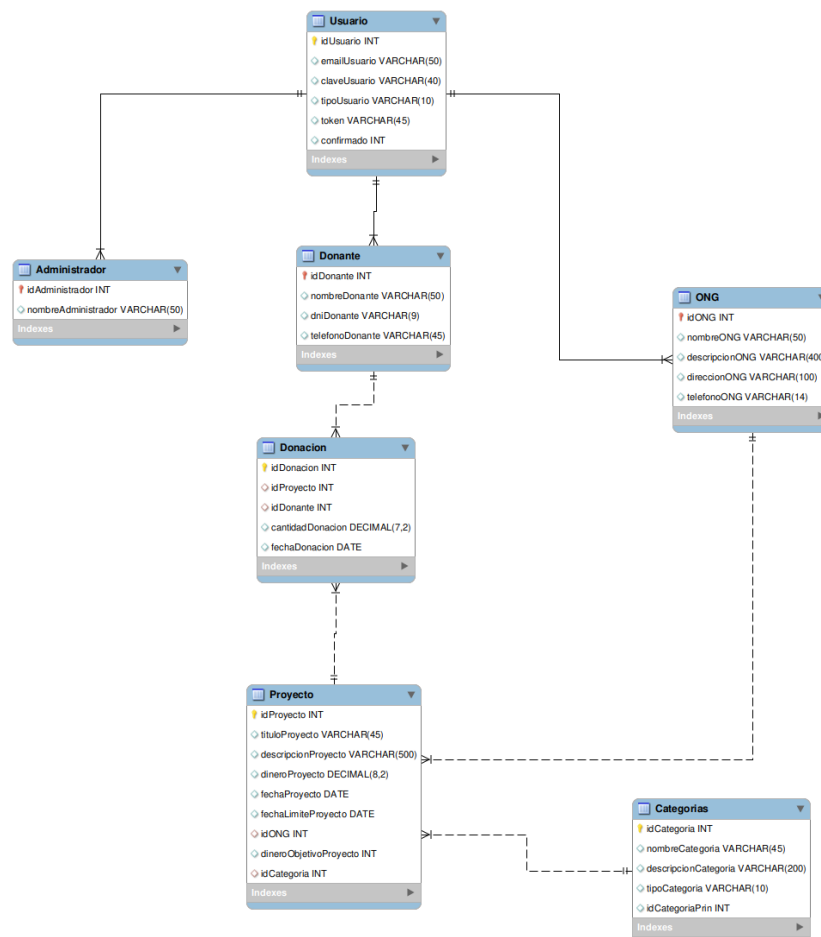
ONG(idONG, nombreONG, descripcionONG, direccionONG, telefonoONG)

Proyecto(idProyecto, tituloProyecto, descripcionProyecto, dineroProyecto, fechaLimiteProyecto, fechaProyecto, idCategoria)

Categoría(idCategoria, nombreCategoria, descripcionCategoria)

Donacion(idDonacion, idProyecto, idDonante, cantidadDonacion, fechaDonacion)

Esquema MySQL Workbench



NextJS y React

React

Facebook lanzó React en 2013. Es una biblioteca de código abierto para proyectos de JavaScript, que permite a los desarrolladores crear aplicaciones web e interfaces de usuario. A pesar de ser relativamente nueva, React se ha vuelto enormemente popular y ha ganado una gran comunidad.

React tiene una gran variedad de beneficios para los desarrolladores. Algunos de ellos son:

- Es fácil de aprender.
- Es fácil de usar con un proyecto.
- Es flexible.
- Tiene componentes reutilizables.
- Es de alto rendimiento.
- Es flexible.
- Mejora la productividad.
- Ofrece estabilidad de código.
- Ofrece muchas herramientas para el desarrollador.
- Tiene un vasto ecosistema.
- Facilita un fuerte desarrollo web.
- Muchas de las principales empresas del mundo utilizan React.

Next.js

Creado por Vercel, Next.js es un marco de trabajo que permite a los desarrolladores crear aplicaciones de página única y aplicaciones web de alto rendimiento a través de la renderización del lado del servidor. Además ofrece generaciones de sitios estáticos, renderizado previo, excelente funcionalidad y otras características. Next.js es una opción extraordinariamente popular.

Ventajas de nextjs

- Es ultrarrápido
- La velocidad de Next.js es una de sus principales ventajas. Este rendimiento ultrarrápido reduce el tiempo de creación.
- Se puede utilizar rutas API (interfaz de programación de aplicaciones)
- ¿Desea utilizar una API de terceros? Next.js facilita esto al ofrecer rutas API. De esa manera, se pueden crear API directamente dentro de la aplicación.

- Es altamente personalizable
- A diferencia de Create-React-App, Next.js es fácil de personalizar. Se pueden agregar complementos de Babel y cargadores de Webpack, por ejemplo.
- Es desplegable
- La acción de desplegar con Next.js es simple. Puede desplegar fácilmente sus aplicaciones React, sin la necesidad de una supervisión exhaustiva.

Una vez explicado esto quiero exponer las ventajas que me hicieron elegir NextJS.

En NextJS contamos con nuestra propia api en el mismo proyecto, de modo que nos permite tener el backend y el frontend en un mismo lugar. Permite renderizar del lado del servidor, de modo que podemos tomar datos antes de que se renderice la página para mostrarlos en esta. Esto es útil por ejemplo, a la hora de mostrar datos de una base de datos por pantalla. Tiempos de carga más rápidos, Next.js es lo suficientemente inteligente como para cargar sólo el Javascript y el CSS necesarios para una página determinada. Ruteo automático, en NextJS las páginas que creamos se enrutan automáticamente basándose en el directorio que fueron creadas y en sus nombres. Estas páginas se crearán dentro de la carpeta pages. Por ejemplo, si creo el fichero Home en pages podré acceder a él simplemente añadiendo /Home a la dirección de nuestra web.

Estructura de la aplicación



Al ir desarrollando la aplicación con React he creado varios componentes y elementos en la estructura del proyecto. Todos estos tienen una finalidad que detallaré más adelante.

Estructura de la carpeta pages

Dentro de esta carpeta se encuentran las páginas a través de las que navegaremos en nuestra web. Esta carpeta la he dividido en varias partes. Existe la parte pública, las cuales son páginas a las que puede acceder cualquiera y que suelen ser de bienvenida, etc. Y las páginas para cada usuario. Por ejemplo para el usuario administrador he creado la carpeta admin y dentro de ella todas las páginas necesarias a las que se accederán mediante `.../admin/nombre_pagina`.

admin

Dentro de esta carpeta se encuentran las páginas accesibles por el usuario administrador, a las cuales se accede a través de la ruta /admin/página.

Dentro se encuentra un fichero por cada tabla que es administrada mediante esta página. Además también contamos con un Template que usaremos para que todas tengan el mismo aspecto

ONGPage

Contiene las páginas accesible por los usuario ONG, en esta caso solo cuenta con un index que mostrará los proyectos de la cuenta ONG que inició sesión, y una carpeta proyecto con una ruta dinámica que mostrará los detalles del proyecto que le pasemos por URL.

userPage

Actúa de forma similar a la carpeta anterior, pero en lugar de mostrar los proyectos creados por el usuario, muestra todos los proyectos y en la carpeta proyecto cambia el aspecto para mostrar los detalles, por ejemplo agrega el botón para donar.

recuperarPass

Contiene las páginas a las que se redirige en caso de querer reestablecer la contraseña, es una ruta dinámica a la que le pasamos un token por URL, comprobará que este token es correcto y mostrará el cambio de contraseña para el usuario asociado al token.

_app.js

Este fichero es el que Next.js usa el para inicializar páginas. Los cambios realizados en este puede afectar a todas. Por ejemplo, en mi caso, la he utilizado para envolver todas las páginas en la etiqueta <AnimatePresence> que es utilizado por framer-motion para las animaciones.

confirmarEmail y errorEmail

Son las páginas mostradas en caso de confirmar un Email correctamente o fallar al confirmar un Email. Simplemente son informativas y no tienen más funcionalidad.

HomePage

Es la primera página que se muestra en la web y la de bienvenida, desde esta se pueden acceder a las demás.

index

Es la primera página que renderiza react, en esta caso verifica si existe un token de inicio de sesión y muestra el HomePage.

ONGS y sobreNosotros

Simplemente son páginas informativas sobre la web.

Login

Muestra el Login de inicio de sesión, cuenta con el botón para reestablecer la contraseña. No se redirige a ella en ninguna otra página simplemente se muestra como una ventana Modal.

Además al apretar el botón Login se encarga de hacer la llamada a la api correspondiente y de guardar el token de inicio de sesión en las cookies.

Register

Contiene el registro para los usuarios donantes

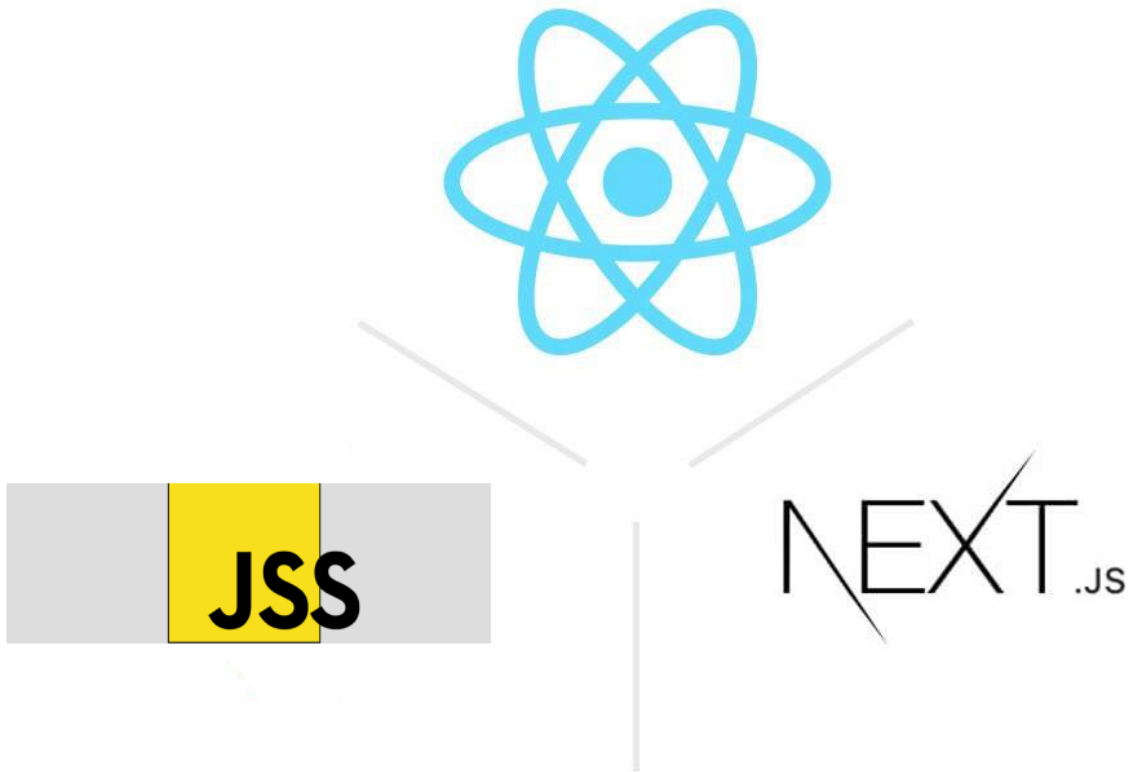
RegisterONG

Contiene el registro para los usuarios ONG

RegisterPage

A través de esta página se puede acceder a las dos clases anteriores según nuestras necesidades. Podemos decir que engloba a Register y RegisterONG

Explicación de react para la creación de páginas



En este apartado, explicaré como hice la creación de las diferentes páginas en mi aplicación web.

Para comenzar, las páginas se encuentran dentro de la carpeta pages, las cuales se enrutan automáticamente a una dirección dependiendo del nombre que le pusimos.

Cada fichero javascript representa una página de nuestra web. Estos ficheros contienen una clase, que devolverá el contenido de la web. Las funciones necesarias también se encuentran dentro de esta clase. Por último, exportamos dicha clase por defecto.

Por ejemplo:

```
const Prueba = () => {  
  return(  
    <h1>¡Hola mundo!</h1>  
  )  
}
```

Esto mostraría un simple Hola mundo por pantallas, obviamente esto se puede complicar mucho más.

Funciones nextjs

React y Nextjs cuentan con algunas funciones útiles a la hora de realizar varias tareas. Aquí voy a exponer y explicar las más usadas a lo largo de mi aplicación:

useState(). Al usar esta función nos devuelve una variable para guardar información que representará el estado de un componente y una función para actualizar dicho estado. Sin embargo, ¿Por que utilizar useState en lugar de una variable normal?

A la hora de actualizar el estado de un componente este se vuelve a renderizar para mostrarse con el estado actualizado.

En mi caso, por ejemplo, lo uso a la hora de mostrar ventanas modales. La ventana solo se muestra si el estado es true. Al actualizar dicho estado se rerenderiza y se muestra u oculta dependiendo del estado.

getServerSideProps. Esta función se tiene que crear fuera de la clase principal. Sirve para renderizar desde el lado del servidor. ¿Para qué sirve esto? Pues bien, lo utiliza en caso de necesitar datos del backend antes de renderizar la página. Por ejemplo, a la hora de mostrar una tabla con los datos de la base de datos.

getServerSideProps tiene que devolver los resultados con:

```
return {  
  props: {  
    datos: datos  
  }  
}
```

Teniendo que devolver los datos como un json. Estos datos son tomados por la clase principal como parámetros.

```
const ONGPage = ({datos}) => {  
    return(  
        ...  
    )  
}
```

useRouter. Se utiliza para tratar con la ruta de nuestro componente. En mi caso lo uso para refrescar la ruta o reemplazarla. Ejemplo:

```
router.replace(router.asPath);
```

Link. Es el equivalente a la etiqueta <a> en Nextjs, utilizado para navegar entre componentes. Ejemplo:

```
<Link href="/" />
```

Todos estos componentes/funciones tienen que importarse antes de ser usados.

En NextJS existen muchas más funciones para ser utilizadas, estas son las más usadas por mi.

Dependencias usadas en mi proyecto

A la hora de crear mi proyecto he necesitados varias dependencias para solucionar algunos problemas y crear alguna funcionalidad. En este apartado voy a especificar las más utilizadas.

Axios.

Probablemente la más usada en mi proyecto. Es utilizada para realizar peticiones ajax. Por ejemplo para obtener los datos del backend. Se le pueden pasar parámetros a la petición, así como modificar los headers. También puede realizar diferentes peticiones como post, get, put o delete. Ejemplo.

```
axios.put('/api/ejemplo, formValue, config)
```

formValue y config son jsons. El formValues contiene datos de un formulario y el config contiene configuración, como por ejemplo algunos headers.

El contenido de la variable config sería:

```
const config = {  
  headers:{  
    authorization: token  
  }  
};
```

En esta caso establecemos un token en la cabecera, que será requerido para tomar dependiendo que datos del backend.

react-icons.

En este caso es una dependencia algo más sencilla, ya que simplemente cuenta con iconos para ser utilizados en nuestra web react simplemente importandolos y estableciendo la etiqueta.

jsonwebtoken

Usado para generar tokens con información. Por ejemplo, es usado para crear el token de inicio de sesión con la información del usuario y que se guarda en una cookie. Este token será encriptado con un clave, la cual deberá ser la misma para desencriptar la información. También son usados estos tokens para funciones que explicaré más adelante.

js-cookie

Utilizado para crear, eliminar o consultar cookies. En este caso es usado para crear, consultar o eliminar la cookie de inicio de sesión.

nodemailer

Esta dependencia es necesaria para enviar correos, en esta aplicación es usada para enviar los correos de confirmar usuario y recuperar contraseña.

framer-motion

Usado para realizar animaciones, al pasar de una página a otra o al cambiar de estado. Todo esto para tener una interfaz más elaborada y estética.

mysql2

Para realizar peticiones al servidor de base de datos. He usado el mysql2 en lugar del uno para usar peticiones asíncronas y conexión a un pool de conexiones.

@paypal

Usado para implementar la API de Paypal para realizar Pagos. Para utilizar esta API tendremos que pasarle en la etiqueta PayPalScriptProvider con el atributo options el client-id que obtendremos al crear un proyecto en la página de paypal developer.

En el atributo createOrder tendremos una petición ajax que llamará al backend para crear la orden de compra.

En el backend crearemos un paypal.core.SandboxEnvironment con el clientID y el clientSecret obtenidos del PayPal developer, y un paypal.core.PayPalHttpClient. Con este paypal.core.PayPalHttpClient crearemos y haremos la petición pasando parámetros como la cantidad o la moneda.

También tendremos un atributo onCancel y onApprove para establecer lo que ocurre si se aprueba la petición o se cancela.

NextJS backend

Una de las ventajas que tiene NextJS es la creación del backend dentro del mismo proyecto y así tener todo en el mismo sitio. Aunque Next cuente con su propia api, es posible utilizar otra api externa en caso de ser necesario.

Api NextJS

Para comenzar, cualquier archivo dentro de la carpeta pages/api se asigna /api/* y se tratará como un punto final de API. Son paquetes solo del lado del servidor y no aumentarán el tamaño del paquete del lado del cliente. Resumiendo, todos estos archivos representarán una ruta de esta api.

Cada fichero contiene una función exportada por defecto llamada handler con los parámetros:

- req: una instancia de `http.IncomingMessage` , además de algunos middleware prediseñados
- res: una instancia de `http.ServerResponse` , además de algunas funciones auxiliares

Representan la petición y la respuesta.

Por ejemplo, esto mandará una respuesta Hola mundo.

```
res.status(200).json({ name: 'Hola mundo' })
```

Para manejar diferentes métodos HTTP en una ruta API, puede usar `req.method` en su controlador de solicitudes, así:

```
export default function handler(req, res) {  
  if (req.method === 'POST') {  
    // Process a POST request  
  } else {  
    // Handle any other HTTP method  
  }  
}
```

Al igual que las rutas dentro de pages, podremos crear rutas dinámicas. Para esto al nombre del archivo lo envolveremos con corchetes: [nombre].js.

La palabra entre corchetes es la que tomaremos como parámetros.

Por ejemplo, para el archivo [id].js tomaremos el parámetro de la siguiente forma:

```
const id=req.query.id
```

Una vez hecho esto podremos usarlo con la constante/variable en la que la almacenamos, en este caso en la constante id.

Con toda esta explicación, mostraré el funcionamiento de la api de mi aplicación.

API de mi aplicación web

Dentro de mi api he creado una dirección para cada tabla de la base de datos. Esta dirección realizará una acción en la base de datos dependiendo de la petición que le enviemos.

- GET. Obtiene datos de la tabla.
- POST. Inserta datos en la tabla.
- PUT. Actualiza los datos de una tabla.
- DELETE. Elimina un registro de la tabla.

Se toman los parámetros enviados por la petición ajax para realizar las respectivas funciones.

También tenemos que saber que para que la api nos devuelva ciertos valores o ejecute ciertas acciones, debemos pasar un token de sesión válido. Este token lo pasaremos por la cabecera y comprobaremos con jsonwebtoken si es válido, en caso de no ser válido enviaremos una respuesta 500 de acceso denegado.

Para esto he creado una función llamada authenticated, que envolverá a la función por defecto de la api.

La función authenticated quedaría de la siguiente forma:

```
const authenticated = (fn) => async (
  req,
  res
) => {
  jwt.verify(req.headers.authorization, serverRuntimeConfig.secret, async function(err,
  decoded){
    if(!err && decoded){
      return await fn(req,res)
    }

    res.status(403).json({message : 'No estás autenticado'})
  })
}
```

Por ejemplo, la api para la table de usuarios se vería así:

```
export default authenticated(async function handler(req, res) {
  switch (req.method){
    case "GET":
      const [rows]=await pool.query("SELECT * FROM proyectointegrado.Usuario join Donante on idUsuario=idDonante;")
      return res.status(200).json(rows)
    case "POST":
      const email=req.body.email
      const clave=req.body.clave
      const nombre=req.body.nombre
      const dni=req.body.dni
      const telefono=req.body.telefono
      const [r]=await pool.query("INSERT INTO `proyectointegrado`.`Usuario` (`emailUsuario`, `claveUsuario`, `tipoUsuario`) VALUES ('"+email+"', sh
      await pool.query("INSERT INTO `proyectointegrado`.`Donante` (`idDonante`, `nombreDonante`, `dniDonante`, `telefonoDonante`) VALUES ('"+r.inse
      return res.status(200).json(r)
    case "PUT":
      const id= req.body.id
      const email=req.body.email
      const clave=req.body.clave
      const nombre=req.body.nombre
      const dni=req.body.dni
      const telefono=req.body.telefono
      if (clave==""){
        await pool.query("UPDATE `proyectointegrado`.`Usuario` SET `emailUsuario` = '"+email+"' WHERE (`idUsuario` = '"+id+"');")
      }else if(clave!=""){
        await pool.query("UPDATE `proyectointegrado`.`Usuario` SET `emailUsuario` = '"+email+"', `claveUsuario` = '"+clave+"' WHERE (`idUsuario` = '"+id+"');")
      }
      const re=await pool.query("UPDATE `proyectointegrado`.`Donante` SET `nombreDonante` = '"+nombre+"', `dniDonante` = '"+dni+"', `telefonoDonante` = '"+telefono+"' WHERE (`idDonante` = '"+id+"');")
      return res.status(200).json(re)
    case "DELETE":
      const id=req.body.id
      await pool.query("DELETE FROM `proyectointegrado`.`Donante` WHERE (`idDonante` = '"+id+"');")
      await pool.query("DELETE FROM `proyectointegrado`.`Usuario` WHERE (`idUsuario` = '"+id+"');")
      return res.status(200).json(id)
  }
})
```

Envío de emails

A continuación explicaré algunas rutas de las api que realizan unas funciones un poco diferentes.

Por ejemplo, la ruta para registrarse envía un email para confirmar el correo. Esto se hace con la librería de nodemailer. El primer paso es establecer el correo que utilizaremos para enviar los mensajes. En mi caso he creado el correo proyectofinalivantriguero@gmail.com. Una vez hecho esto tendremos que pedir a paypal una clave para utilizar su servicio en nuestras aplicaciones, de forma similar a lo hecho con paypal. Con estos parámetros crearemos la conexión con nodemailer.createTransport, tendremos que establecer el servidor smtp del servicio de correos que estemos usando, en mi caso gmail.

Quedaría algo así:

```
const mail = {  
  user: 'proyectofinalivantriguero@gmail.com',  
  pass: 'lxgbxywybexrbekq'  
}  
  
let transporter = nodemailer.createTransport({  
  host: "smtp.gmail.com",  
  port: 465,  
  secure: true,  
  auth: {  
    user: mail.user,  
    pass: mail.pass,  
  },  
});
```


El siguiente paso es crear una función que nos devuelva una plantilla html. Para enviar en el correo.

```
const getTemplate= (name, token, email) => {  
  return `  
    <contenido html>  
  ,  
}
```

Como último paso crearemos la función para enviar el email de esta forma:

```
const sendEmail = async (email, subject, html) =>{  
  try{  
    await transporter.sendMail({  
      from: mail.user,  
      to: email,  
      subject,  
      text: "Hello!",  
      html  
    });  
  }catch(error){  
    console.log(error)  
  }  
}
```

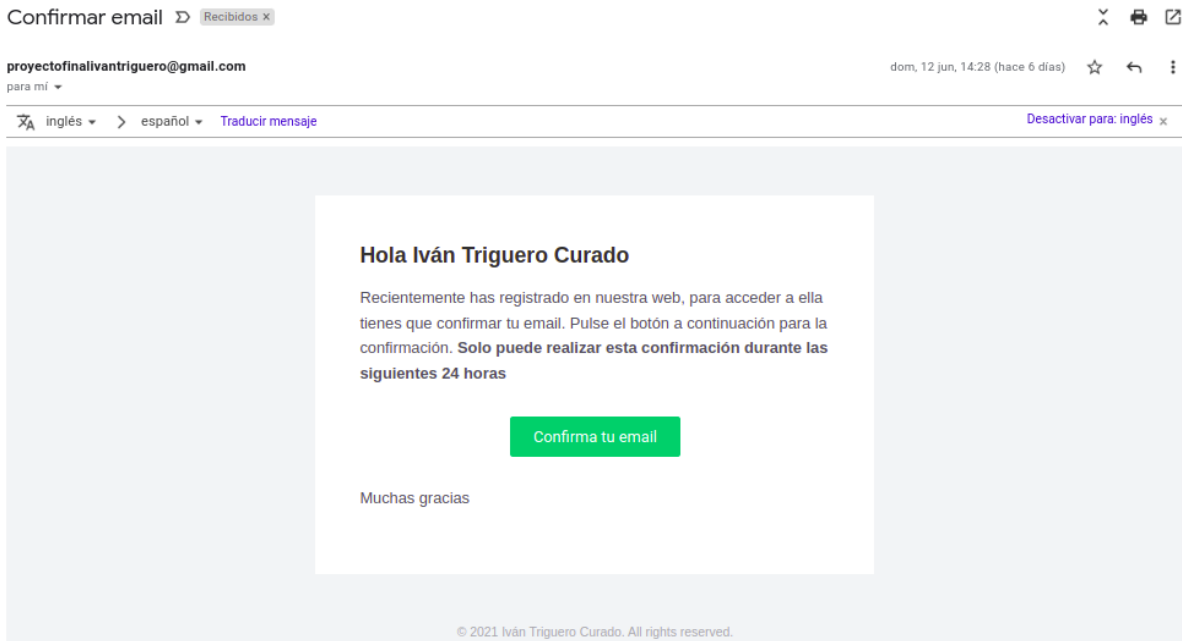
Le pasaremos como parámetros el email destino, el asunto, y la plantilla html.

Esta es la función que usaremos en la api de registro para enviar correos.

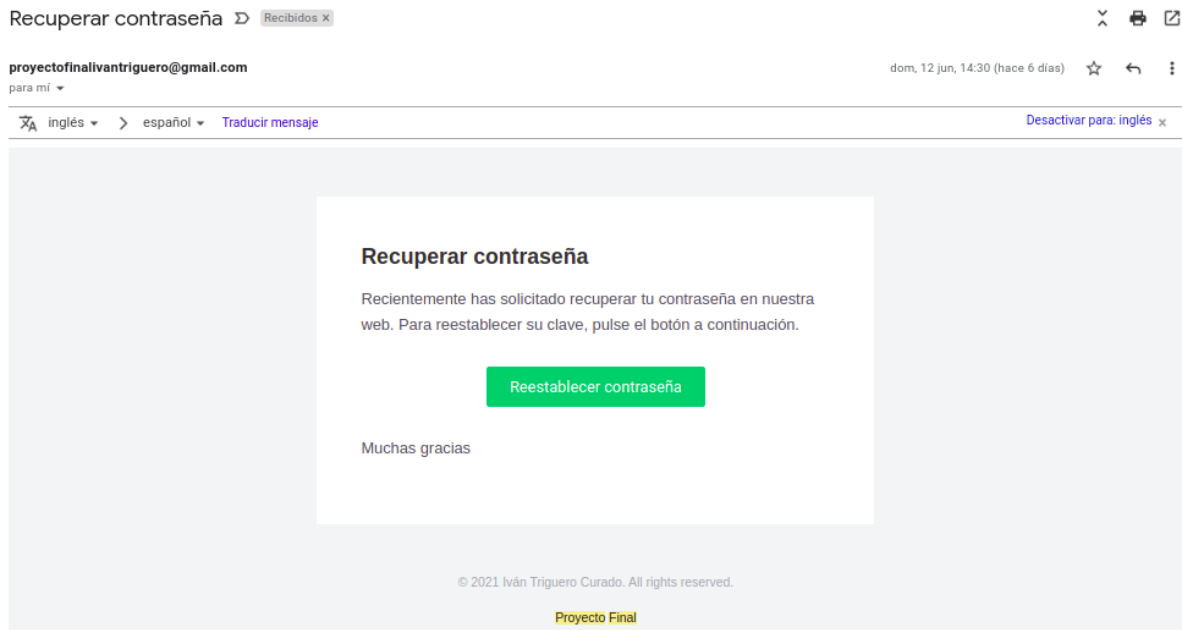
Proyecto Integrado

La apariencia de los correos es algo así.

Correo de confirmación:



Correo de recuperar contraseña:



API de registro

En nuestra API de registro una vez comprobado que el email no está en la base de datos, lo registramos y enviaremos un email con:

```
await sendEmail(email, 'Confirmar email',template)
```

Confirmar email

Para confirmar email, tenemos una ruta con un parámetro dinámico, este parámetro será un token generado único.

Al enviar el email de confirmación, el botón será un enlace a esta api con el token de dicho usuario. Si el token del parámetro es el mismo que el guardado en la base de datos de dicho usuario se confirmará el email. De lo contrario no se confirmará.

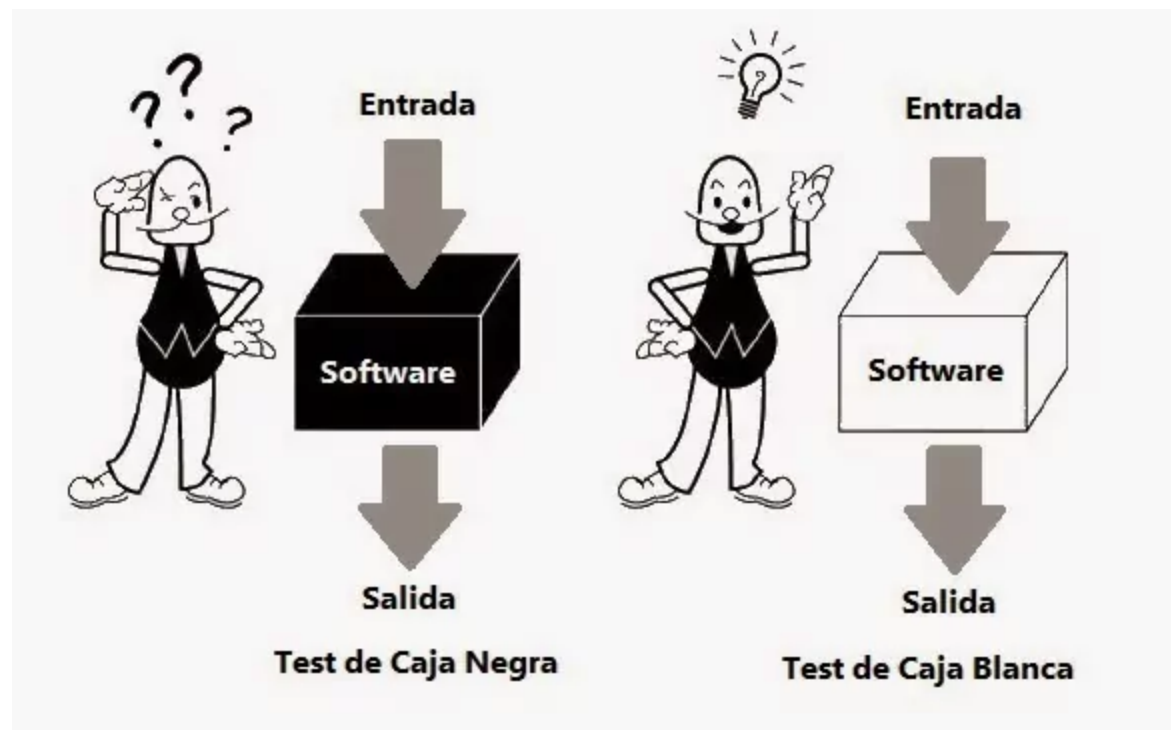
Recuperar contraseña

De forma parecida a la anterior, para recuperar la contraseña, enviaremos un email, con un enlace a la api para recuperar la contraseña. El botón de dicho email llevará a la api con el token del usuario. En caso de que el token sea correcto podremos cambiar la contraseña de dicho usuario.

Planes de pruebas

El siguiente apartado lo dedicaremos para las pruebas de nuestro programa, así comprobaremos que todo funciona correctamente.

Deberemos establecer una serie de pruebas para caja negra y otras para caja blanca.



Plan de pruebas caja negra

Las Pruebas de Caja Negra, es una técnica de pruebas de software en la cual la funcionalidad se verifica sin tomar en cuenta la estructura interna de código, detalles de implementación o escenarios de ejecución internos en el software.

He establecido varias pruebas para comprobar manualmente en mi aplicación.

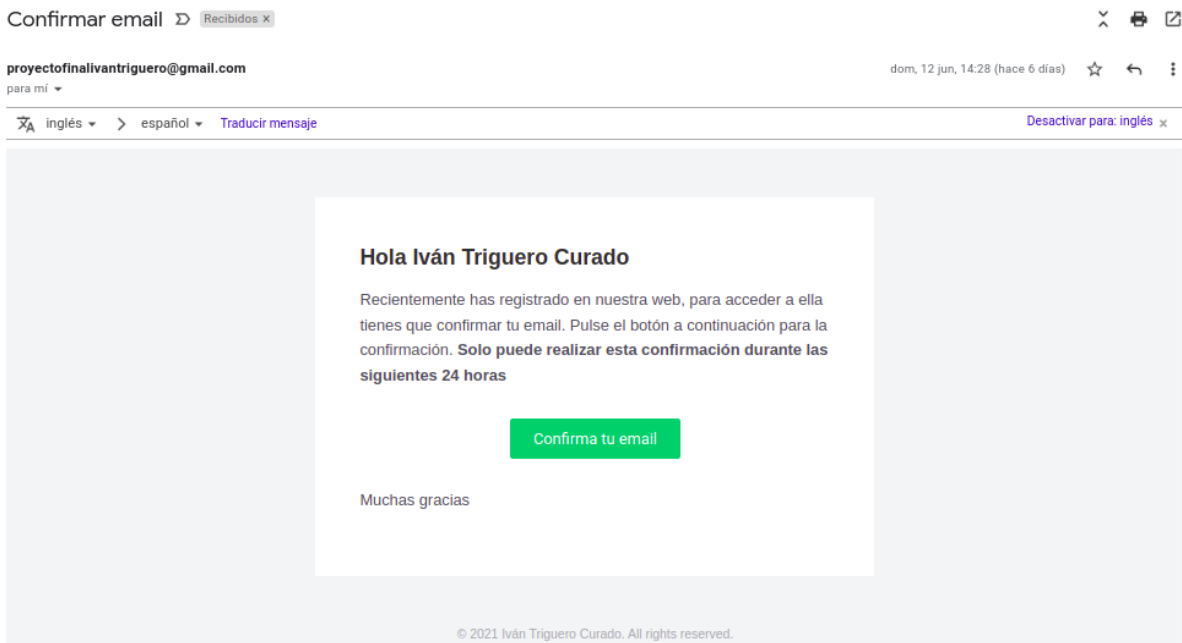
| Prueba | Resultado |
|--|--|
| Registrar usuario | Crear nuevo usuario y recibir email de confirmación |
| Confirma usuario por correo | Confirmar email y poder acceder a la aplicación |
| Registrar proyecto como ONG | Registrar y visualizar el proyecto |
| Editar, eliminar proyecto como ONG | Editar campos o eliminar proyecto correctamente |
| Crear, editar eliminar registro como administrador | Crear, editar o eliminar registros en la base de datos desde la pantalla del administrador |
| Recuperar contraseña por correo | Poder cambiar contraseña con el correo de recuperación |
| Donar como donante | Permitir donar mediante Paypal |

Informe pruebas caja negra

Registrar usuario

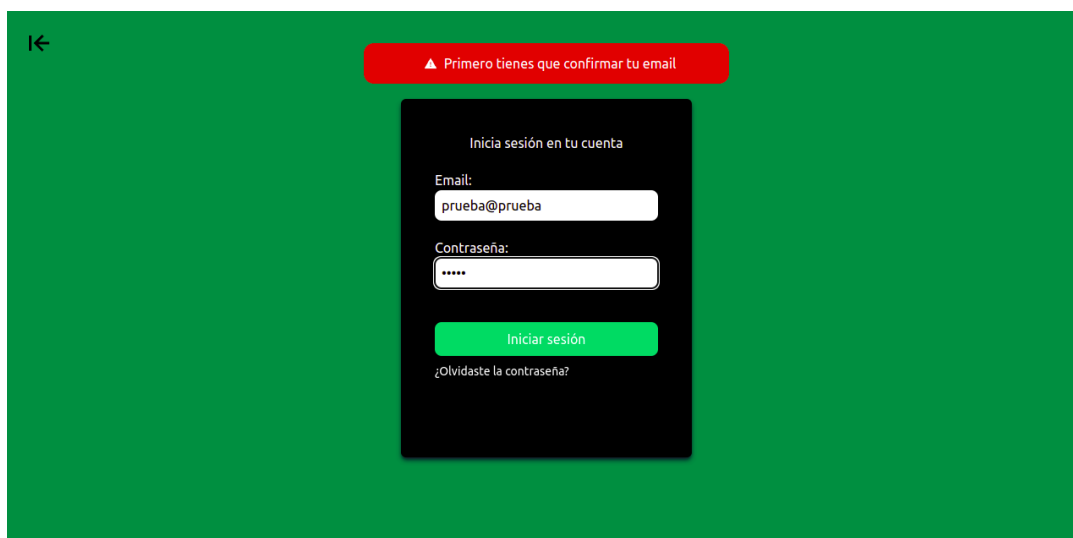
Accedemos al registro, rellenamos los campos y comprobamos que recibimos el email de confirmación.

Email de confirmación:



Confirma usuario por correo

Al intentar iniciar sesión no podremos si no hemos confirmado nuestro correo:



Al confirmar ya podremos acceder.

Registrar proyecto como ONG

Realizamos registro y comprobamos de la misma forma si nos llega el correo y lo confirmamos.

Editar, eliminar proyecto como ONG

Entramos en la página de administración para las ong y creamos, eliminamos y editamos un proyecto.

Crear, editar eliminar registro como administrador

Accedemos a la página del administrador y creamos, editamos y eliminamos los registros de las diferentes tablas.

Recuperar contraseña por correo

Probamos el recuperar contraseña:

Al no estar registrado el correo nos salta un aviso.

X

Introduce tu correo para recuperar tu contraseña

Este correo no está registrado

Enviar correo

Si no está confirmado nos salta otro aviso.

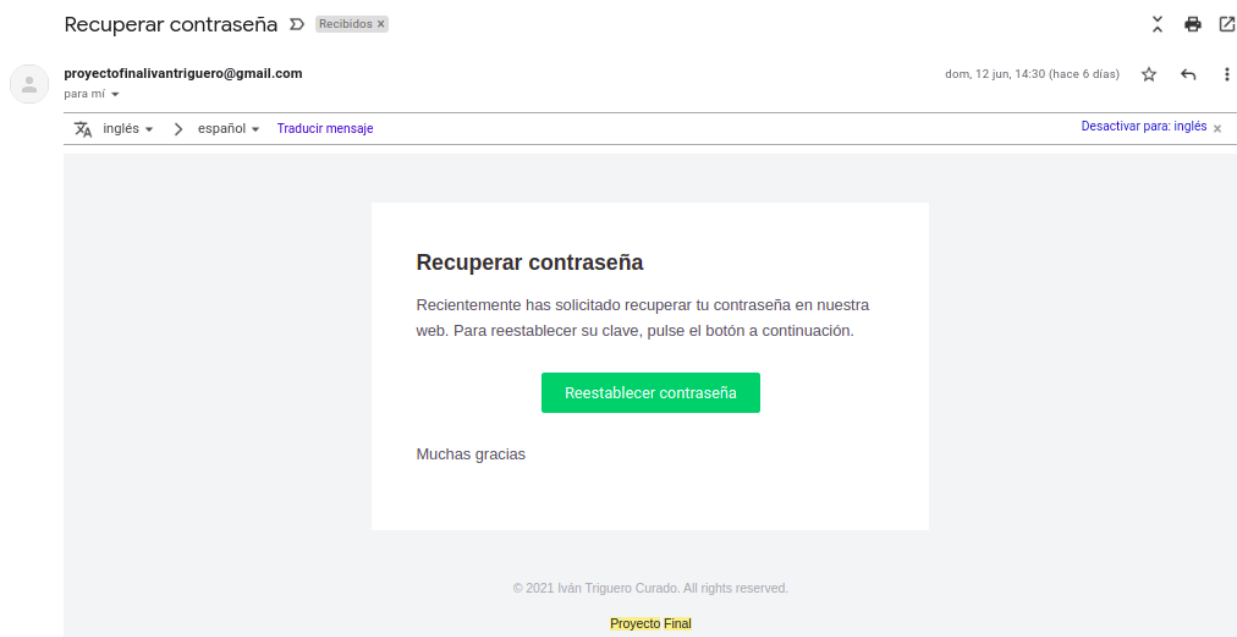
×

Introduce tu correo para recuperar tu contraseña

Este correo no está registrado

Enviar correo

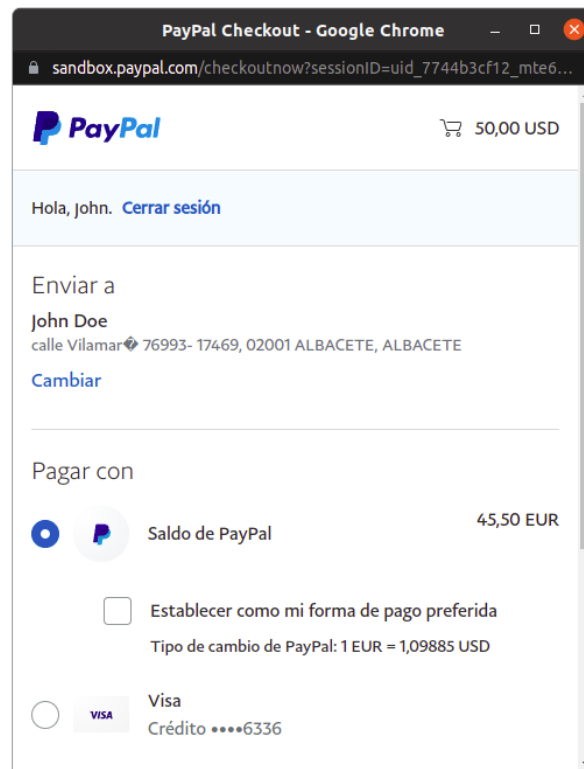
Si está registrado y confirmado nos enviará un correo:



Al pulsar en reestablecer contraseña podremos cambiar la clave, probaremos que podemos iniciar sesión posteriormente.

Donar como donante

Para probar la funcionalidad de la api de PayPal podemos tratar de donar con los usuarios de prueba que nos proporciona el propio PayPal. Al introducir dicho usuario nos muestra la ventana de compra.



Después de comprobar todos los casos, funcionan como se esperan.



Pruebas caja negra

Para realizar estas pruebas, iremos realizando comprobaciones a las diferentes funciones establecidas en la aplicación, por ejemplo, a la hora de actualizar el estado, que cambie la vista o que muestren los datos correctamente. Para probar la api utilicé el post-man que permite realizar peticiones y visualizar la respuesta.

Manual de despliegue

Para comenzar tendremos que tener instalado node, en mi caso es la versión 17.9.0. Podemos instalar node con la terminal usando `sudo apt install nodejs`. También instalaremos npm. `sudo apt install npm`.

Podemos comprobar la versión de node con `node -v`.

El siguiente paso es bajar el proyecto del repositorio de github. Este contendrá tanto el proyecto como la base de datos. La base de datos la podemos importar desde mysql workbench fácilmente.

La **contraseña de acceso a la base de datos** es: Studium2022;

Si queremos cambiar la configuración de la conexión de la base de datos de nuestra aplicación tendremos que editar en la carpeta config, el fichero `bd.js`, donde están los datos de conexión.

Una vez bajado el proyecto e importada la base de datos abriremos una terminal en el directorio donde tenemos el proyecto y ejecutaremos el comando:

```
npm run dev
```

Para arrancar el proyecto. Este iniciará en el puerto 3000.

Cuenta con **un usuario administrador** con nombre root y contraseña root

Repositorio github:

https://github.com/ivantriguero/Proyecto_Integrado

Bibliografía

- <https://www.udemy.com/course/react-guia-definitiva-hooks-router-redux-next-proyectos/learn/lecture/28377500#questions>
- <https://www.hostinger.es/tutoriales>
- <https://stackoverflow.com/>
- <https://nextjs.org/>
- <https://tailwindcss.com/>
- https://www.youtube.com/watch?v=dPXqzmzVm_2I&ab_channel=Skillthrive
- <https://www.npmjs.com/package/jsonwebtoken>
- <https://www.npmjs.com>