ToDo Manager

Por Hugo Méndez González

Hugo Méndez González
2º DAW

Índice

Hugo Méndez González
2º DAW

Introducción

ToDo Manager es una aplicación web dedicada a organizar tareas como si se tratara de post-its en una nevera. El usuario podrá crear, editar y eliminar proyectos que dentro contendrán las tareas, que también podrá tratar el mismo.

El objetivo principal de esta aplicación es la organización de forma cotidiana basada en tareas, que estarán divididas en proyectos para que se pueda diferenciar el tema general sobre el que traten.

Para la realización del proyecto he hecho uso de varias tecnologías en las que he trabajado a lo largo de este curso en las diferentes asignaturas por las que he tenido que pasar :

- Referente a la asignatura de Desarrollo Web en Entorno Servidor he utilizado el lenguaje PHP para realizar las comunicaciones pertinentes entre la aplicación web y la base de datos, y el entorno XAMPP para alojar la base de datos en un servidor al que poder acceder.

- He usado el lenguaje de marcas HTML y Bootstrap para el diseño de las páginas web, aprendidos en la asignatura Diseño de Interfaces Web.

- JavaScript para la programación lógica requerida en el proyecto relacionada con el navegador y el cliente, sobre el cual he adquirido conocimiento en la asignatura Desarrollo Web en Entorno Cliente.

- El uso de GitHub para subir el repositorio del proyecto, sobre el que he adquirido conocimientos gracias a la asignatura Despliegue de Aplicaciones.

- He montado todo el núcleo de la aplicación en un proyecto de React, tecnología que he aprendido por mi cuenta basándome en tutoriales, vídeos y páginas en Internet.
  He creado un repositorio en GitHub dónde está toda la información y todos los archivos relacionados a mi aprendizaje.

Agradezco al Colegio Montecastelo y, en especial, a mis profesores Tacio Camba, Martín García, Daniel Sánchez y Andrés Luna por la formación que me han procurado durante este curso y las facilidades que me han proporcionado para realizar este proyecto.

Desarrollo

Análisis

Para mi aplicación he planteado y ejecutado los siguientes requisitos que un cliente podría necesitar :

- **Entidades de datos.**
  Se necesitará crear entidades de datos relacionadas entre ellas para que el usuario las pueda ver y tratar, como la creación de una base de datos con la estructura necesaria.

- **Tratado de los datos.**
  Tal y como se listan los datos también tienen que ser tratados, por lo que el usuario también podrá modificarlos a su antojo según su rol.

- **Gestión de usuarios.**
  Un cliente necesitará diferentes usuarios para dividir los proyectos en diferentes cuentas de usuario, ya que puede tratarse de que varias personas vayan a usar la aplicación.

- **Roles de usuario.**
  Ya que en la aplicación habrá gestión de usuarios, se necesitarán roles que den privilegios a los mismos. Se podrá contar con dos roles : el usuario normal, que simplemente podrá modificar sus propios datos, proyectos y tareas; y el administrador, que podrá consultar todos los usuarios que hay registrados y eliminarlos si lo considera oportuno.

- **Página de uso del administrador.**
  Al contar con administradores, es útil que tengan un centro de mando desde el que manejar acciones comunes acordes a sus privilegios.

- **Registro e inicio de sesión.**
  Debido a que contaremos con usuarios, también habrá que contar con que una persona se pueda registrar y posteriormente iniciar sesión en su cuenta.
  Por lo que cada usuario tendrá unas credenciales con las que tenga que registrarse y después iniciar sesión en la aplicación.

- **Notificación de registro.**
  Nunca está de más que se le notifique al usuario mediante correo electrónico el hecho de que su cuenta haya sido creada, por lo que cuando un usuario se registre se le notificará de dicha acción vía correo electrónico.

- **Mantenimiento de la sesión.**
  Habrá un mantenimiento de la sesión del usuario que la haya iniciado, para que cuando vuelva a acceder a la aplicación no tenga que volver a escribir sus credenciales y autorizarse. Contará con variables guardadas en el almacenamiento local del navegador que faciliten su funcionamiento cuando el usuario vuelva a abrirla con la sesión iniciada.

- **Diseño intuitivo.**
  Este tipo de aplicaciones un punto a tener muy en cuenta es el diseño, es importante que el usuario pueda localizar todos los elementos con los que puede interactuar y este es el objetivo de un diseño que le ayude a ello.

- **Listado de datos.**
  El cliente necesitará ver los datos listados para después poder trabajar con ellos.
  Por lo que habrá listas de las diferentes entidades de datos como proyectos, usuarios o tareas.

Diseño

He optado por que la aplicación se despliegue mediante un proyecto de React.
Ésta se divide en dos grandes bloques como son el backend y el frontend.

Backend

Lo primero que hay que tener en cuenta es que un requisito para que el proyecto funcione es tener el entorno XAMPP instalado en el equipo para tener un servidor en funcionamiento que sirva la base de datos.

Una vez esté instalado, hay que enfocarse en la base de datos “todomanager” a generar gracias a un archivo SQL de creación que he programado.

La estructura de la base de datos está conformada por las siguientes tablas :

Privilegios

ID	**Nombre**
1	Lectura
2	Lectura + Escritura
3	Lectura + Escritura + Sobre otros usuarios

Tabla_Privilegios

La tabla **Privilegios** se encarga de gestionar los diferentes privilegios disponibles para los roles, se compone de :

- Una ID autogenerada y única como clave primaria
- Un nombre para identificar mejor al privilegio en cuestión, de tipo cadena

A la hora de crear la base de datos se generarán los privilegios indicados en la tabla.

Roles

ID	**Nombre**	**Privilegios**
ID Autogenerada	Nombre del Rol	IDs de los privilegios que tiene

Tabla_Roles

La tabla **Roles** se ocupa de los roles de los usuarios, componiéndose de :

- Una ID autogenerada y única como clave primaria
- Un nombre asociado al rol
- El/los ID correspondientes según los privilegios que tenga el rol, como clave foránea a la tabla **Privilegios**

Usuarios

ID	Email	Nickname	Password	Rol
ID Autogenerada	Email del usuario	Nickname del usuario	Contraseña encriptada	ID del Rol que posee

Tabla_Usuarios

La tabla **Usuarios** alberga a los usuarios registrados, se compone de :

- Una ID autogenerada y única como clave primaria
- Un email asociado al usuario, como campo único
- Un nickname asociado al usuario, de tipo cadena
- Una contraseña asociada al usuario, encriptada, de tipo cadena
- El ID que el usuario posee según su rol, como clave foránea a la tabla **Roles**

Por defecto se crea el usuario administrador, cuyo email es "admin@admin.com" y su contraseña es "admin".

Proyectos

ID	Usuario_Creador	Nombre	Descripcion	Fecha_ Creacion
Auto ID	ID del usuario creador	Nombre del proyecto	Descripción del proyecto	Fecha de cuando el proyecto fue creado

Tabla_Proyectos

La tabla **Proyectos** se encarga de gestionar los diferentes proyectos en los que las tareas pueden estar metidas, está compuesto de :

- Una ID autogenerada y única como clave primaria
- La ID del usuario creador del proyecto en cuestión, como clave foránea a la tabla **Usuarios**
- Un nombre para el proyecto en cuestión, tipo cadena

Tabla_Tareas

La tabla **Tareas** se encarga de gestionar las tareas, componiéndose de :

- Una ID autogenerada y única como clave primaria
- Un nombre para la tarea en cuestión, de tipo cadena
- Una descripción de la tarea, de tipo cadena
- Una fecha de creación de la tarea
- Una fecha de modificación de la tarea
- Un ID haciendo referencia al proyecto al que pertenece, como clave foránea a la tabla **Proyectos**
- Un ID haciendo referencia la tarea padre de la tarea en cuestión, en el caso de tener uno estaríamos hablando de una subtarea
- Un binario (o booleano) indicando si la tarea está pendiente o finalizada

Para manejar las tablas de esta base de datos he programado una serie de archivos en PHP que hay que colocar en la carpeta que usa XAMPP para acceder a ficheros de servidor, su ruta es C:\xampp\htdocs.

Dichos archivos se componen de la siguiente estructura :

- Archivos generales : Constantes usadas por todos los archivos, salvar errores de peticiones CORS y funciones usadas por todos los archivos.

- Archivos que recogen y tratan las peticiones al servidor por parte de la app, simulando una API.

- Archivos que contienen las funciones PHP que manejan la base de datos y efectúan los cambios pedidos gracias a la petición a la API.

Frontend

Se basa en la estructura del proyecto de React, sus funcionalidades generales son :

- Contiene una página de registro y una página de login para la autorización del usuario.

- Mantiene la sesión del mismo guardando sus credenciales necesarias en el almacenamiento local del navegador y en el contexto de la aplicación.

- Las demás páginas contarán con una barra de navegación que tendrá la función de llevar al usuario a las rutas principales de la aplicación.

- El usuario ve mostradas las entidades de datos pertinentes para poder tratarlas.

- Permite realizar operaciones CRUD al usuario con sus proyectos y tareas.

- El usuario puede cambiar sus propias credenciales en la página referente a su perfil.

- Si el usuario que haya iniciado sesión es un administrador, podrá acceder al panel del administrador, donde podrá consultar y eliminar los demás usuarios registrados en la base de datos.

Implementación

Backend

Tal como he explicado en el capítulo anterior, los archivos PHP que componen el backend se pueden dividir entre grupos basados en la función principal que realizan.

En el archivo *Utils.php* cuento con la función que se dedica a realizar la conexión a la base de datos, y un par de funciones que realizan las acciones pertinentes en el caso de que la función que actúa con la base de datos resulte satisfactoria o en error, respectivamente.

```php
// Compruebo el resultado de la ejecución de la sentencia y devuelvo un booleano según corresponda
return comprobarResultadoDeQuery($conexion, $sentencia);
```

Lugar en el que pongo en práctica una de estas funciones.

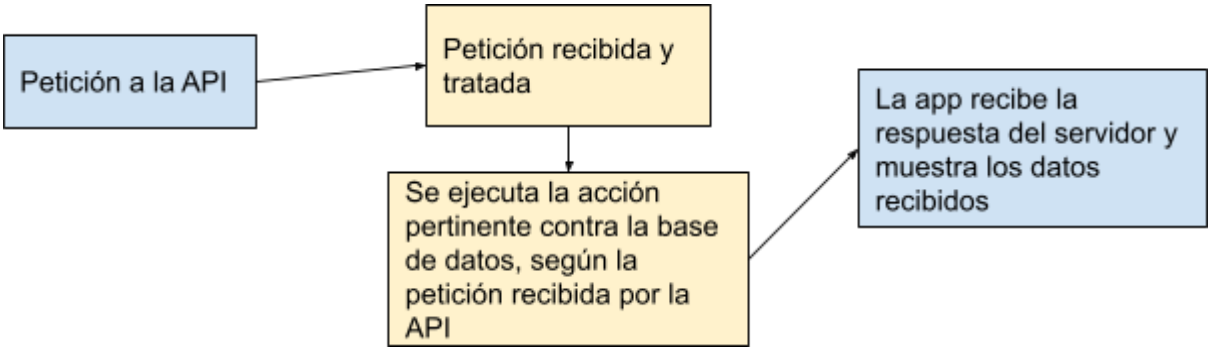
Para los archivos que recogen las peticiones a la API, los he dividido basándome en la entidad de datos a la que van a afectar como pueden ser usuarios, proyectos, tareas o roles.

Un ejemplo de tratado de petición a la API sería el siguiente :

```php
if (isset($_GET["listaUsuarios"])) {
    echo leerUsuarios($conexionBBDD);
}
```

Recoge la petición y ejecuta en este caso una función que devuelve con un *echo* la lista de los usuarios registrados en la base de datos.

El funcionamiento de la app a la hora de las peticiones al servidor sería el siguiente :
El color azul representa el frontend, el color amarillo representa el backend.

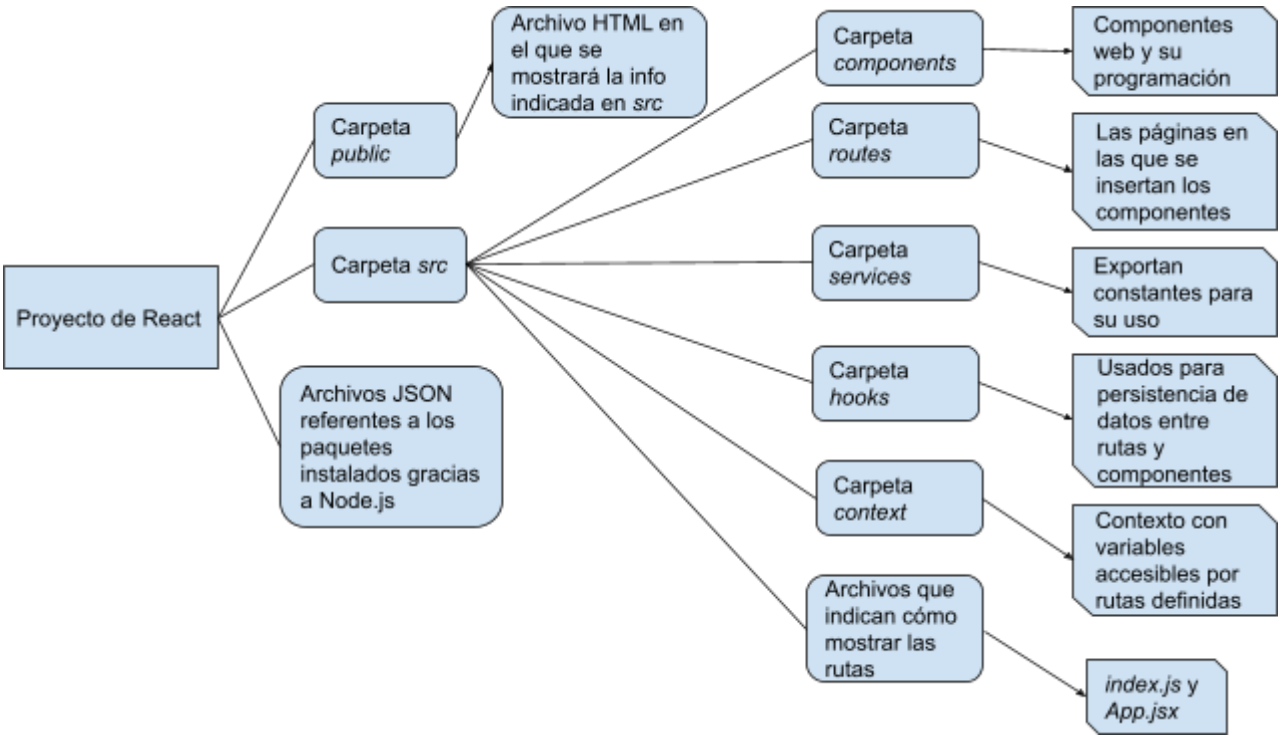


Esquema_Funcionamiento_Peticiones

Frontend

Está basado en React y la estructura interna en la que se rige, programando en JavaScript y en JSX.

Dicha estructura se ve representada así :



Esquema_Funcionamiento_React

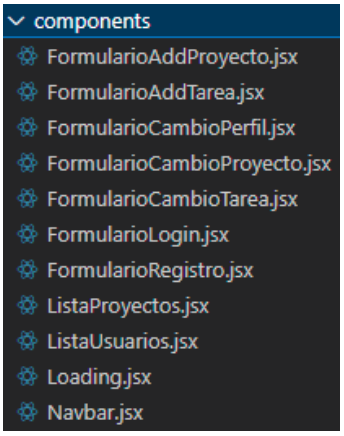
React es una biblioteca de JavaScript para construir interfaces de usuario, que usa Node.js como sistema para ejecutar las diferentes acciones que se pueden realizar como crear un proyecto, instalar un paquete Node en un proyecto, lanzar la versión de desarrollo del proyecto en el navegador, etc…
En este proyecto estoy usando los paquetes React Router DOM, Sweet Alert 2 y Axios.

JSX es una extensión de la sintaxis de JavaScript en la que se permite el uso de etiquetas con atributos al estilo de XML.
En React se usa para definir cómo va a ser la interfaz de usuario, pone el maquetado de la web y su lógica en el mismo archivo.

```jsx
<div className="d-flex justify-content-center"> {/* Centro el spinner */}
    <div className="spinner-border" style={spinnerStyle} role="status"> {/* Establezco el estilo del spinner */}
        <span className="visually-hidden">Loading...</span>
    </div>
</div>
```

Maquetado en JSX de un spinner.

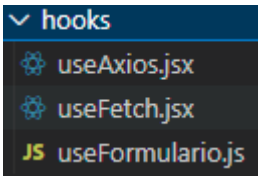
React no requiere el uso de JSX, pero es de gran utilidad a la hora de usar dicha librería. Una de las principales funcionalidades de React es poder separar los componentes de la web en archivos independientes, pudiendo ser reutilizados si así se desea.



Como ejemplo, estos son los componentes de React usados en mi proyecto.

La norma es que el nombre del archivo referente a un componente debe empezar siempre por mayúscula.

React también cuenta con otras funcionalidades como el uso de hooks para la persistencia y el trato correcto de datos.



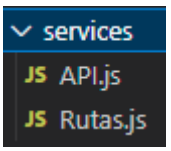
La norma es que el nombre del archivo referente a un hook personalizado debe empezar siempre por “use”.
En mi caso, tengo hooks para manejar y recoger datos del uso de formularios y de peticiones a la API.

Existen hooks predefinidos por React como *useState*, *useEffect* o *useContext*. También se pueden crear hooks personalizados para realizar una acción concreta requerida.

Otra funcionalidad principal de React es el uso de contextos, archivos JSX en los cuales se declara y se gestiona algún tipo de lógica (variables, funciones, etc…) y después se indica en el archivo *index.js* qué rutas pueden acceder a dichas entidades.

Para el acceso al contexto existe el hook nativo *useContext*, los archivos donde un contexto es definido, deben de tener un nombre que acabe por “Provider”.

He utilizado constantes guardadas en archivos JS para usar URLs de la API y los nombres de las diferentes rutas de la aplicación, así evito números o textos mágicos y puedo acceder de manera más cómoda a las rutas.



Archivos en los que se guardan constantes referentes a las rutas del proyecto.

Para hacer efectivo el mantenimiento de sesión del usuario, he utilizado el almacenamiento local del navegador para guardar la ID, el nickname y el rol del mismo.
También establezco las mismas variables en el contexto del proyecto, ya que es más fácil acceder a ellas mediante ese modo.

Este es el fragmento de código donde asigno los datos del usuario al local storage y al contexto :

```
localStorage.setItem("ID", response.data.usuario.id); // Establezco la ID en el localStorage
id = response.data.usuario.id; // Establezco la variable referente al ID en el contexto

localStorage.setItem("nickname", response.data.usuario.nickname); // Establezco el nickname en el localStorage
nickname = response.data.usuario.nickname; // Establezco la variable referente al nickname en el contexto

localStorage.setItem("rol", response.data.usuario.rol); // Establezco el rol en el localStorage
rol = response.data.usuario.rol; // Establezco la variable referente al rol en el contexto
```

En el mismo archivo defino una función dedicada a comprobar si el usuario ha iniciado sesión para acceder a la ruta en la que se encuentre, así evito que alguien sin iniciar sesión pueda acceder a las páginas de la aplicación.

```
/**
 * Comprueba si están presentes los datos de inicio de sesión
 * Y en caso de que no lo estén, vuelve a la página de login
 */
const comprobarLogin = () => {
  // Compruebo si no están guardados los datos de la sesión
  if (id == "" && nickname == "" && rol == null) {
    navigate("/"); // Y vuelvo a la página del login
  }
}
```

Comprueba si el valor de las variables del contexto está definido, y en el caso de que no lo esté vuelve a la página de login.

Para listar las diferentes entidades que el usuario puede visualizar en las páginas he usado la función *fetch()* para realizar la petición a la API.
Como en JSX se puede escribir código lógico dentro de una etiqueta, he rellenado las listas utilizando esa misma técnica.

```
{ // Listo todos los usuarios que haya para que se muestren
  data.map(item =>
    <tr key={item.id}> {/* Uso los nombres de los campos en la BBDD MySQL */}
      <th scope="row" className="ps-0">{item.id}</th>
      <td className="ps-0">{item.email}</td>
      <td className="ps-0">{item.nickname}</td>
      <td className="ps-0">{item.pwd}</td>
```

Ejemplo del listado de usuarios en un componente JSX.

En cuanto al diseño y el estilo de las páginas, esta es la estructura principal :

```
<div>
  <Navbar /> {/* Llamo al componente Navbar */}

  <div style={{height: "85vh", overflowX: "none"}}>
    <Outlet /> {/* Con esta etiqueta indicamos que aquí se debe pintar la ruta en la que esté */}
  </div>

  {/* Footer */}
  <footer className="bg-light text-center text-lg-start">
    <div className="text-center p-3"> Hugo Méndez © 2022 </div>
  </footer>
</div>
```

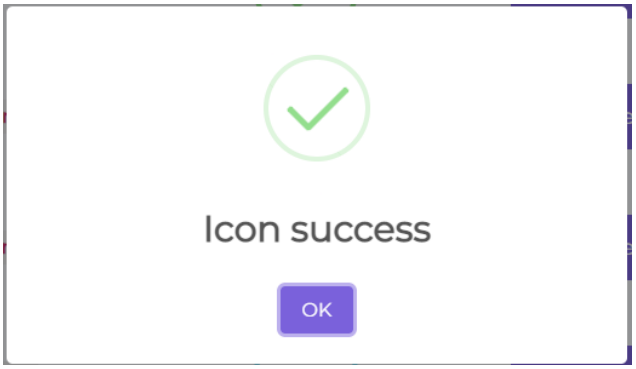
Se mostrará un Navbar en la parte superior del navegador, inmediatamente más abajo se mostrará toda la información de la ruta en la que el usuario se encuentre y finalmente en la parte más baja encontraremos un footer.

Paquetes Node en el proyecto

El paquete más importante de los instalados es React Router DOM, React Router es una biblioteca de enrutamiento del lado del servidor y del cliente con todas las funciones para React. Se puede ejecutar en cualquier lugar donde se ejecute React.

Este paquete me permite definir rutas y establecer su acceso, usar enlaces entre las mismas y establecer la vista de las diferentes páginas del proyecto a efectos generales.

Otro paquete de Node.js instalado es Sweet Alert 2. Se basa en un diseño de alerta interactiva que aporta dinamismo a la interacción del usuario con la aplicación.



Ejemplo de un Sweet Alert que ha resultado satisfactorio.

A la hora de programarlo es bastante fácil de usar. Simplemente hay que importarlo en el archivo en el que se necesite, llamar a la función *fire()* para mostrar la alerta y establecer sus parámetros.

```
Swal.fire({ // Muestro el modal indicando éxito
  title: 'Éxito!',
  text: ""+message,
  icon: 'success',
  showConfirmButton: false,
  timer: 1500
});
```

Ejemplo de cómo muestro un Sweet Alert satisfactorio en el proyecto.

El tercer paquete usado en este proyecto es EmailJS y su función es mandar un correo electrónico a la dirección que se desee. Es necesario crear una cuenta en su página web para poder generar un plugin que permita mandar el correo, y después en el código sólo hay que importarlo en el archivo requerido y especificar los códigos tanto de la plantilla como del usuario.

```
emailjs.send("service_127gzsg","template_qoen1zh",{to_name: txtEmail.trim()}, "EivE0959jTxI2uoZq"); // Mando un email
```

Ejemplo de llamada a la función para enviar un email usando este paquete.

El último paquete utilizado se llama Axios, es un Cliente HTTP basado en peticiones para Node.js y el navegador.
En el lado del servidor usa el módulo nativo *http* de Node.js, mientras que en el lado del cliente usa XMLHttpRequests.

Para utilizarlo hay que importarlo en el archivo que se quiera y llamar a sus funciones disponibles para realizar peticiones como *get()* y *post()*.

Pruebas

Backend

Debido a las pruebas realizadas, he tenido que incluir un archivo PHP para corregir los errores CORS y que permitan peticiones desde el proyecto de React.

En todas las funciones PHP he comprobado el caso en el que salte un error debido a que el tipo de dato que se está intentando introducir no corresponde con el tipo de dato que debe tener el campo, y he controlado cualquier tipo de error referente a la base de datos para que devuelva un código que se interpretará como un error.

A la hora de eliminar una entidad que esté relacionada con otra (como puede ser un proyecto, relacionado con tareas) he controlado las acciones a realizar para que se eliminen todas las entidades relacionadas con la que se está eliminando.
Por ejemplo, a la hora de eliminar un proyecto se eliminarán también todas las tareas que éste comprenda. O a la hora de eliminar un usuario, se eliminarán todos sus proyectos y por lo tanto también todas sus tareas.

Gracias a las funciones que he programado en el archivo *Utils* me ha sido más fácil la comprobación del resultado de las consultas, ya que sólo tengo que llamar a la función correspondiente para saber si ha resultado satisfactorio.

Para que funcione correctamente con la API, he tenido que convertir a JSON todos los datos que son devueltos para que los pueda interpretar correctamente en el proyecto de React.

Frontend

Debido al mal funcionamiento de la función *fetch()* en algunos lugares del código, he tenido que utilizar el paquete *Axios* explicado anteriormente para realizar las peticiones a la API de manera satisfactoria.

He añadido comprobaciones en todas las páginas pertinentes para que nadie sin identificarse pueda acceder a la aplicación.

He añadido mensajes de error para mostrar cuando el usuario intente realizar una acción que no sea posible, como introducir un proyecto sin un nombre.

Para un mejor diseño, he programado un componente referente a un spinner de Bootstrap para que se muestre mientras la petición a la API no ha obtenido una respuesta.
He tenido que controlar los datos devueltos por las peticiones a la API para que no haya errores a la hora de querer mostrar los datos cuando aún no se han obtenido.

Conclusiones

Autentificación de usuarios y mantenimiento de la sesión de los mismos :
He creado páginas referentes a el registro y el inicio de sesión de un usuario con las comprobaciones pertinentes para ello.

Para el mantenimiento de la sesión he decidido guardar los datos del usuario con sesión iniciada en el almacenamiento local del navegador, para que cuando se vuelva a acceder a la aplicación el usuario no tenga que volver a iniciar sesión.

También he incluido comprobaciones sobre si el usuario ha iniciado sesión en cada página, para que nadie pueda acceder a páginas desde la URL sin estar autenticado.

Gestión de usuarios :
Mi aplicación comprende usuarios administradores que pueden consultar y eliminar otros usuarios, y usuarios sin privilegios que simplemente podrán gestionar sus propios datos.

En la aplicación se pueden crear usuarios mediante su registro. Un administrador puede crear más administradores.

Gestión de las entidades de datos de cada usuario :
La aplicación permite crear, listar, editar y eliminar proyectos y tareas a cada usuario, sólo pueden acceder a las entidades que ellos mismos hayan creado.

Diseño de la parte gráfica :
Se ha realizado el diseño con Bootstrap, y se ha hecho tanto funcional como usable para que el usuario sepa en todo momento lo que puede hacer.

Validación de la entrada de datos :
Se valida cualquier introducción de datos por parte del usuario desde el lado cliente y el lado servidor, y se muestra el mensaje de error correspondiente en el caso de que haya algo mal.

Objetivos personales :

Me había propuesto un diseño para la página principal en el que se mostraran los proyectos en una columna a la izquierda de la pantalla y a su derecha en un área mucho mayor se mostraran las tareas del proyecto seleccionado.
Finalmente no tuve el suficiente tiempo como para obtener dicho resultado y tuve que separar las vistas en dos páginas diferentes, cuando se haga click en el nombre de un proyecto la aplicación se irá hasta otra página donde estén listadas las tareas de dicho proyecto.

Otro objetivo propuesto fue el conseguir un diseño más interactivo con el usuario, con animaciones y que diera la sensación de dinamismo.
Debido al tiempo utilizado para el buen funcionamiento de la aplicación, no pude conseguir dicho acabado estético.

Líneas Futuras

Implementación de subtareas :
La base de datos está construida para poder soportar subtareas de unas tareas padre, pero no tuve el tiempo suficiente para implementarlo.
En un futuro se podría añadir un botón a cada tarea listada, que lleve a otra página en la que se puedan ver sus subtareas y poder realizar operaciones CRUD con ellas.

Cambiar el tipo de dato de las IDs :
Otra idea a implementar en un futuro es cambiar el tipo de dato de las IDs de las diferentes entidades por el formato UUID (Universally Unique Identifier), para obtener una mayor seguridad en la base de datos y más profesionalidad a la hora de tratar con este tipo de datos.

Mejor diseño para la página principal del usuario :
En vez de que haya dos páginas diferentes en las que se muestre la lista de proyectos y en la otra la lista de tareas, incluir ambas listas en una sola página.

Bibliografía

Repositorio de GitHub donde he subido todos los archivos referentes a este proyecto	https://github.com/hugo3011mendez/ProyectoDAW
Repositorio de GitHub donde he subido toda la información sobre mi aprendizaje de React	https://github.com/hugo3011mendez/Aprendiendo-React
Bootstrap	https://getbootstrap.com/
React	https://es.reactjs.org/
XAMPP	https://www.apachefriends.org/
JSX	https://es.reactjs.org/docs/introducing-jsx.html
Node.js	https://nodejs.org/es/
React Router	https://reactrouter.com/en/main
Sweet Alert 2	https://sweetalert2.github.io/#download
EmailJS	https://www.emailjs.com/
Axios	https://axios-http.com/es/docs/intro

Anexos

Tabla_Tareas

ID	Nombre	Descripcion	Fecha_ Creacion	Fecha_ Modificacion
Auto ID	Nombre de la tarea	Descripción de la tarea	Fecha y hora en formato DateTime	Fecha y hora en formato DateTime

Proyecto	ParentID	Estado
ID del proyecto al que pertenece	ID de su tarea padre (si tiene)	Binario, 0 si está pendiente, 1 si está finalizada