

Proyecto de Investigación

2 DAW



Ángel Ríos Minaya

LICENCIA

Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional



Esta no es una licencia de Cultura Libre.

Indice

1. Marco de la investigación.....	5
1.1 Temática elegida.....	5
1.2 Contextualización.....	5
1- React.....	5
2- Firebase.....	7
2. Organización del proyecto.....	9
2.1 Temporalización.....	9
1- Iniciación en React.....	9
2- Conocimientos básicos.....	9
3- Aplicación de componentes un poco más complicados.....	9
4- Iniciación en Go y MongoDB.....	10
5- Problemas con Go.....	10
6- Firebase.....	10
7- Inicio de la memoria.....	11
8- Implementación de Redux en el proyecto.....	11
9- Creación del anuncio y exclusión de Redux.....	11
10- Inclusión de los test unitarios en la aplicación.....	11
11- Refactorización del código.....	11
2.2 Recursos.....	12
1- Lenguajes:.....	12
2- Librerías:.....	13
3. Aplicación práctica.....	16
3.1 Introducción.....	16
3.2 Ciclo de vida.....	16
3.2.1 Análisis de requerimientos.....	16
3.2.2 Diseño.....	24
3.2.3 Implementación.....	24
3.2.4 Pruebas.....	38
4. Manual De Usuario.....	43
5. Valoración Personal.....	47
6. Agradecimientos.....	48
7. Fuentes Bibliográficas.....	49

1. Marco de la investigación.

1.1 Temática elegida

Para la elaboración de este proyecto de investigación he elegido dos tecnologías.

La primera la Librería React y la segunda Firebase Cloud Firestore .

Ahora pasaré a explicar el por qué estas dos tecnologías y no otras.

Escogí React porque quería aprender a utilizarlo, ya que está en crecimiento y están empezando a salir muchos proyectos, y a parte, también fue un gran factor que en las prácticas se me asignó un proyecto en el que estaban trabajando con React y esto fue el punto para decantarme por investigarlo.

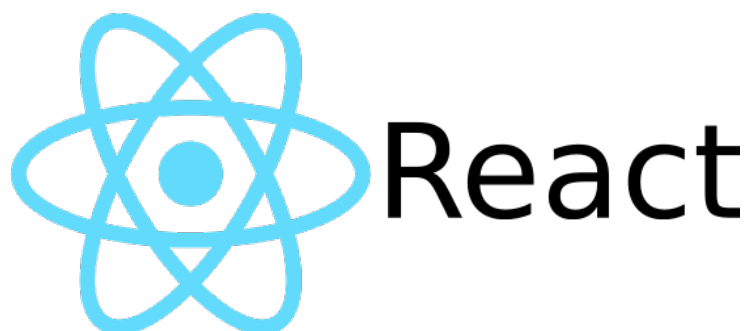
La segunda tecnología al principio era Go pero debido a la falta de tiempo me decanté por Firebase y en concreto aprendí a usar Firebase Cloud Store que funciona por colecciones como por ejemplo MongoDB.

1.2 Contextualización

1- React.

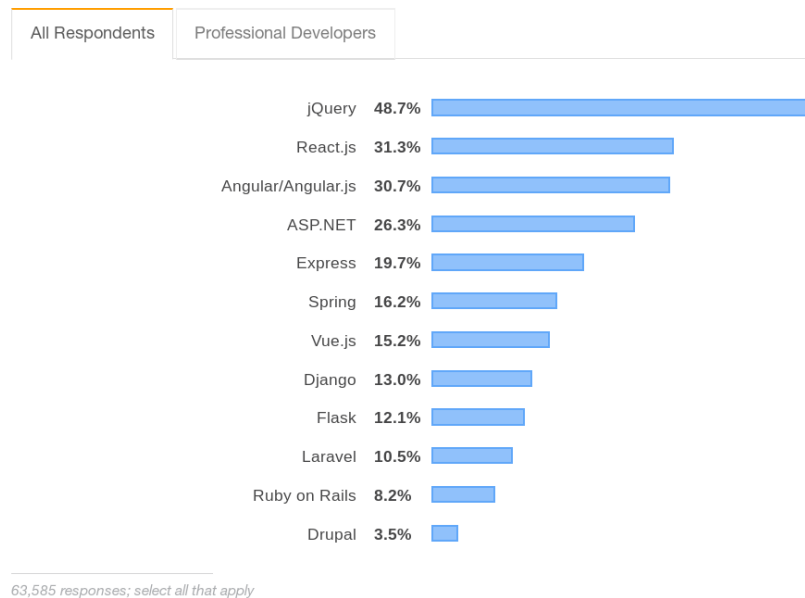
Como he mencionado en el anterior punto, React es una librería de JavaScript que se centra en la creación de interfaces de usuario. Pero esto no significa que no se puedan hacer aplicaciones web con React ya que también podemos hacer páginas web completas o aplicaciones móviles.

React es de código abierto y esto ha hecho que crezca mucho la comunidad de desarrolladores, el inicio del desarrollo fue hecho por Facebook y actualmente React es el corazón de Facebook.

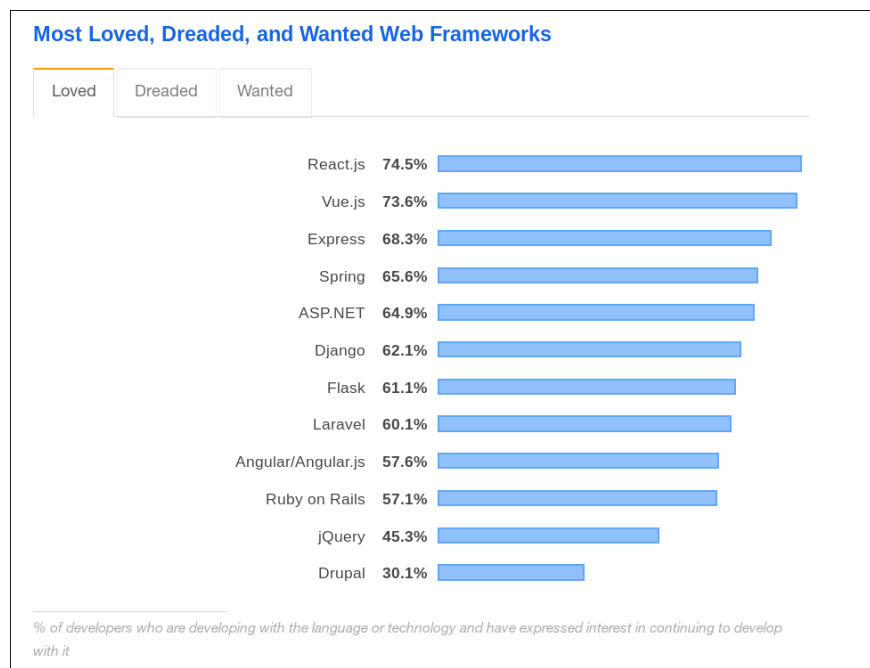


Como podemos ver en la siguiente imagen, React es la segunda tecnología mas popular en el desarrollo web y esto fue también un buen punto para decantarme por esta herramienta para mi proyecto de investigación.

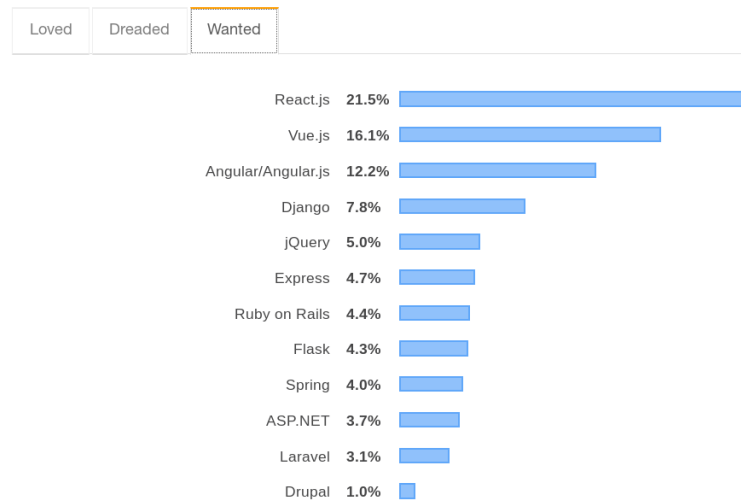
Web Frameworks



A parte de ser la segunda herramienta más popular para la creación de aplicaciones Web, también es la herramienta más querida y más amada por los programadores según la encuesta de [Stack Overflow Survey 2019](#).



Most Loved, Dreaded, and Wanted Web Frameworks



% of developers who are not developing with the language or technology but have expressed interest in developing with it

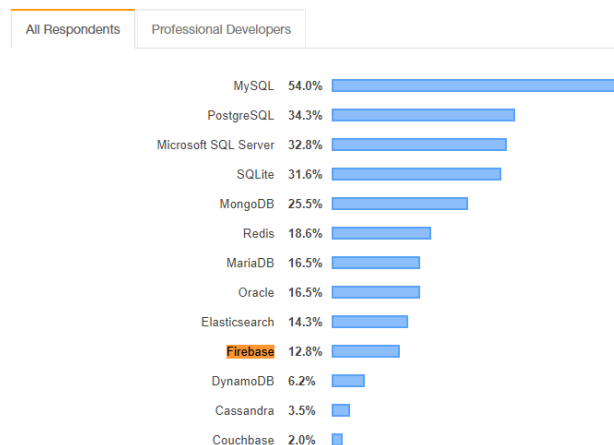
Algunas de las alternativas para esta tecnología serían Angular y Vue que también están en puestos elevados de popularidad y son de las más queridas por la comunidad de programadores.

2- Firebase.

Escogí Firebase porque después de llevar algo de tiempo con Go y ver que no tenía tiempo decidí cambiar a esta tecnología.

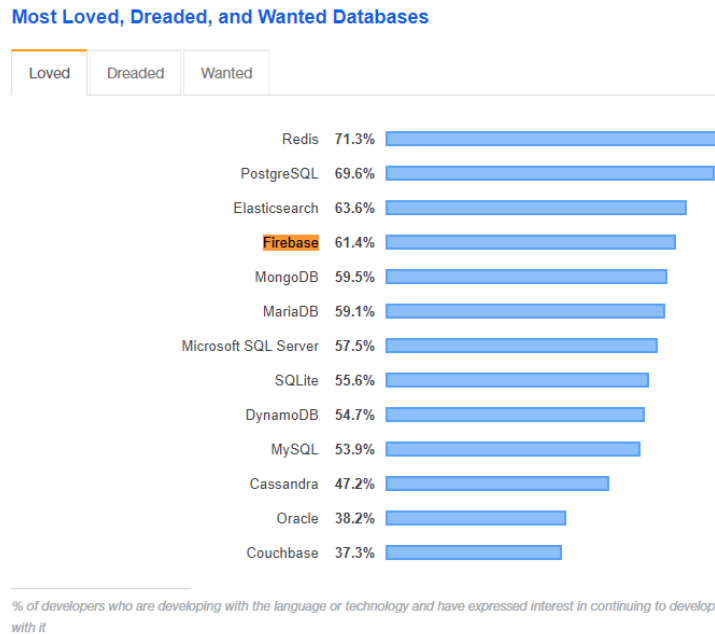
Actualmente según las encuestas de [Stack Overflow Survey 2019](#), Firebase no es una de las bases de datos más populares. Por delante de ella se encuentran, por ejemplo, MongoDB, MariaDB o MySQL lo cual son alternativas muy buenas a esta tecnología.

Databases



75,023 responses; select all that apply

Como podemos seguir viendo en la siguiente imagen Firebase si que está entre las mas queridas y amadas por la comunidad.



2. Organización del proyecto.

2.1 Temporalización.

Mis nociones de React y Firebase eran nulas cuando decidí empezar a hacer Listen Me así que aprovechando las vacaciones antes de empezar las prácticas empecé a investigar sobre estas tecnologías.

A continuación enumeraré por semanas el camino que hice para poder completar este proyecto:

1- Iniciación en React.

Como menciono antes, mi concepto sobre React era nulo así que empecé por lo mas simple que era como crear un proyecto en React.

Más adelante empecé a buscar información sobre react-router para poder ir adelantando y ver como se creaban los sistemas de routing en React.

2- Conocimientos básicos.

En la segunda semana de mi aprendizaje sobre React empecé a implementar la funcionalidad del router. Tuve algunos problemas ya que estuve mirando por Youtube tutoriales que explicasen el Routing pero me fallaban muchas cosas ya que la nueva versión de React se hacia diferente el routing y no funcionaba, entonces recurrí a lo que tendría que haber recurrido primero que es la API de React.

Empecé a crear componentes básicos como el navbar y algunas páginas.

3- Aplicación de componentes un poco más complicados.

Ya que mi aplicación es sobre música pues necesitaba un componente para reproducir los videos y decidí usar la etiqueta propia de html para crear un reproductor de vídeo.

Lo complicado de esto era que no sabía como pasar al componente un grupo de vídeos para que se creasen automáticamente solo con un componente y dedicándole un poco de tiempo, al final encontré una solución por mi mismo y no buscando la información sobre esto.

Empecé a implementar los componentes de las páginas y a mirar como funcionaba la librería Axios.

4- Iniciación en Go y MongoDB.

Esto fue más difícil que la iniciación en React ya que es un lenguaje totalmente distinto a JavaScript.

Estuve un par de días de la cuarta semana investigando sobre Go, su funcionamiento, que es un lenguaje compilado y que funciona ejecutando un .exe o compilándolo desde consola.

Con MongoDB la cosa iba bien ya que no era muy difícil de entender ni de utilizar, la única dificultad que encontré era en entender como funcionaban las colecciones de esta base de datos.

5- Problemas con Go.

Al estar en las prácticas no podía dedicar mucho tiempo a la investigación sobre Go ya que estaba estudiando React porque me tocó en un proyecto ya bastante avanzado sobre React y para una organización europea muy importante como la EUIPO y no quería retrasar el proyecto así que me puse al cien por ciento a estudiar React.

Estuve pensando si dejar Go de lado pero al final decidí seguir y gracias a una serie de videos en Youtube de iniciación en Go pude hacer una API muy sencilla pero que me podría servir para realizar este proyecto.

6- Firebase.

Debido a los problemas de tiempo y que algunas funcionalidades de la aplicación requerían de un conocimiento mayor en la parte de Go decidí dejar Go de lado y MongoDB y lo sustituí por Firebase.

Firebase no me llevó mucho tiempo aprenderlo era bastante sencillo y como escogí la base de datos Cloud Firestore se me hizo todavía más fácil porque funcionaba con colecciones como MongoDB.

Unos de los problemas que me surgieron con el cambio de MongoDB a Firebase es que tenía que cambiar todas las llamadas que estaban hechas y no solamente bastaba con cambiar la dirección a la que llaman si no quitar Axios y añadir una herramienta propia de Firebase para hacer peticiones a la base de datos.

7- Inicio de la memoria.

Esta semana fue cuando decidí comenzar la memoria del proyecto ya que la aplicación estaba prácticamente acabada.

8- Implementación de Redux en el proyecto.

Después de lo aprendido en las prácticas con el proyecto de la EUIPO decidí implementar Redux en mi proyecto ya que éste te ofrece un estado global para la aplicación mediante la creación de una store.

9- Creación del anuncio y exclusión de Redux.

Quería tener un anuncio sobre mi aplicación web ya que eso le da un toque profesional a la presentación.

Al tener experiencia editando videos para Youtube no tuve ninguna dificultad a la hora de editar el vídeo pero lo más costoso fue encontrar los clips para el anuncio.

Debido a la falta de tiempo decidí no implementar la librería redux en mi proyecto pero aún así he decidido explicarlo un poco en recursos utilizados ya que ha sido parte de mi proyecto de investigación aunque finalmente no se haya utilizado en el proyecto.

10- Inclusión de los test unitarios en la aplicación.

Aquí llegó el punto de pensar si quería implementar en la aplicación tests unitarios ya que quería poner en práctica lo que había aprendido en Sopra.

Al final de esta semana decidí implementarlos de una manera sencilla, no intentando llegar al cien por ciento de cobertura, pero si por lo menos que cada componente llegase a cincuenta por ciento.

11- Refactorización del código.

Después de seguir viendo la aplicación y ver que tenía mucho código basura decidí empezar a limpiar un poco la aplicación.

Es cierto que podría haber refactorizado toda la aplicación y habría quedado el código mucho más limpio pero al final decidí limpiarlo un poco.

2.2 Recursos.

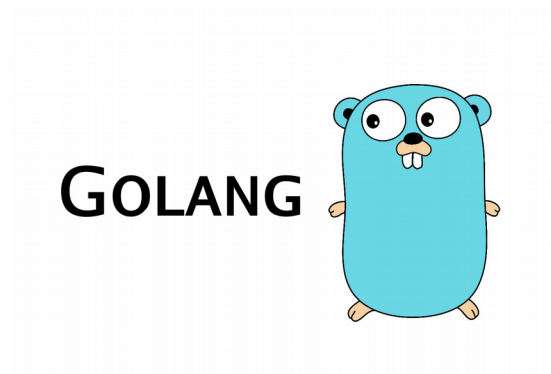
Lista de los recursos utilizados en Listen Me:

1- Lenguajes:



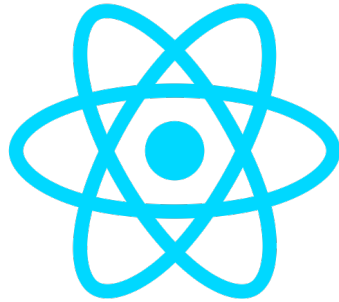
JavaScript – En este caso se utiliza una extensión de JavaScript llamada JSX que es propia de React y gracias a esto podemos mezclar HTML y JS en los componentes.

Go(Excluido) – Go/Golang es un lenguaje pensado para el internet de hoy en día, es decir, un internet rápido.



2- Librerías:

React – Ya que esta herramienta aparece en el apartado de librerías es un momento idóneo para puntualizar que React no es un framework es una librería y con ella esta hecha el cien por ciento de la aplicación.



Redux(Excluido) – Como he explicado en el apartado de la temporalización, esta librería no la he utilizado en el proyecto pero ha formado parte de la investigación ya que invertí tiempo en aprender a usar un poco por encima esta librería.



Jest – Para la parte de los tests unitarios para tener cubierta la aplicación he utilizado la librería [Jest](#).



IDEs:

La herramienta elegida que utilicé para programar es Visual Studio Code ya que es una herramienta gratuita y una de las mejores.



Más tecnologías utilizadas:



git - <https://git-scm.com/>



github - <https://github.com/>

Tecnologías externas a la aplicación:

Adobe Photoshop- Para la creación del logo.



Sony Vegas Pro- He utilizado esta herramienta para hacer el anuncio de la página.



3. Aplicación práctica.

3.1 Introducción

Listen Me es una aplicación web creada con React y Firebase que está enfocada a la gente que quiere empezar en este mundo de la música.

Esta idea surgió por mi gran afición a la música y pensé en una aplicación donde músicos que acaban de empezar puedan subir sus videoclips o canciones.

3.2 Ciclo de vida

3.2.1 Análisis de requerimientos

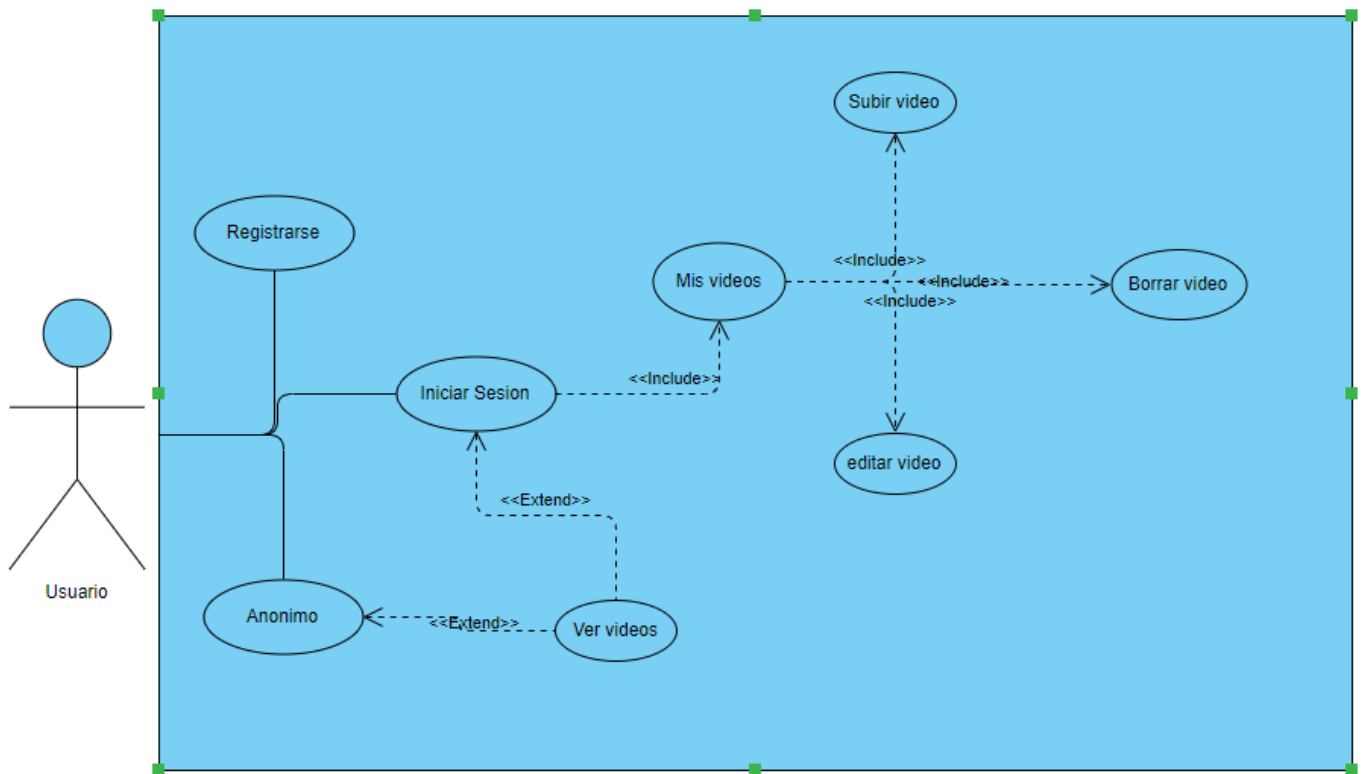
1 - Descripción global de la aplicación.

Listen Me es una plataforma web para músicos que acaban de empezar que quieran darse a conocer pudiendo subir sus canciones y compartiendo sus redes sociales.

En la aplicación hay dos tipos de usuarios pero cualquier usuario puede ejercer un tipo o los dos.

Usuario	<ul style="list-style-type: none">• Escuchar música.• Conocer a nuevos artistas.
Artista	<ul style="list-style-type: none">• Subir canciones.• Promocionar redes sociales.

El primer tipo de usuario es el usuario básico, que es el que solo accede a la aplicación para escuchar música y conocer a nuevos artistas. El segundo tipo de usuario es el artista que es el que sube las canciones a la plataforma.

2- Diagrama general del caso de uso de la aplicación.

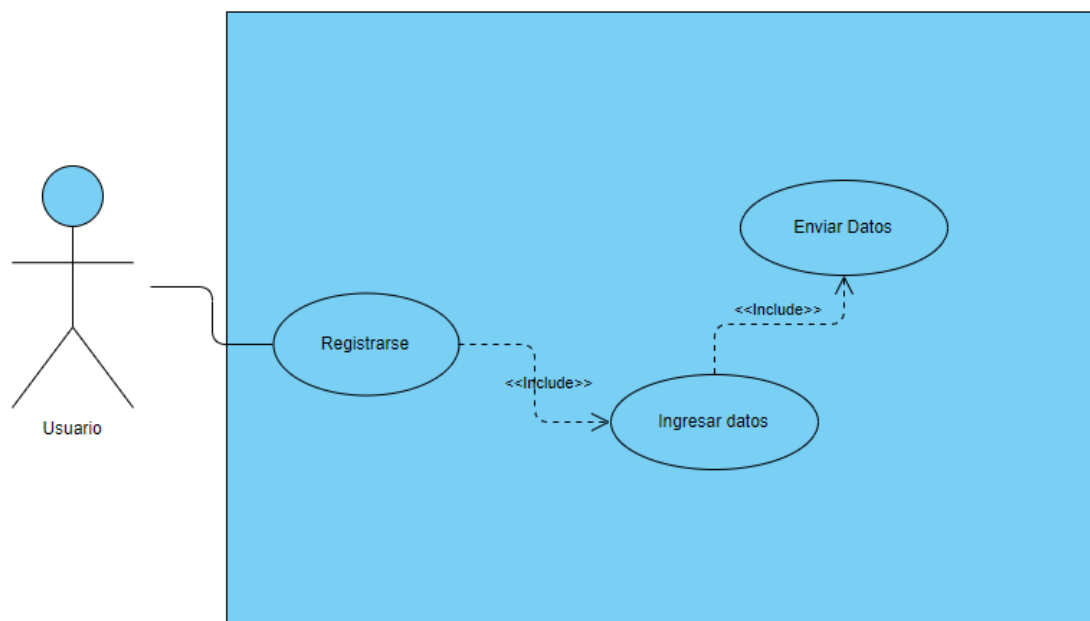
Como podemos observar en el mapa de arriba un usuario puede acceder a la aplicación y iniciar sesión, registrarse o simplemente entrar como anónimo.

A su todos los tipos de usuario pueden ver videos pero solo los usuarios logeados pueden acceder a la parte de mis videos y sus funcionalidades.

3- Casos de uso.

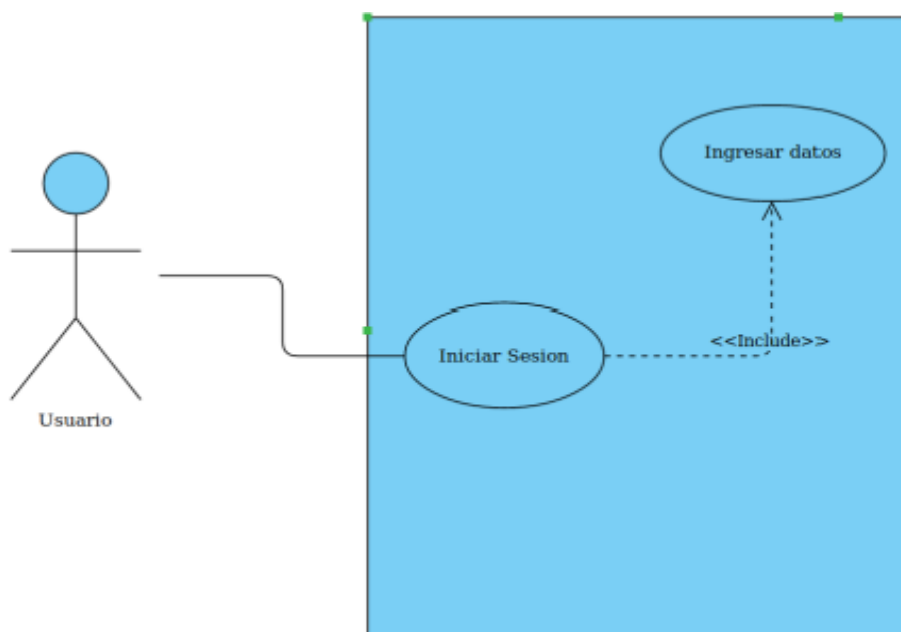
CU01- Registrarse

Descripción:	Registrarse en la página.
Actores:	Usuarios.
Precondiciones:	Estar dentro de la página.
Postcondiciones:	El usuario podrá iniciar sesión.
Flujo Normal:	<ul style="list-style-type: none">• Abre la ventana de registrarse y rellena el formulario.
Flujo Alternativo:	<ul style="list-style-type: none">• Error del servidor.• Error de validación.• Existe el usuario.



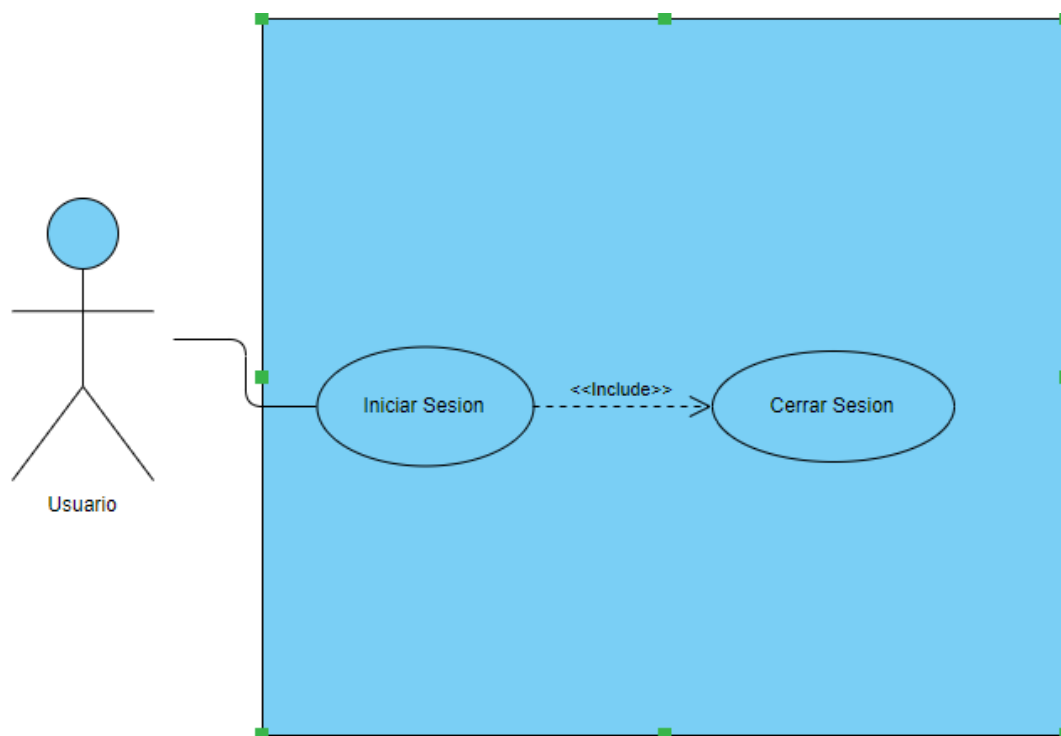
CU02- Iniciar sesión

Descripción:	Iniciar sesión en la aplicación.
Actores:	Usuarios.
Precondiciones:	Estar registrado en la aplicación.
Poscondiciones:	El usuario podrá acceder a las opciones de un usuario.
Flujo Normal:	<ul style="list-style-type: none">• Abre la ventana de iniciar sesión, inserta correo y contraseña y inicia sesión.
Flujo Alternativo:	<ul style="list-style-type: none">• Error del servidor.• Error de validación.• No existe el usuario.



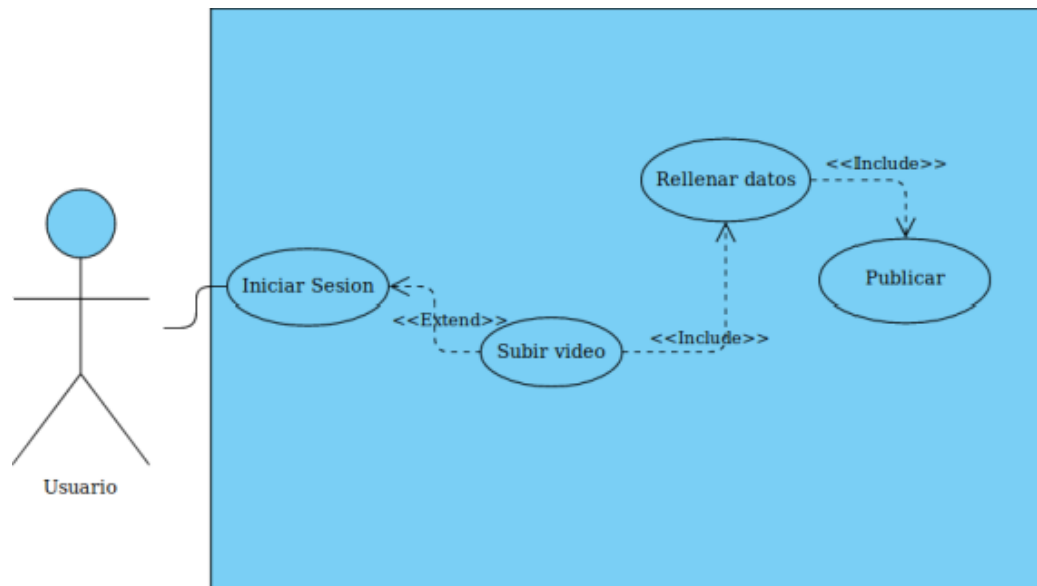
CU03- Cerrar sesión

Descripción:	Cerrar sesión en la página.
Actores:	Usuarios.
Precondiciones:	Estar logeado.
Postcondiciones:	El usuario ya no estará logeado.
Flujo Normal:	<ul style="list-style-type: none">• Selecciona su perfil y hace click en cerrar sesión.
Flujo Alternativo:	<ul style="list-style-type: none">• No se le mostrará la opción porque no está logeado en la aplicación



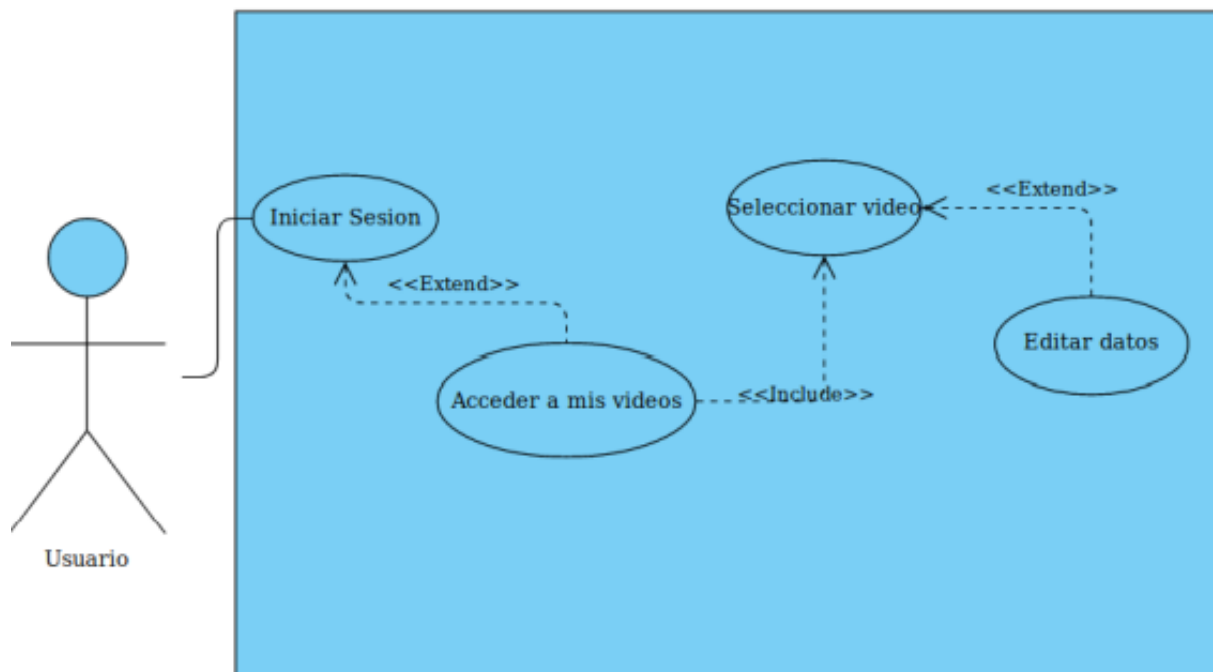
CU04- Subir video

Descripción:	Subir un vídeo a la página.
Actores:	Usuarios.
Precondiciones:	Estar logeado.
Postcondiciones:	Rellenar la información del vídeo.
Flujo Normal:	<ul style="list-style-type: none"> • Seleccionar subir vídeo. • Seleccionar el vídeo. • Rellenar la información del vídeo.
Flujo Alternativo:	<ul style="list-style-type: none"> • Error del servidor. • Formato no correcto.



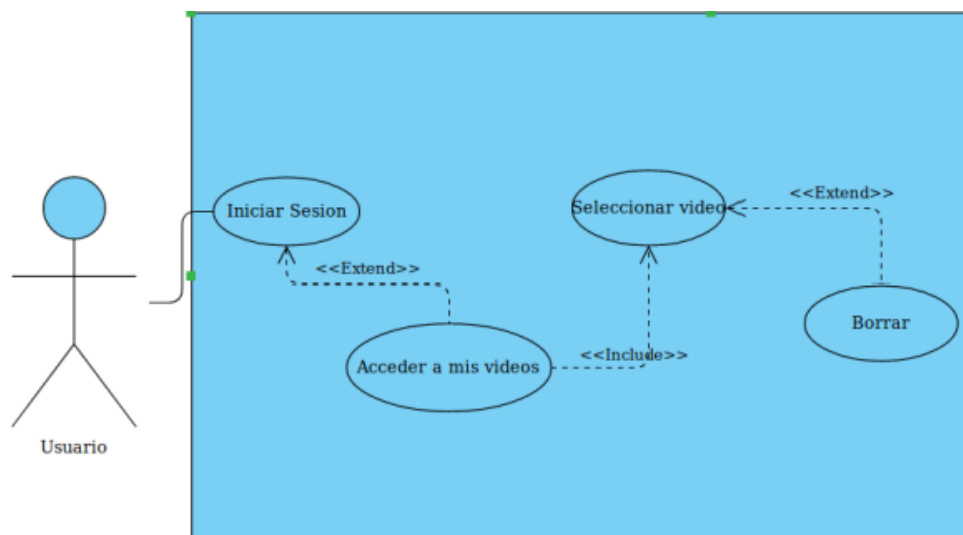
CU05- Editar video

Descripción:	Editar un vídeo.
Actores:	Usuarios.
Precondiciones:	<ul style="list-style-type: none">• Estar logeado.• Tener un vídeo.
Postcondiciones:	<ul style="list-style-type: none">• Información del video editada.
Flujo Normal:	<ul style="list-style-type: none">• Seleccionar editar video.
Flujo Alternativo:	<ul style="list-style-type: none">• Error del servidor.



CU06- Borrar video

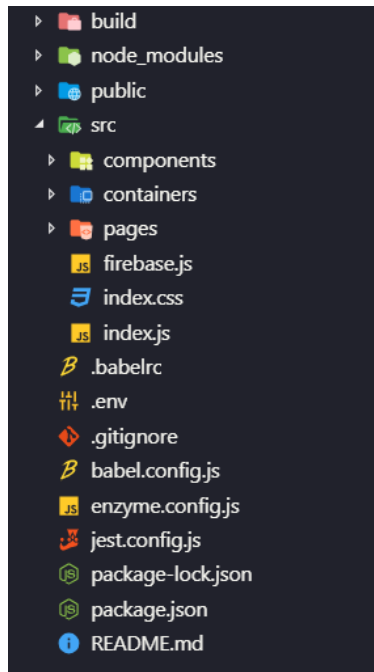
Descripción:	Borrar un vídeo.
Actores:	Usuarios.
Precondiciones:	<ul style="list-style-type: none">• Estar logeado.• Tener un vídeo.
Postcondiciones:	
Flujo Normal:	<ul style="list-style-type: none">• Seleccionar borrar video.
Flujo Alternativo:	<ul style="list-style-type: none">• Error del servidor.



3.2.2 Diseño

En este apartado del proyecto se va a mostrar una imagen de como está estructurado el proyecto.

La estructura del proyecto sería esta:



3.2.3 Implementación

Antes de pasar con la implementación en la que se mostrarán diversas capturas de pantalla del código, primero hay que aclarar unos conceptos básicos de como se ha organizado la estructura de este proyecto.

Containers:

Son los componentes que contienen lógica. También se les llama componentes listos o funcionales.

Componentes:

Estos componentes son los que no tienen ninguna lógica. También son llamados componentes tontos o no funcionales.

Pages:

Las páginas son las que llaman a los dos tipos de componentes mencionados anteriormente.

1- Containers.

1.1- App.

La App es el componente principal de la aplicación y por ello nos vamos a encontrar que dentro de este componente está el router.

```
render() {  
  return (  
    <BrowserRouter>  
      <div>  
        <Navbar />  
        <Switch>  
          <Route path="/" component={Home} exact />  
          <Route path="/mostListened" component={MostListened} />  
          <Route path="/watch/:id" component={Watch} />  
          <Route path="/login" component={LoginPage} />  
          <Route path="/register" component={Register} />  
          <Route path="/upload" component={UploadPage} />  
          <Route path="/myvideos" component={MyVideos} />  
        </Switch>  
      </div>  
    </BrowserRouter>  
  );  
}
```

El funcionamiento es bastante sencillo, solo tenemos que pasarle al componente Route que viene incorporado con el paquete `npm react-router-dom` y para invocar este componente simplemente hay que importarlo

```
import { BrowserRouter, Route, Switch } from "react-router-dom";
```

1.2- Navbar

El Navbar tenía claro que debía estar siempre en la parte superior de la página y para hacer solo había que importar el Navbar en el archivo js principal que es App.js y meterlo dentro del sistema de router.

```
return (  
  <BrowserRouter>  
    <div>  
      <Navbar />  
      <Switch>  
        <Route path="/" component={Home} exact />
```

Y ahora vamos a pasar a explicar el componente Navbar.

El Navbar tenía la importancia de que cuando un usuario se logea en la aplicación debería aparecer su nombre en el nav. Para ello lo que hice es descargar el paquete `npm universal-cookie` y gracias a esto podría recoger el nombre de usuario de quien se ha logeado en la aplicación para que se mostrase en el nav.

```
import Cookies from "universal-cookie";

class Navbar extends Component {
  render() {
    const cookies = new Cookies();

    return (
      <nav className="navbar-inverse">
        <div className="container-fluid">
          <div className="navbar-header">
            <a className="navbar-brand" href="/">
              Listen Me
            </a>
          </div>
          <ul className="nav navbar-nav">
            <li className="active">
              <NavLink to="/">Home</NavLink>
            </li>
            <li>
              <NavLink to="/mostListened">Most Listened</NavLink>
            </li>
          </ul>
          {!cookies.get("user") ? (
            <ul className="nav navbar-nav navbar-right">
              <li>
                <NavLink to="/login">Login</NavLink>
              </li>
              <li>
                <NavLink to="/register">Register</NavLink>
              </li>
            </ul>
          ) : (
            <ul className="nav navbar-nav navbar-right">
              <UserPanel user={cookies.get("user")} />
            </ul>
          )}
        </div>
      </nav>
    );
  }
}
```

Vamos a trocear esta imagen en partes más pequeñas para poder explicarlo mejor.

```
<ul className="nav navbar-nav">
  <li className="active">
    <NavLink to="/">Home</NavLink>
  </li>
  <li>
    <NavLink to="/mostListened">Most Listened</NavLink>
  </li>
</ul>
```

En esta primera imagen vemos el componente NavLink que también está incluido en el paquete `npm react-router-dom` y funciona como un href pero las direcciones hacen referencia al router principal.

```
{!cookies.get("user")} ? (  
  <ul className="nav navbar-nav navbar-right">  
    <li>  
      <NavLink to="/login">Login</NavLink>  
    </li>  
    <li>  
      <NavLink to="/register">Register</NavLink>  
    </li>  
  </ul>  
) : (  
  <ul className="nav navbar-nav navbar-right">  
    <UserPanel user={cookies.get("user")} />  
  </ul>  
)}
```

Este es el menú variable como me gusta decir a mi, simplemente comprueba a través de una ternaria que si no existe una cookie guardada que se llame user mostrará los botones Login y Registro y si existe la cookie saldrá el botón que llamará al componente de `UserPanel`.

1.3- Panel de usuario.

El panel de usuario lo saqué de una librería hecha por Microsoft que es Fabric.

Así que el primer paso para la instalación de esta librería es instalar el paquete npm:

`npm install office-ui-fabric-react`.

Y ya luego implementarlo en el componente.

```

export default class UserPanel extends Component {
  constructor(props) {
    super(props);

    this.state = {
      showPanel: false
    };
  }

  showPanel = () => this.setState({ showPanel: true });
  hidePanel = () => this.setState({ showPanel: false });

  logout = () => cookies.remove("user");
  render() {
    return (
      <div>
        <DefaultButton onClick={this.showPanel} text={this.props.user} />

        <Panel
          isOpen={this.state.showPanel}
          type={PanelType.smallFixedFar}
          onDismiss={this.hidePanel}
          headerText={this.props.user}
          closeButtonAriaLabel="Close"
          onRenderFooterContent={this._onRenderFooterContent}
        >
          <NavLink to="/">
            <PrimaryButton onClick={this.logout}>Logout</PrimaryButton>
          </NavLink>

          <PrimaryButton onClick={this.hidePanel}> Exit </PrimaryButton>

          <NavLink to="/upload">
            <PrimaryButton onClick={this.hidePanel}>upload</PrimaryButton>
          </NavLink>
          <NavLink to="/myvideos">
            <PrimaryButton onClick={this.hidePanel}>My videos</PrimaryButton>
          </NavLink>
        </Panel>
      </div>
    );
  }
}

```

Y como hemos hecho antes vamos a trocear la imagen para poder explicarlo mejor.

```

constructor(props) {
  super(props);

  this.state = {
    showPanel: false
  };
}

showPanel = () => this.setState({ showPanel: true });
hidePanel = () => this.setState({ showPanel: false });

logout = () => cookies.remove("user");

```

En esta imagen vemos tres funciones la primera cambia el estado de false a true, la segunda cambia el estado a false y la tercera borra la cookie del usuario para hacer así el desconectar.

```
render() {  
  return (  
    <div>  
      <DefaultButton onClick={this.showPanel} text={this.props.user} />  
  
      <Panel  
        isOpen={this.state.showPanel}  
        type={PanelType.smallFixedFar}  
        onDismiss={this.hidePanel}  
        headerText={this.props.user}  
        closeButtonAriaLabel="Close"  
        onRenderFooterContent={this._onRenderFooterContent}  
      >  
        <NavLink to="/">  
          <PrimaryButton onClick={this.logout}>Logout</PrimaryButton>  
        </NavLink>  
  
        <PrimaryButton onClick={this.hidePanel}> Exit </PrimaryButton>  
  
        <NavLink to="/upload">  
          <PrimaryButton onClick={this.hidePanel}>upload</PrimaryButton>  
        </NavLink>  
        <NavLink to="/myvideos">  
          <PrimaryButton onClick={this.hidePanel}>My videos</PrimaryButton>  
        </NavLink>  
      </Panel>  
    </div>  
  );  
}
```

Lo primero que encontramos aquí es el botón principal que abre el panel que en este caso le pasamos la función del show panel para que cuando haga click en el botón se abra el panel.

Para que aparezca el panel este tiene que saber cuando el estado esta a true entonces eso lo comprueba en el `isOpen` y ahí le pasamos el estado para que cuando sea true este se abra.

1.4- Delete Button.

Este botón sirve para eliminar un video de tu lista de videos.

```
export default class DeleteButton extends Component {
  constructor(props) {
    super(props);

    this.state = {
      recuperado: false
    };
  }

  componentWillMount() {
    video = [];
    db.collection("videos")
      .where(["user", "==", cookies.get("user")])
      .where("id", "==", this.props.id)
      .get()
      .then(querySnapshot => {
        querySnapshot.forEach(doc => {
          video.push(doc.data());
          this.setState({ recuperado: true });
        });
      });
  }

  deleteVideo = () => {
    db.collection("videos")
      .doc(video[0].id)
      .delete();

    window.location.href = "/myvideos";
  };

  render() {
    if (!this.state.recuperado) {
      return <Spinner />;
    } else {
      return <DefaultButton key={video[0].id} onClick={this.deleteVideo}>Delete</DefaultButton>;
    }
  }
}
```

Empecemos explicando lo que hace el `componentWillMount()` este lo que hace es ejecutar lo que hay dentro de esta función antes de que se monte el componente.

El funcionamiento de este componente es muy sencillo. El componente recibe por parámetros el id del video que se va a borrar y antes de que se monte el componente busca en la base de datos un vídeo que sea igual al id que me han pasado y que sea del usuario que está logeado en ese momento y entonces me guardo los datos de ese video en un array.

Ahora cuando el usuario le da al botón de borrar este llamará a la función `deleteVideo` que buscará en la colección de firebase un video que tenga ese id y lo borrará.

1.5- Login

```
class Login extends Component {
  constructor(props) {
    super(props);
    this.state = {
      email: "",
      password: ""
    };
  }

  handleChange(e) {
    this.setState({ [e.target.name]: e.target.value });
  }

  login(e) {
    e.preventDefault();
    db.collection("usuarios")
      .where("email", "==", this.state.email)
      .where("password", "==", this.state.password)
      .get()
      .then(function(querySnapshot) {
        querySnapshot.forEach(function(doc) {
          cookies.set("user", doc.data().name);
        });
        window.location.href = "/";
      });
  }
}
```

Como se puede ver en esta parte del componente hay una petición para comprobar que el email y el password son los correctos y si esto es correcto se creará una cookie con el nombre de usuario.

La otra función que es `handleChange` esta hecha para que cambie el estado.

1.6- Logout

```
import React, { Component } from "react";
import Cookies from "universal-cookie";
const cookies = new Cookies();

Logout = () => {
  cookies.remove("user");
};
```

Hice que este componente simplemente fuera una función y lo que hace es borrar la cookie que se llama user.

1.7- My videos

My videos es el apartado donde la gente puede ver los videos que han subido y tienen la opción de poder borrarlos.

```
class MyVideos extends Component {
  constructor() {
    super();
  }

  this.state = {
    recuperado: false
  };

  componentWillMount() {
    videos = [];
    db.collection("videos")
      .where("user", "==", cookies.get("user"))
      .get()
      .then(querySnapshot => {
        querySnapshot.forEach(doc => {
          videos.push(doc.data());
          this.setState({ recuperado: true });
        });
      });
  }
}
```

En esta imagen se ve lo mismo que en prácticamente todos los componentes que es el `componentWillMount` y dentro de la función lo que hace es almacenar en un array todos los videos del usuario que está logeado.

```
return (
  <div className="home">
    <div className="home-tittle">My videos</div>
    {videos.map(video => {
      return (
        <div className="my-videos-card" key={video.id}>
          <div>
            <NavLink
              to={"/watch/" + video.id}
              key={video.id}
              id={video.id}
              title={video.title}
              views={video.views}
              user={video.user}
            >
              <VideoCard
                div={"video-card_item"}
                class={"video"}
                key={video.id}
                id={video.id}
                title={video.title}
                views={video.views}
                user={video.user}
                width="320"
                height="240"
              />
            </NavLink>
          </div>
          <div>
            <DeleteButton id={video.id} />
          </div>
        </div>
      );
    })}
  </div>
);
```

En el return del componente simplemente recorro el array y voy pintando cada video.

1.8 – Upload

```
class Upload extends Component {
  constructor(props) {
    super(props);
    this.state = {
      title: "",
    };
  }

  handleChange(e) {
    this.setState({ [e.target.name]: e.target.value });
  }

  upload(e) {
    e.preventDefault();
    db.collection("videos")
      .doc(this.state.title + cookies.get("user"))
      .set({
        id: this.state.title + cookies.get("user"),
        title: this.state.title,
        user: cookies.get("user"),
        views: 0
      });
  }
}
```

En este componente vamos a hablar de la subida de un video a la página.

Lo que hace la función `upload` es guardar en la base de datos la información del nuevo video.

1.9- Watch

```
class Watch extends Component {
  constructor(props) {
    super(props);
    this.state = {
      recuperado: false
    };
  }

  componentDidMount() {
    db.collection("videos")
      .where("id", "=", this.props.match.params.id)
      .get()
      .then(querySnapshot => {
        querySnapshot.forEach(doc => {
          video = doc.data();
          this.setState({ recuperado: true });
        });
      });
  }

  updateView = () => {
    db.collection("videos")
      .doc(video.id)
      .update({
        id: video.id,
        title: video.title,
        user: video.user,
        views: video.views + 1
      });
  };
}
```

Este componente es la página de visualización de los vídeos y como vemos ahora la función que antes se llamaba `componentWillMount` ahora se llama `componentDidMount` esta se llama nada más se ha montado el componente y lo que hace es buscar en la base de datos el vídeo que tenga ese id.

Y en segundo lugar podemos ver en esta imagen otra función que se dedica a subir las visitas del vídeo cada vez que se accede a la página de visualización de ese vídeo.

El resto del componente es idéntico a los demás un return pintando el vídeo.

2- Componentes.

2.1- Videocard.

Al ser una página de vídeos musicales es necesario tener un reproductor y para ello usé la etiqueta html <video> para hacerlo.

```
let video = "../videos/" + this.props.title + ".mp4";
let user = this.props.user;
let views = this.props.views;
return (
  <div className={this.props.div}>
    <div className="video-card__header">{this.props.title}</div>
    <div className={this.props.class}>
      <video
        id={String(this.props.id)}
        className={this.props.class}
        width={this.props.width}
        height={this.props.height}
        controls
        controlsList="nodownload 0.2"
      >
        <source src={video} type="video/mp4" />
      </video>
    </div>
    <div className="video-card__footer">
      <div className="user">{user}</div>
      <div className="views">{views}</div>
    </div>
  </div>
);
```

2.2- Spinner

Este componente está sacado de react-bootstrap y se instala mediante

`npm install react-bootstrap bootstrap`.

Lo importamos a nuestro componente y ya simplemente llamamos al spinner de bootstrap.

```
import { Spinner as Spin } from "react-bootstrap";
import React, { Component } from "react";

class Spinner extends Component {
  render() {
    return (
      <Spin animation="border" role="status" className="spinner">
        <span className="sr-only">Loading...</span>
      </Spin>
    );
  }
}

export default Spinner;
```

3- Páginas

3.1- Home

En la home debido a la falta de tiempo simplemente se muestran todos los vídeos que hay en la aplicación pero en el futuro serán vídeos destacados o recomendados para la gente. También decir que la home lo único que tendría que hacer es llamar al container home pero en este caso la propia página es la que hace todo.

La home se divide en dos partes:

1- La llamada al back:

```
componentWillMount() {
  videos = [];
  db.collection("videos")
    .get()
    .then(querySnapshot => {
      querySnapshot.forEach(doc => {
        videos.push(doc.data());
        this.setState({ recuperado: true });
      });
    });
}
```

2- La home

```

render() {
  if (!this.state.recuperado) {
    return <Spinner />;
  } else {
    return (
      <div className="home">
        <div className="home-tittle">Recommended</div>
        <div className="video-card">
          {videos.map(video => {
            return (
              <NavLink
                to={"/watch/" + video.id}
                key={video.id}
                id={video.id}
                title={video.title}
                views={video.views}
                user={video.user}
              >
                <VideoCard
                  div={"video-card__item"}
                  class={"video"}
                  key={video.id}
                  id={video.id}
                  title={video.title}
                  views={video.views}
                  user={video.user}
                  width="320"
                  height="240"
                />
              </NavLink>
            );
          })}
        </div>
      </div>
    );
  }
}

```

Dado a que ya hemos llegado al apartado de explicar las páginas y esta es la única página en la que hay algo de código ya que las demás simplemente llaman al container (que es lo que deberían de hacer todas) vamos a explicar el funcionamiento de como sale o no el spinner y es más en esta página es en la que más se puede apreciar el spinner.

Como se puede observar al inicio del render, hay una comprobación exactamente esta:

```

`if(!this.state.recuperado) {
  return <Spinner>
}`

```

Esta comprobación lo que hace es que si recuperado es false se mostrará el spinner y es en la función `componentDidMount()` donde cuando le llegan los vídeos cambia el estado de recuperado a true.

3- Login

```
import React, { Component } from "react";
import Login from "../../containers/login/component";

export default class LoginPage extends Component {
  render() {
    return <Login />;
  }
}
```

En esta imagen se ve el ejemplo claro de lo que tiene que ser una página que como podemos ver simplemente llama a un componente en este caso al container de login.

Las demás páginas son iguales simplemente se importa el componente en cuestión y se llama en el render.

3.3 - Firebase

Para cerrar el punto de implementación solo falta poner el fichero de configuración de Firebase el cual nos permite poder conectarnos a nuestra base de datos.

```
import firebase from 'firebase';

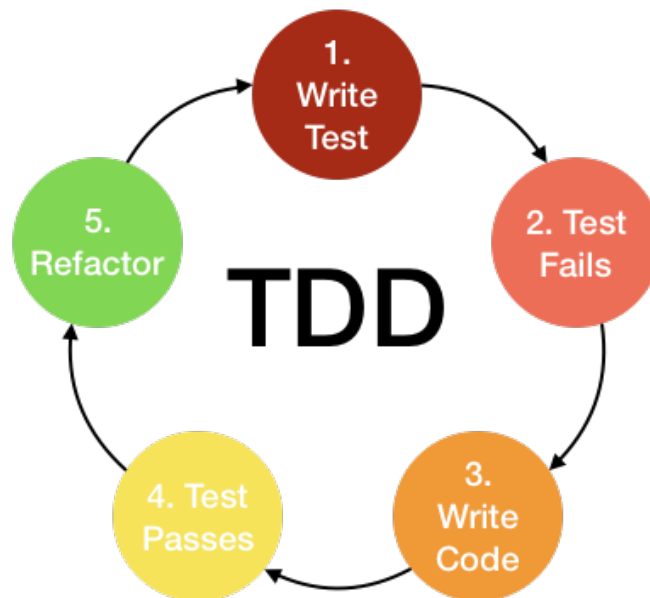
const config = {
  apiKey: "AIzaSyBnvkGTfE0-b-GQchSHL3XFdWS7EC34kTA",
  authDomain: "listen-me-c3203.firebaseio.com",
  databaseURL: "https://listen-me-c3203.firebaseio.com",
  projectId: "listen-me-c3203",
  storageBucket: "listen-me-c3203.appspot.com",
  messagingSenderId: "231265047183",
  appId: "1:231265047183:web:c29448ef84244dc2"
}

const fireb = firebase.initializeApp(config);
export default fireb;
```

Firebase ya nos proporciona el objeto config cuando nos creamos nuestra base de datos, lo único que tendríamos que hacer sería inicializar la aplicación con `firebase.initializeApp()`.

3.2.4 Pruebas

Aunque me hubiera gustado seguir la metodología TDD (Test-Driven-Development) como aprendí en el proyecto de la empresa el tiempo ha sido un impedimento así que los tests los hice después de aplicar el código en vez de seguir el flujo de trabajo del TDD.



La librería utilizada (como bien ya la he mencionado antes en el apartado de recursos) es Jest que me permite crear los tests de una forma bastante sencilla.

La aplicación web no está cubierta al cien por ciento ya que por tiempo no he podido lograr este objetivo y al final solo he cubierto las cosas más básicas de la aplicación.

Jest realmente no es una herramienta para hacer tests en si, es más bien un lanzador con el cual ejecutaremos los tests entonces para la creación de los tests nos vamos a apoyar en enzyme que es una utilidad para los tests.

Vamos a explicar poco a poco por imágenes el fichero de test de el Navbar.

Lo primero son los imports

```
import React from "react";
import { shallow } from "enzyme";
import Navbar from "../component.jsx";
import { ButtonToolbar, Button } from "react-bootstrap";
```

Quiero hacer especial incapie a el import del shallow y es porque también se podría haber importado el otro tipo que es mount.

La diferencia entre estos dos son que el mount crea el componente que vas a testear con todos los niveles que este posea por ejemplo:

```
<BrowserRouter>
  <div>
    <Navbar />
    <Switch>
      <Route path="/" component={Home} exact />
      <Route path="/mostListened" component={MostListened} />
      <Route path="/watch/:id" component={Watch} />
      <Route path="/login" component={LoginPage} />
      <Route path="/register" component={Register} />
      <Route path="/upload" component={UploadPage} />
      <Route path="/myvideos" component={MyVideos} />
    </Switch>
  </div>
</BrowserRouter>
;
```

En el archivo app.js donde se encuentra el router de la aplicación podemos ver que hay 3 niveles de componentes.

Uno seria ``<BrowserRouter>``, el siguiente ``<Switch>`` y el siguiente ``<Route>``.

Pues la diferencia de usar mount o shallow sería que mount lee los tres niveles mientras que shallow solo lee hasta el segundo nivel.

Una de las ventajas del shallow frente al mount es que al leer menos del componente ejecuta los tests mas rápido.

Después de esta pequeña explicación vamos a pasar al test.

Fichero Global del test:

```
describe("<Navbar />", () => {  
  let Component = null;  
  
  describe("shallow", () => {  
    beforeEach(() => {  
      Component = shallow("<Navbar />");  
    });  
  
    afterEach(() => {  
      Component = null;  
    });  
  
    it("should render without crashing", () => {  
      expect(Component.exists()).toBeTruthy();  
    });  
  
    it("should have ul", () => {  
      expect(Component.find("<ul />")).toBeTruthy();  
    });  
  
    it("should have ButtonToolbar", () => {  
      expect(Component.find(ButtonToolbar)).toBeTruthy();  
    });  
  
    it("should have Button", () => {  
      expect(Component.find(Button)).toBeTruthy();  
    });  
  
    it("should open the login modal", () => {  
      Component.setState({ modal: true });  
      expect(Component.find(Login)).toBeTruthy();  
    });  
  });  
});
```

Como podemos ver el test empieza con un `describe` esto es el mensaje que le damos al test para luego al ejecutar jest poder detectarlo bien si ha fallado alguno.

Dentro de este hay otro describe esto es a gusto personal ya que más adelante si quieres implementar más funcionalidad al componente y acabas teniendo una parte del componente con 3 o más niveles poder hacer otro describe con un mount sería lo ideal.

De todas formas si usas shallow con la función `Dive()` puedes entrar en los siguientes niveles pero si usas más de dos `Dive()` es mejor usar mount

Vamos a trocear la imagen para explicarlo mejor.

```
describe("<Navbar />", () => {  
  let Component = null;  
  describe("shallow", () => {  
    beforeEach(() => {  
      Component = shallow(<Navbar />);  
    });  
  
    afterEach(() => {  
      Component = null;  
    });  
  });  
});
```

Esta es la parte de la preparación para los tests para ello vamos a crear una variable Component, luego en el shallow ya empezamos a preparar las cosas.

`beforeEach()` Esto es lo que va a hacer antes de ejecutar los tests que en este caso monta el componente y lo almacena en Component.

`AfterEach()` Es lo que va a hacer después de que un test haya acabado que en este caso lo que hace es dejar la variable Component a null.

```
it("should render without crashing", () => {  
  expect(Component.exists()).toBeTruthy();  
});  
  
it("should have ul", () => {  
  expect(Component.find(<ul />)).toBeTruthy();  
});  
  
it("should have ButtonToolbar", () => {  
  expect(Component.find(ButtonToolbar)).toBeTruthy();  
});  
  
it("should have Button", () => {  
  expect(Component.find(Button)).toBeTruthy();  
});  
  
it("should open the login modal", () => {  
  Component.setState({ modal: true });  
  expect(Component.find(Login)).toBeTruthy();  
});
```

En esta imagen ya estamos viendo el funcionamiento de los tests quiero recalcar que debido a la falta de tiempo son tests muy sencillos.

Para empezar podemos ver que todos los tests tienen la palabra ``it()`` esto lo que hace es encapsular el test y tiene dos apartados.

1- Mensaje.

El primer apartado es el mensaje de ese test y es importante no confundirlo con el describe explicado anteriormente ya que el describe es el mensaje de un set de tests mientras que el it es un mensaje para ese test.

2- La función.

Dentro de la función del it podemos ver ``expect()`` que esto significa lo que espera recibir el test en el primer caso como podemos ver es que simplemente el componente exista.

Fuera del expect en este caso se ha puesto ``toBeTruthy()`` que lo que quiere decir es que si es verdadero que el componente existe pues dará positivo el test.

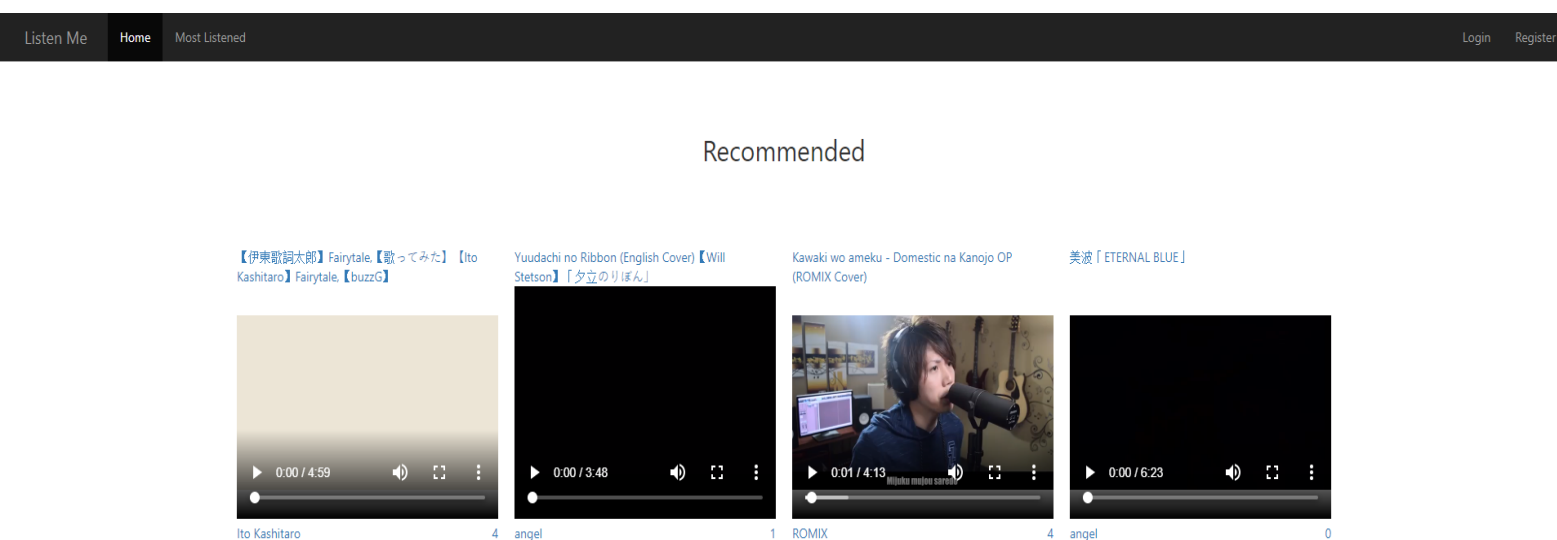
4. Manual De Usuario

Antes de dar comienzo al manual de uso se avisa que el diseño de la aplicación está en beta con lo cual lo que se va a ver a continuación en las imágenes no es el diseño final.

En este punto vamos a dar una vuelta por la aplicación enseñando que puedes hacer siendo un usuario anónimo y un usuario registrado.

Empecemos con el usuario anónimo

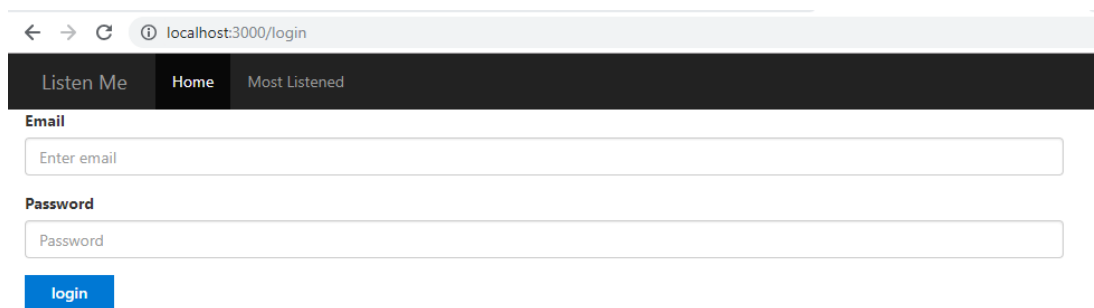
Esta sería la página principal al acceder a la aplicación web.



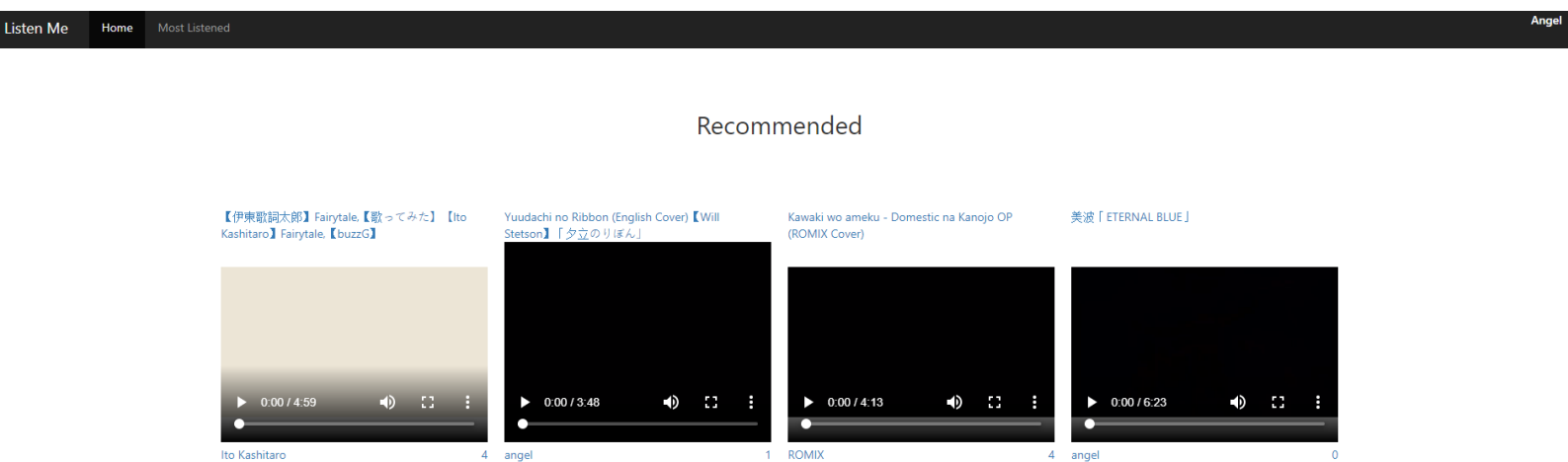
El usuario vería la sección de recomendados y los videos que se muestran en ella.

La siguiente ventana que podría ver el usuario es la de Login o Register.

The screenshot shows the 'Listen Me' application's register page. At the top is a dark navigation bar with 'Listen Me', 'Home', and 'Most Listened' links. Below the navigation bar is a registration form with three input fields: 'Email' (with placeholder text 'Enter email'), 'Password' (with placeholder text 'Password'), and 'name' (with placeholder text 'name'). Below the input fields is a green 'Signup' button. The browser's address bar shows 'localhost:3000/register'.

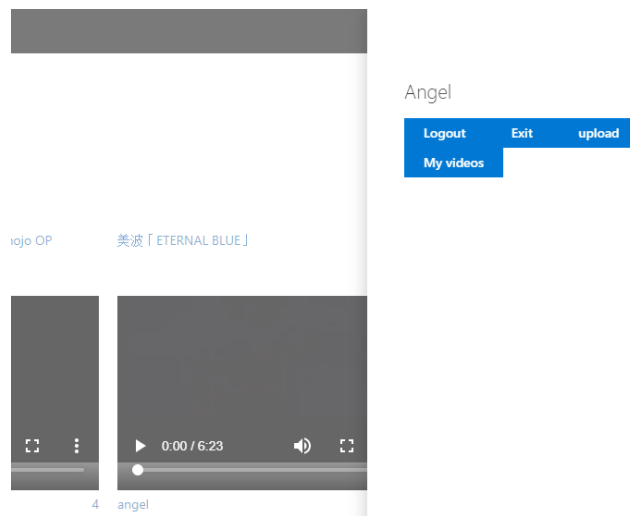


Ahora vamos a ver lo que puede hacer un usuario registrado en la aplicación.



Podemos observar que en la página principal de un usuario registrado y logeado aparece en el Navbar su nombre de usuario.

Si hacemos click en el nombre de usuario se abrirá el panel de usuario en el que aparecerán las diferentes funcionalidades que tenemos como usuarios de la aplicación.

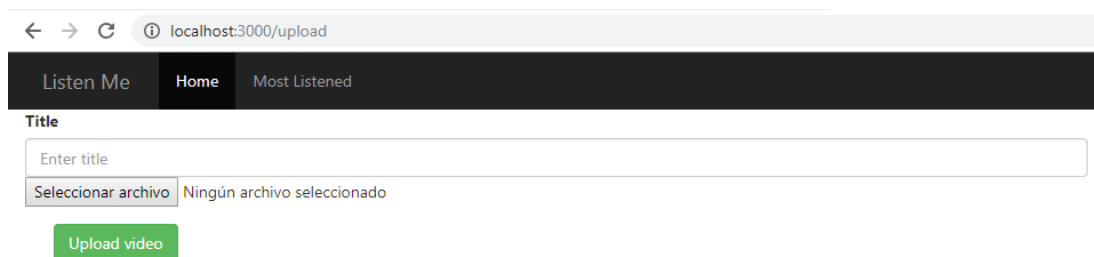


En el panel de usuario aparece nuestro nombre de usuario y cuatro botones con diferentes funcionalidades:

Logout – Cierra la sesión del usuario y te redirige a la página principal.

Exit – Cierra el panel de usuario.

Upload -

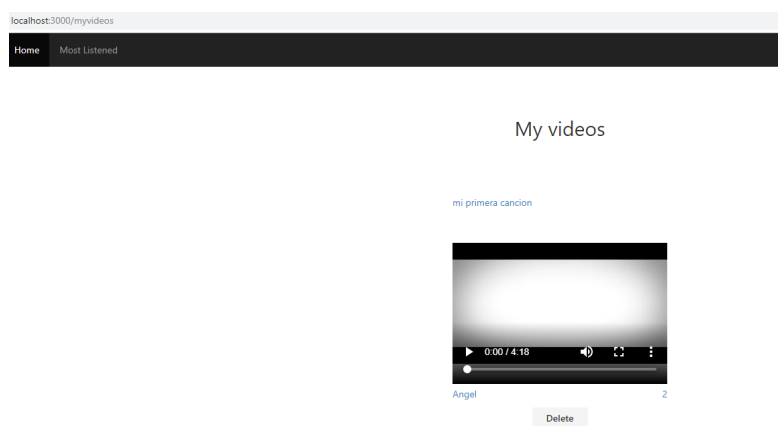


Cuando accedemos a la página de upload podremos subir un video a la aplicación y solo necesitamos escribir el título de nuestra canción. Por ejemplo:

Aquí dejo un ejemplo en forma de video para que puedan ver como funciona.

<https://www.youtube.com/watch?v=FA15u-TdKEA&feature=youtu.be>

Borrar videos- Para borrar videos que hayamos subido y ya no queremos que estén ahí vamos a nuestro panel de control – My videos.



Y hacemos click a Delete.

5. Valoración Personal

Después de realizar este proyecto puedo decir que estoy contento de haber elegido la librería de React ya que es una librería muy querida por la gente y es casi seguro que en 2019 tenga un crecimiento exponencial y muchas empresas quieran empezar a hacer proyectos con esta librería con lo cual siempre está bien aprender a usarla para tener unos conocimientos básicos sobre ésta.

Uno de los problemas que he tenido a la hora de hacer el proyecto ha sido el tiempo por las prácticas y no he podido poner en el proyecto todo lo aprendido sobre esta librería y algunas librerías más como Redux o Jest.

Una de las motivaciones con este proyecto fue el tema que escogí ya que me gusta mucho pero por otra parte estoy descontento ya que no he podido hacer todas las funcionalidades que quería por falta de tiempo.

Hablando sobre las tecnologías empleadas estoy bastante contento con el nivel de React que he obtenido ya que a parte de mirar por mi cuenta también se me asignó en las prácticas un proyecto de React así que aprendí bastante. Mientras que con Go no puedo decir lo mismo debido a que el proyecto de la empresa ya estaba bastante avanzado y tuve que centrarme en React y no pude darle el tiempo para aprender mejor Go con lo cual tuve que quitarlo del proyecto.

Y ya para acabar de valorar las tecnologías usadas en este proyecto voy a mencionar dos librerías que no he podido usar todo lo que quería que son Jest y Redux.

Podría haber usado mucho más Redux en la aplicación por ejemplo para llevar el tema de todas las llamadas a Firebase o para almacenar cosas en la store, mientras que con Jest podría haber testado más la aplicación pero al final solo he testado pequeñas cosas para que sirvan de ejemplo en este proyecto.

Para finalizar con mi opinión personal todo empezó con una idea bastante grande y bastante ambiciosa pensando que me daría tiempo a hacer todo pero al final cuando me di cuenta de que no iba a dar tiempo tuve que reducir bastante la aplicación pero al final no me ha disgustado el resultado.

Algunas de las mejoras para futuras ediciones sobre proyectos finales es que se podría dar información sobre el proyecto final después de los exámenes de la segunda evaluación ya que por ejemplo en mi caso empecé el proyecto la semana antes de las prácticas y empecé prácticamente el proyecto a ciegas ya que no tenía ninguna guía.

6. Agradecimientos

En este pequeño apartado de mi memoria quería agradecer primero antes que nada a mis compañeros Luis y Alex por el apoyo recibido durante el duro recorrido que ha sido hacer este proyecto debido a la falta de tiempo.

Otra de las grandes ayudas que he tenido ha sido Paloma que me ha ayudado a la hora de hacer el logo de la página y el apoyo recibido por ella.

Y a los últimos que quiero poner en este pequeño punto de mi memoria es a mis compañeros de proyecto en Sopra Pablo, Javi y Darío los cuales me han ayudado mucho a la hora de aprender sobre React y mejorar muchos aspectos sobre la programación.

7. Fuentes Bibliográficas

- **Redux**
 - <https://es.redux.js.org>
- **React**
 - <https://es.wikipedia.org/wiki/React>
- **Firebase Doc**
 - <https://firebase.google.com/docs/firestore?hl=es-419>
- **Introducción a Go**
 - <https://www.youtube.com/watch?v=juVYPx0UG80>
- **Curso Go Jesús Conde**
 - https://www.youtube.com/watch?v=d8_X7jQi078&list=PLEtcGQaT56cgPIRY15DVSUKGMw3tlzjLj
- **Curso Go G Coding Academy**
 - <https://www.youtube.com/watch?v=lodF5BaZdDY>
- **Jest**
 - <https://jestjs.io/docs/es-ES/tutorial-react>
- **Microsoft Fabric Get Started**
 - <https://developer.microsoft.com/en-us/fabric#/get-started>
- **Enzyme**
 - <https://github.com/airbnb/enzyme>
- **Firebase**
 - <https://firebase.google.com/?hl=es-419>
- **Axios**
 - <https://github.com/axios/axios>
- **React Universal Cookie**
 - <https://www.npmjs.com/package/universal-cookie>

- **TDD Train Easy Level**
 - <http://codingdojo.org/kata/FizzBuzz/>
- **React Dom Router**
 - <https://reacttraining.com/react-router/web/guides/quick-start>
- **React Bootstrap**
 - <https://react-bootstrap.github.io/>
- **React Materialize**
 - <http://react-materialize.github.io/react-materialize/?path=/story/react-materialize--welcome>
- **React Curso Primeros Videos**
 - https://www.youtube.com/watch?v=iHqa6ojKnHI&list=PLL0TiOXBeDai6x37_wQwWX6_qNZUM4FBm