

MovieMedia



Índice

0.- Capítulo 0: URLs proyecto

 0.1.- GitHub

 0.2.- Despliegue

1.- Capítulo 1: Introducción

 1.1.-Introducción del proyecto

 1.2.- Propósito

 1.3.- Objetivos del proyecto

 1.4.- Coste del proyecto

 1.4.1.- Costes de desarrollo

 1.4.2.- Costes de implementación

2.- Capítulo 2: Análisis del sistema

 2.1.- Introducción

 2.2.- Análisis de requisitos

 2.2.1.- Requerimientos hardware

 2.2.2.- Requerimientos software

 2.2.3.- Requisitos de red

3.- Capítulo 3: Estudio de la información relevante

 3.1.- Introducción

 3.2.- Valoración y comparación con antecedentes

4.- Capítulo 4: Técnicas utilizadas

 4.1.- Introducción

 4.2.- Login

 4.3.- Registro

 4.4.- Toolbar

4.5.- Chat

4.6.- Grid películas

4.7.- Detalles película

4.8.- Trailer

4.9.- Comentarios

4.10.- Tendencias y estrenos

5.- Capítulo 5: Prueba de ejecución

5.1.- Introducción

5.2.- Login

5.3.- Registro

5.4.- Pantalla principal

5.5.- Pantalla estrenos, week y daily

5.6.- Toolbar con info del usuario

5.7. Detalles de la película

5.8.- Trailer

5.9.- Comentarios

5.10.- Chat en directo

6.- Capítulo 6: Conclusiones

6.1.- Conclusiones

6.2.- Propuestas futuras

7.- Capítulo 7: Bibliografía y referencias bibliográficas

8. Anexo 1: Manual de instalación

CAPÍTULO 0: URLs PROYECTO

0.1.- GITHUB

Front-end:

https://github.com/GutierrezRomeroSantiago/Proyecto_FrontVF_peliculas-REACT

Back-end:

<https://github.com/GutierrezRomeroSantiago/BackEndV3react->

0.2.- DESPLIEGUE

Front-end:

<https://proyecto-front-vf-peliculas-react.vercel.app/>

Back-end:

<https://apirestmoviex.herokuapp.com/>

CAPÍTULO 1: INTRODUCCIÓN

INTRODUCCIÓN DEL PROYECTO

En nuestro proyecto hemos elaborado una aplicación web la cual está enfocada al ámbito de las redes sociales y las películas. En la aplicación diseñada los usuarios podrán interactuar entre ellos, valorar películas, hacer comentarios en películas, chatear en directo con otros usuarios, buscar cualquier película o descubrir los últimos estrenos y tendencias del momento.

PROPOSITO

El propósito de nuestro proyecto en un punto inicial consistía en la creación de una red social sobre un tema concreto, películas y series, planteamos también el hecho de que los usuarios pudiesen llevar a cabo diferentes actividades de carácter social en torno a películas y series.

OBJETIVOS DEL PROYECTO

- 1.- Analizar y estudiar la tecnología con la que construiremos el proyecto (React).
- 2.- Análisis de los recursos que utilizaremos, así como bases de datos, APIs, paquetes necesarios...
- 3.- Desarrollo del esqueleto del proyecto en el que se mostraban películas traídas desde la API pública TMDB
- 4.-Añadimos otras funcionalidades como login, chat en directo, buscador de películas, comentarios en directo, rating de películas
- 5.- Redacción de la memoria del proyecto

COSTE DEL PROYECTO

En nuestro caso no ha habido un coste económico del producto final ya que hemos utilizado “Free trials” que nos han permitido desarrollar una versión gratuita de la aplicación, aún así adquirir los servicios de pago de las bases de datos que utilizamos en este proyecto permitiría tener una mayor velocidad a la hora de realizar consultas y no habría un límite de usuarios simultáneos conectados a la aplicación.

1.4.1.-COSTES DE DESARROLLO

En la versión que hemos realizado los costes de hardware, software y personal han sido de 0€ aunque si se quisiese dar un upgrade a la aplicación se necesitaría un presupuesto de unos de 100€ a 500€

1.4.2 COSTES DE IMPLANTACIÓN

Los costes de implantación del producto pasarían por el pago y mantenimiento de un dominio que nos permitiera desplegar la aplicación en internet, además también habría que pagar a una serie de desarrolladores para que mantuvieran la aplicación y dieran soporte a la misma.

CAPÍTULO 2: ANÁLISIS DEL SISTEMA

2.1.- INTRODUCCIÓN

En este segundo capítulo se va a especificar como se ha desarrollado el proyecto según los requisitos que se han planteado en apartados anteriores.

2.2.- ANÁLISIS DE REQUISITOS

Los requerimientos necesarios para llevar a cabo este proyecto son de carácter técnico, en primer lugar y el requisito más importante ha sido el controlar la librería REACT ya que esta es la base de nuestro proyecto, en segundo lugar controlar el lenguaje de programación de javascript, el cual ha sido la herramienta mediante la que hemos construido el proyecto.

Por otra parte también es totalmente necesario tener conocimientos sobre bases de datos, en nuestro caso hemos aplicado una de las utilizadas durante el curso(MongoDB) y otra que hemos aprendido por nuestra propia cuenta (Firebase).

En último lugar será necesario conocer como integrar todas estas tecnologías, es decir como coordinar tanto el back-end como el front-end que hemos diseñado.

2.2.1. REQUERIMIENTOS HARDWARE

El equipo hardware que se ha utilizado para la realización de este proyecto ha sido únicamente un ordenador portátil, siempre y cuando no se tengan en cuenta los equipos que componen los cluster de nuestro proyecto en las bases de datos de MongoDB y firebase.

2.2.2. REQUERIMIENTOS SOFTWARE

Para la realización de este proyecto será necesario un editor de código, tener instalado NODE.JS, crear un proyecto de react (NPX create react app) con los siguientes paquetes dentro del proyecto:

- Universal Cookies
- React Bootstrap
- MUI CORE
- REACT PLAYER
- Mongoose
- Express
- Firebase

- Cors
- Typescript
- React router DOM

Estas tecnologías forman parte de la aplicación front-end y back-end.

2.2.3.REQUISITOS DE RED

En este caso los requisitos red necesarios serán básicamente una conexión a internet que nos permita consultar la documentación sobre las diferentes tecnologías que conforman nuestro proyecto.

CAPÍTULO 3: ESTUDIO DE LA INFORMACIÓN RELEVANTE

3.1.- INTRODUCCIÓN.

En este apartado se intentará dar una visión, partiendo de lo que ya conocemos del propósito del proyecto y los elementos que en él se necesitan para poder llevarlo a cabo, de lo que teníamos referente a la funcionalidad de nuestro proyecto antes y durante el desarrollo del mismo.

3.2.- VALORACIÓN Y COMPARACIÓN CON ANTECEDENTES.

La valoración en el punto inicial está marcada por la experimentación, en un primer momento no conocíamos la librería de REACT por lo que tratábamos de llevar a cabo de forma experimental los primeros pasos del proyecto.

Transcurrido menos de un mes y con un poco más de soltura con la herramienta pudimos comenzar a realizar tareas algo más complejas como consultas a APIs, heredar componentes, manejar estados, usar hooks...

3.3.- VALORACIÓN Y COMPARACIÓN CON LOS ACTUALES

Actualmente y tras haber terminado por completo el proyecto hemos alcanzado un buen nivel en la librería de REACT, esto nos permitió hacer implementaciones en el proyecto las cuales no habían sido posibles durante las primeras etapas de desarrollo.

Además también ha habido una mejora considerable en la utilización del lenguaje de programación de javascript en el cual hemos obtenido un nivel considerablemente alto, esto en parte nos ha ayudado mucho para implementar nuevas mejoras ya que REACT se construye a través de javascript.

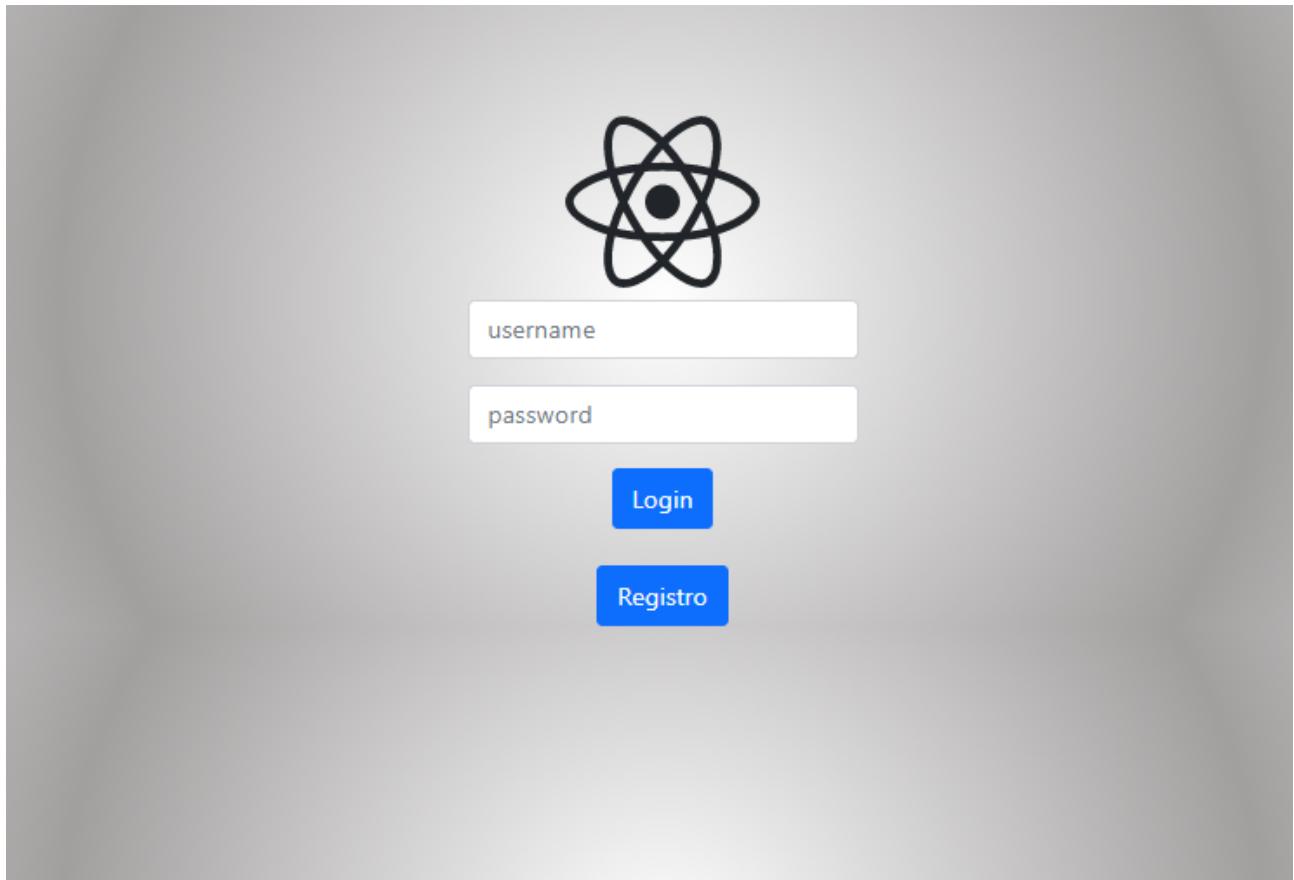
CAPÍTULO 4: TÉCNICAS UTILIZADAS

4.1.- INTRODUCCIÓN.

En el siguiente apartado veremos cómo se ha construido la aplicación. Explicaremos cada una de las partes con las que esta cuenta y cómo están funcionando.

4.2.- LOGIN.

Comenzaremos con el Login, cuenta con el siguiente aspecto y funciona mediante una base de datos alojada en mongoDB:



Como se puede apreciar nuestro login funciona con un formulario de dos campos en los cuales debemos introducir el usuario y contraseña:

```
<div className={styles.body}>
  <div className={styles.login}>
    <FaReact className={styles.image} />
    <br/>

    <Form>
      <Form.Group className="mb-3" >
        <Form.Control autoComplete="off" type="text" name="username" placeholder="username" onChange={this.handleChange} />
      </Form.Group>

      <Form.Group className="mb-3" >
        <Form.Control type="password" name="password" placeholder="password" onChange={this.handleChange} />
      </Form.Group>

      <Button onClick={this.handleClick}>
        | Login
      </Button>
      <br/>
      <Button href="/registro" onClick={this.registro}>
        | Registro
      </Button>
    </Form>
  </div>
</div>
```

Como se puede observar cada vez que se cambia un dato en el formulario (onChange) ejecutamos la función handleChange() la cual será la encargada de actualizar nuestros estados, a continuación podemos ver la función:

```
    async handleChange(e){
      console.log(e)
      if(e.target.name === "username"){
        await this.setState({
          username: e.target.value
        })
      }else{
        await this.setState({
          password: e.target.value
        })
      }
      console.log(this.state);
    }
```

Mediante un condicional cambiamos la ejecución de la función a un apartado del estado u otro.

Tras haber terminado de llenar los datos y pulsar submit se ejecutará la función handleClick() la cual realizará lo siguiente:

```

async handleClick(){
    let response = await axios.post(API_URL + 'credentials', this.state)
    console.log(response)
    if(response.data == ""){
        alert("Usuario o contraseña incorrectos")
    } else {
        this.props.parentCallback(response.data.username, response.data.nombre)
    }
}

```

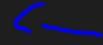
En este caso como podemos observar estamos haciendo una llamada a la restAPI desde la cual realizamos el control de validación de login si la respuesta está vacía significa que el usuario no ha introducido correctamente los datos, en caso contrario pasamos por props el usuario y contraseña al componente login. A continuación veremos la funcionalidad de esto:

```

const a = async function handleCallback(childData, childName) {
    const cookies = new Cookies();
    cookies.set('estado', 'true', { path: '/', expires: new Date(Date.now() + 2592000) });
    cookies.set('usuario', childData, { path: '/', expires: new Date(Date.now() + 2592000) });
    cookies.set('nombre', childName, { path: '/', expires: new Date(Date.now() + 2592000) });
    console.log(cookies.get('usuario'));
    window.location.href = "/";
}

function renderLogin() {
    return (
        <LoginPage parentCallback = {a}>
    )
}

```



Como se puede apreciar desde el archivo App.jsx recibimos la información y con ella creamos tres cookies una que indica que el logeo ha sido correcto, otra con el nombre de usuario y otra con el nombre completo del usuario.

Las siguientes cookies nos servirán para limitar la aplicación a usuarios que se hayan logeado, para ello lo haremos de la siguiente manera:

```

function render() {
    const cookies = new Cookies();
    if ((cookies.get('estado') === 'true')) {
        return renderApp();
    } else if ((cookies.get('registro') === 'true')) {
        return renderRegistro();
    } else {
        return renderLogin();
    }
}
return render()

```

Comprobamos si la cookie que hemos creado cuando el logeo era correcto está creada en el caso de que lo esté ejecutamos renderApp() que nos llevará a la página principal, en caso se nos devolverá a mostrar la pantalla de login.

La función utilizada en la restAPI ha sido la siguiente:

```
private logUser = async (req: Request, res: Response) => {
    await db.conectarBD()
    console.log(req.body.username)
    let data = await Credentials.findOne({
        username: req.body.username,
        password: req.body.password
    })

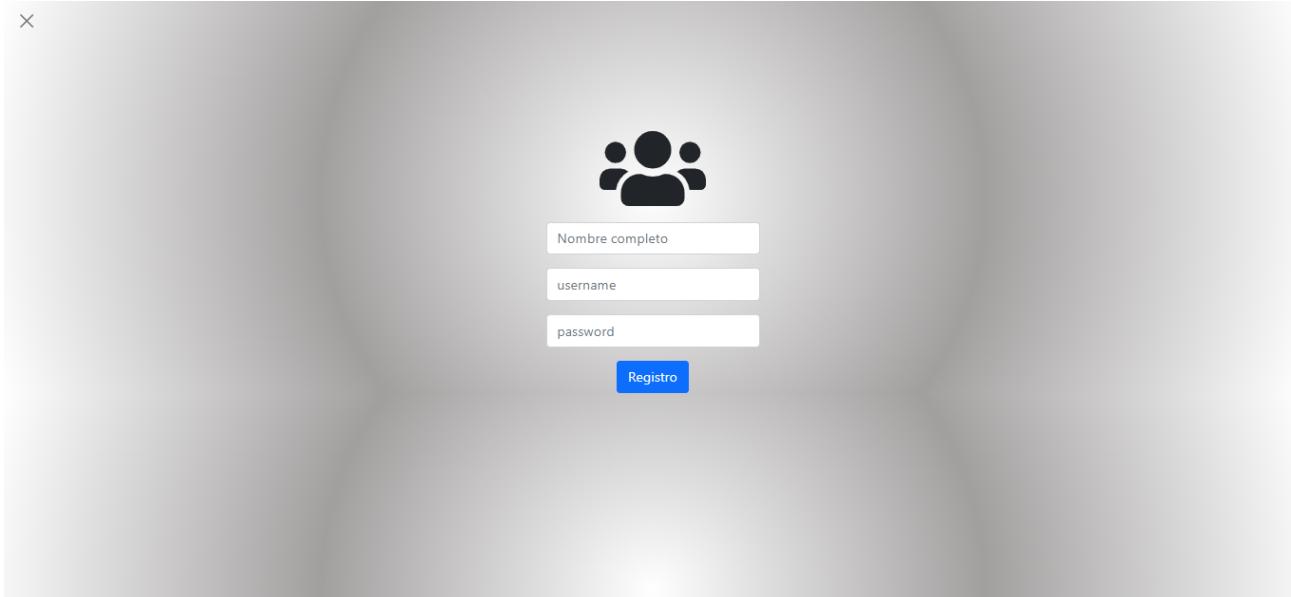
    res.send(data)
    // .then(async () => {
    //     const j = await Credentials.find({})
    //     console.log(j)
    // })
    // .catch((mensaje) => {
    //     res.send(mensaje)
    // })
    await db.desconectarBD()
}

//Posteamos la información referente al usuario. La validación del usuario
```

La cual realiza una consulta en la base de datos y devuelve la respuesta al front-end.

4.3.- REGISTRO.

La pantalla de registro sería la siguiente:



```
render() {
  return(
    <div>
      <CloseButton onClick={this.handleClick} className={styles.topRightButton} />
      <div className={styles.body}>
        <div className={styles.login}>

          <FaUsers className={styles.image} />
          <br/>
          <Form>
            <Form.Group className="mb-3" >
              <Form.Control autoComplete="off" type="text" name="nombre" placeholder="Nombre completo" onChange={this.handleChange} />
            </Form.Group>

            <Form.Group className="mb-3" >
              <Form.Control autoComplete="off" type="text" name="username" placeholder="username" onChange={this.handleChange} />
            </Form.Group>

            <Form.Group className="mb-3" >
              <Form.Control autoComplete="off" type="text" name="password" placeholder="password" onChange={this.handleChange} />
            </Form.Group>

            <Button onClick={this.handleClick}>
              Registro
            </Button>
            <br/>
          </Form>
        </div>
      </div>
    </div>
```

En esta tendremos que indicar nombre, nick, y password, lo hacemos de la siguiente manera:

Creamos nuestro función para actualizar los estados y que se vayan cambiando en tiempo real:

```

async handleChange(e){
    console.log(e)
    if(e.target.name === "username"){
        await this.setState({
            username: e.target.value
        })
    }
    else if(e.target.name === "password"){
        await this.setState({
            password: e.target.value
        })
    }
    else if(e.target.name === "nombre"){
        await this.setState({
            nombre: e.target.value
        })
    }
    console.log(this.state);
}

```

Mediante esta función recogemos la información de los formularios en los estados.

A continuación veremos el handleClick() en el cual se realizan comprobaciones para conocer si el usuario que se está indicando ya existe en la aplicación:

```

async handleClick(){
    //let response = await axios.post(API_URL + 'registrar', this.state)
    let value = {
        nombre: this.state.nombre,
        username: this.state.username,
        password: this.state.password
    }
    let bool = 1
    console.log(value)
    async function validate(){
        let response2 = await axios.get(API_URL + 'user')
        console.log(response2)
        for (let index = 0; index < response2.data.length; index++) {
            console.log(response2.data[index].username)
            if (value.username === response2.data[index].username){
                bool = 0
                break
            }
        }
        if (bool == 0){
            alert("El usuario ya existe")
        } else {
            alert("Usuario registrado")
            await axios.post(API_URL + 'registrar', value)
            const cookies = new Cookies();
            cookies.remove('registro');
            window.location.href="/"
        }
    }
    if (this.state.username == "" || this.state.password == "" || this.state.nombre == ""){
        alert("Se deben llenar todos los campos")
    } else {
        validate()
    }
}

async handleChange(e){}

```

Lo primero que hacemos es crear un doc con formato JSON en el que introducimos la información recogida en el estado.

A continuación creamos la función validate la cual realiza una consulta a la base de datos y comprueba si existe algún nombre de usuario igual.

Si el usuario existe se mostrará un alert, si no existe se creará el usuario posteando la información a la RESTAPI y después se publicará en la base de datos:

Consulta para comprobar usuarios:

```
private getUsers = async (req: Request, res: Response) => {
    await db.conectarBD()
    .then(async () => {
        const query = await Credentials.find({})
        res.json(query)
        console.log(query)
    })
    .catch((mensaje) => {
        res.send(mensaje)
    })
    await db.desconectarBD()
}
```

Posteando Usuarios:

```
}
//Posteamos la información referente al usuario. La validación del usuario se realiza en el frontend.
private postUser = async (req: Request, res: Response) => {
    const { username, password, nombre } = req.body
    await db.conectarBD()
    const dSchema = {
        _id_user: 10,
        username: username,
        password: password,
        nombre: nombre,
    }
    const oSchema = new Credentials(dSchema)
    console.log(oSchema)
    await oSchema.save()
        .then((doc: any) => res.send(doc))
        .catch((err: any) => res.send('Error: ' + err))
    await db.desconectarBD()
}
```

Colección de usuarios:

myFirstDatabase.credentials

STORAGE SIZE: 36KB TOTAL DOCUMENTS: 22 INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns

Aggregation

Search Indexes

INSERT DOCUMENT

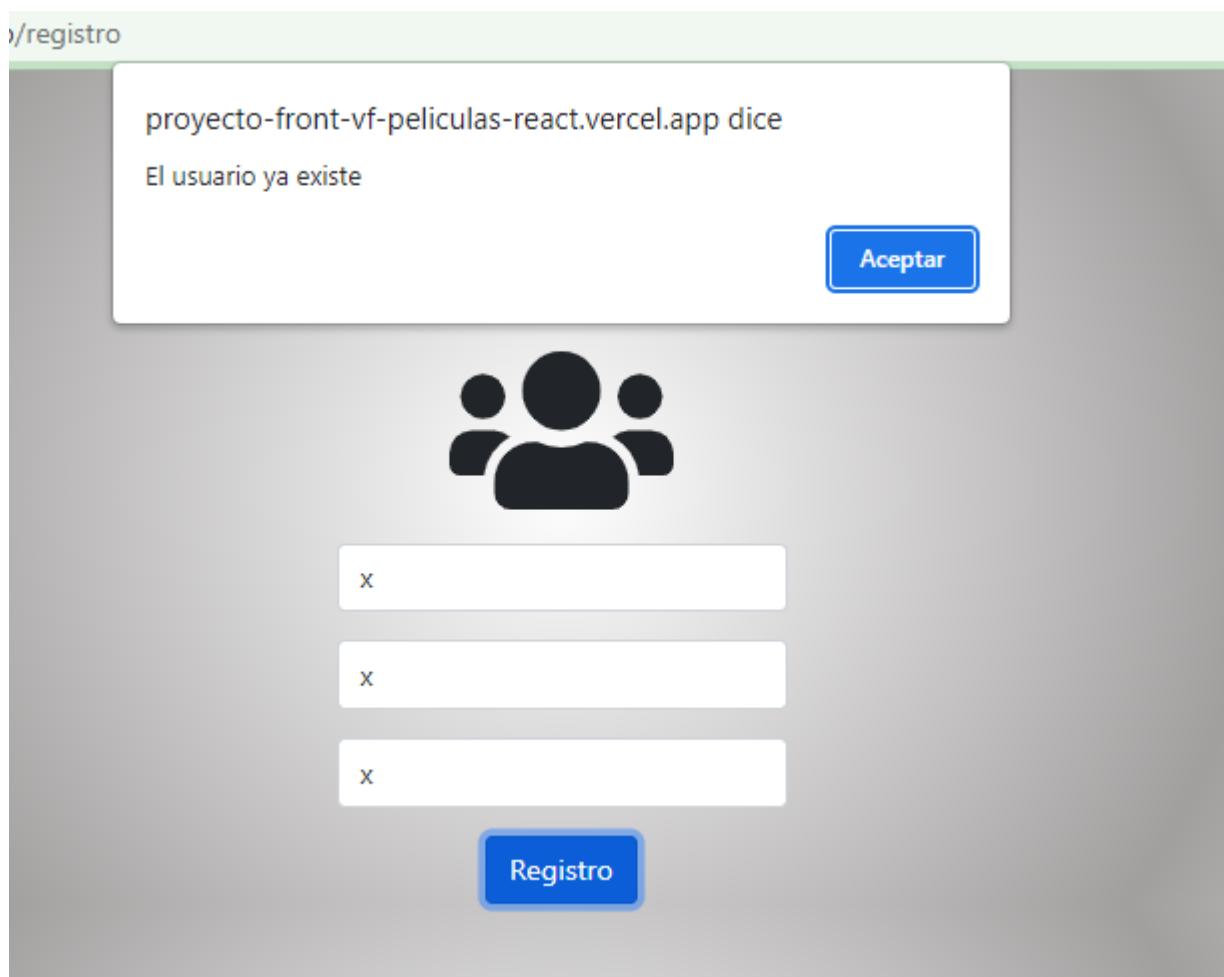
FILTER { field: 'value' }

OPTIONS Apply Reset

```
_id: ObjectId("627ce50f8700d4ff570860fe")
_id_user: 10
username: "Gustavo"
password: "123"
nombre: "Gustavo Adolfo Becquer"

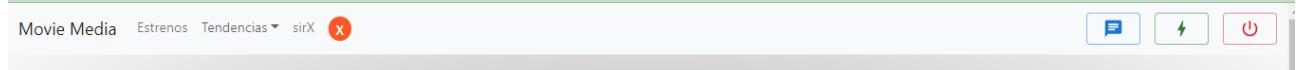
_id: ObjectId("627ce50f8700d4ff570860fe")
_id_user: 10
username: "CrisMar"
password: "123456789"
nombre: "Cristina Martinez Galvez"
```

Alerta mostrada:



4.4.- TOOLBAR

Nuestra toolbar ha sido construida mediante bootstrap y mui core el aspecto que tiene sería el siguiente:



En ella podemos ver el nombre de la aplicación, dos de las opciones de navegación para poder ver los diferentes tipos de películas, el nombre de usuario, un avatar autogenerado y tres botones correspondientes al chat, tecnologías utilizadas y log-out.

Su construcción se ha llevado a cabo de la siguiente manera:

```
return [
  <Navbar bg="light" expand="lg">
    <Container fluid>
      <Navbar.Brand href="/">Movie Media</Navbar.Brand>
      <Navbar.Toggle aria-controls="navbarScroll" />
      <Navbar.Collapse id="navbarScroll">
        <Nav
          className="me-auto my-2 my-lg-0"
          style={{ maxHeight: '100px' }}
          navbarScroll
        >
          <Nav.Link href="/estrenos">Estrenos</Nav.Link>
          <NavDropdown title="Tendencias" id="navbarScrollingDropdown">
            <NavDropdown.Item href="/diario">Diarias</NavDropdown.Item>
            <NavDropdown.Item href="/semanal">Semanales</NavDropdown.Item>
          </NavDropdown>
          <Nav.Link>{cookies.get('nombre')}</Nav.Link>
          <Nav.Link>
            <Stack direction="row" spacing={2}>
              <Avatar
                sx={{ bgcolor: deepOrange[500], width: 27, height: 27 }}
                alt={cookies.get('usuario')}
                src='../default-placeholder.png'
              />
            </Stack>
          </Nav.Link>
        </Nav>
        <FullScreenDialog/>
        <AlertDialogSlide />
        <Button
          sx={{ ml: 2 }}
          variant="outlined" color="error" onClick={a}>
          <PowerSettingsNewIcon/>
        </Button>
      </Navbar.Collapse>
    </Container>
  </Navbar>
]
```

Desde esta barra de herramientas podemos navegar además de desplegar determinadas ventanas como la del chat o la referente a tecnologías, para ello hemos utilizado los siguientes componentes de MUI core:

Este componente devuelve un botón que al ser presionado desplegará una pestaña en la que podremos ver el chat:

```
function FullScreenDialog() {
  const [open, setOpen] = React.useState(false);

  const handleClickOpen = () => {
    setOpen(true);
  };

  const handleClose = () => {
    setOpen(false);
  };

  return (
    <div>
      <Button sx={{ ml: 2 }} variant="outlined" onClick={handleClickOpen}>
        <ChatIcon/>
      </Button>
      <Dialog
        fullScreen
        open={open}
        onClose={handleClose}
        TransitionComponent={Transition}
      >
        <AppBar sx={{ position: 'relative' }}>
          <Toolbarx>
            <IconButton
              edge="start"
              color="inherit"
              onClick={handleClose}
              aria-label="close"
            >
              <CloseIcon />
            </IconButton>
            <Typography sx={{ ml: 2, flex: 1 }} variant="h6" component="div">
              MoviesChat
            </Typography>
          </Toolbarx>
        </AppBar>
        <ChatPage/>
      </Dialog>
    </div>
  );
}
```

En esta otra mostramos algunas de las tecnologías utilizadas:

```
export function AlertDialogSlide() {
  const [open, setOpen] = React.useState(false);

  const handleClickOpen = () => [
    setOpen(true);
  ];

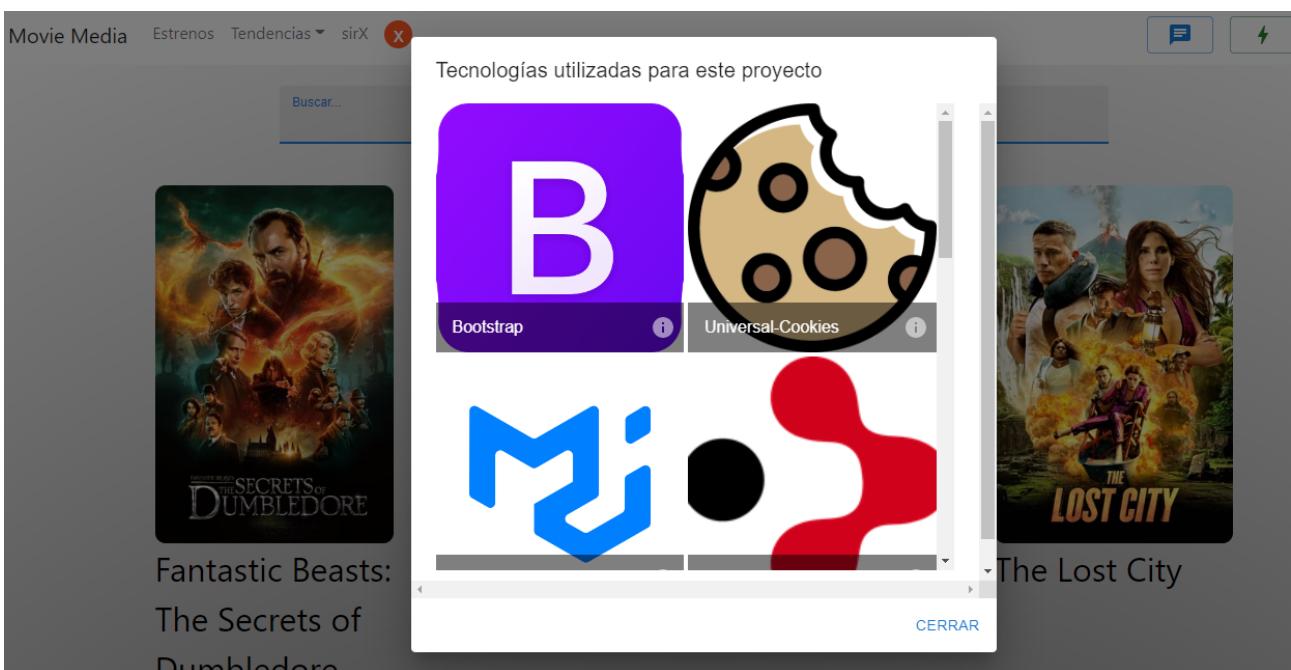
  const handleClose = () => {
    setOpen(false);
  };

  return (
    <div>
      <Button sx={{ ml: 2 }} color="success" variant="outlined" onClick={handleClickOpen}>
        <BoltIcon />
      </Button>
      <Dialog
        open={open}
        TransitionComponent={Transition}
        keepMounted
        onClose={handleClose}
        aria-describedby="alert-dialog-slide-description"
      >
        <DialogTitle>"Tecnologías utilizadas para este proyecto"</DialogTitle>
        <DialogContent>
          <Tecno/>
        </DialogContent>
        <DialogActions>
          <Button onClick={handleClose}>Cerrar</Button>
        </DialogActions>
      </Dialog>
    </div>
  );
}
```

Ventana desplegable del Chat:



Ventana de tecnologías:



4.5.- CHAT

Como hemos podido ver el chat está funcionando desde una ventana desplegable, a continuación veremos como lo hemos construido:

Credenciales para firebase:

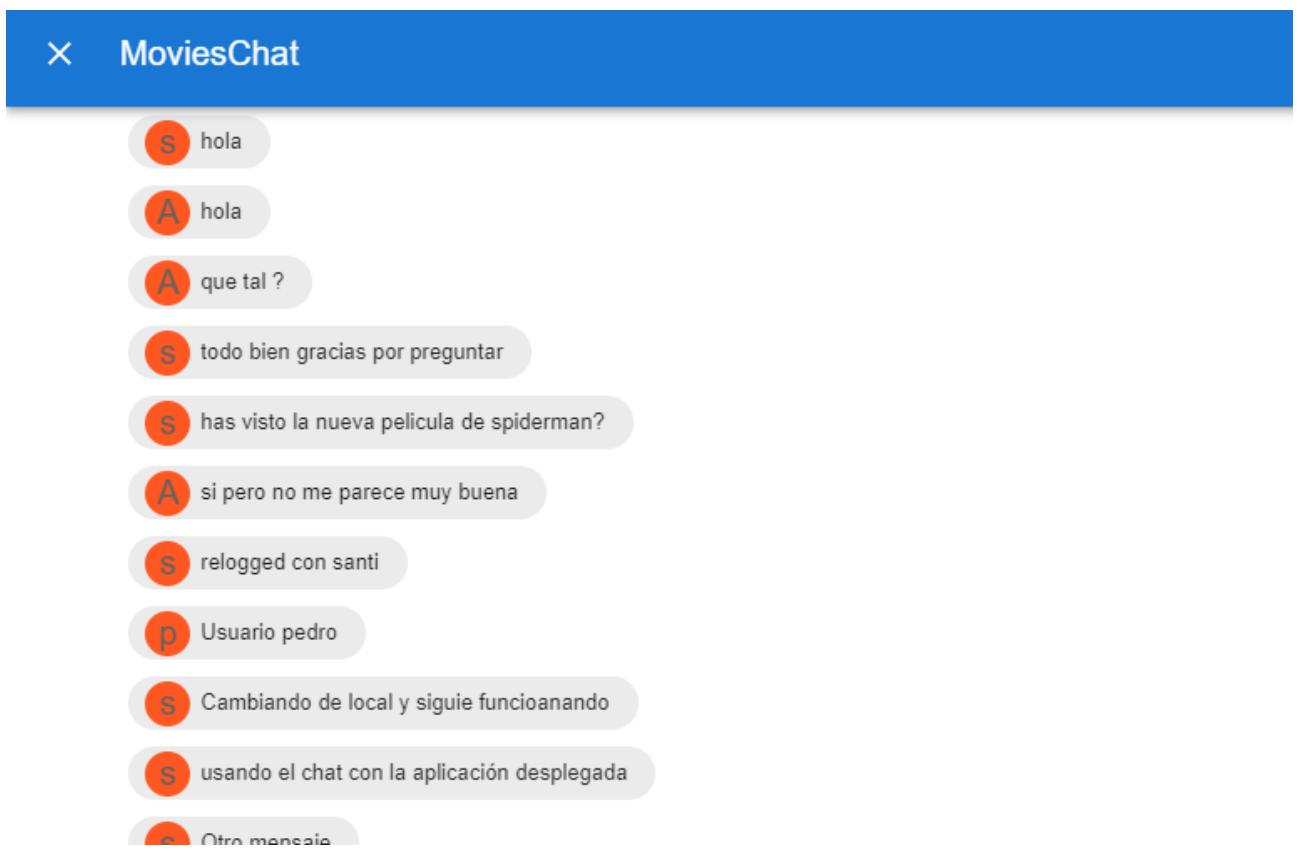
```
const firebaseConfig = {
  apiKey: "AIzaSyC520WwbYJJwLOCoSkBXEuWytzASwZ2qOU",
  authDomain: "fir-2-935b2.firebaseio.com",
  databaseURL: "https://fir-2-935b2-default-rtdb.europe-west1.firebaseio.app",
  projectId: "fir-2-935b2",
  storageBucket: "fir-2-935b2.appspot.com",
  messagingSenderId: "933312935174",
  appId: "1:933312935174:web:5aa100233f3423881ada87",
  measurementId: "G-JTTS1M0D7Q"
};
const app = initializeApp(firebaseConfig);
```

Realizamos una consulta de forma ciclica:

```
useEffect(() => {
  get(child(dbRef, `messages/`)).then((snapshot) => {
    if (snapshot.exists()) {
      setmessages(snapshot.val());
    } else {
      console.log("No data available");
    }
  }).catch((error) => {
    console.error(error);
  });
})
```

Mostramos haciendo un map al array de mensajes:

```
        {
          messages.map(message => {
            return (
              // <li key={message.id}>{message.text}</li>
              <div>
                <Chip sx={{padding: "5px", margin: "5px"}}
                  label={message.text}
                  icon={<Avatar sx={{ backgroundColor: deepOrange[500], width: 27, height: 27 }}>
                    alt={message.user}
                    src='../default-placeholder.png' />
                  <br/>
                </div>
              )
            )
          })
        </Box>
      <form onSubmit={b} sx={{position:"fixed"}}>
        {/* <input onChange={a} type="text" placeholder="Escribe un mensaje" value={message}/> */}
        <TextField sx={{width: "95%", padding: "5px", margin: "5px"}} autoComplete="off" variant="standard" onChange={a} type="text"/>
      </form>
    </div>
  )
}
```



4.6.- GRID PELÍCULAS

Para realizar el grid de películas hemos hecho lo siguiente:

Definimos la consulta en la API pública :

En esta vemos si hay info en el input buscamos la info, si no sacaremos una búsqueda genérica

```
useEffect(() => {
  setIsLoading(true)
  const searchUrl = search
  ? "/search/movie?query=" + search + "&page=" + page
  : "/discover/movie?page=" + page;
  get(searchUrl).then(data => {
    setMovies((prevMovies) => prevMovies.concat(data.results));
    setHasMore(data.page < data.total_pages);
    setIsLoading(false);
  });
}, [search, page])

if (!isLoading && movies.length === 0) {
  return <Empty />
}
```

Retornamos el componente InfiniteScroll con nuestro componente de movieCard en el que estará la foto y nombre de la película:

```
return (
  <InfiniteScroll
    dataLength={movies.length}
    hasMore={hasMore}
    next={() => setPage((prevPage) => prevPage + 1)}
    loader={<Spinner />}
  >
  <ul className={styles.movieGrid}>
    {movies.map((movie) =>
      <MovieCard key={movie.id} movie={movie}/>
    )}
  </ul>
  </InfiniteScroll>
)
```

MovieCard:

```

export function MovieCard({movie}) {
  const imageUrl = getMovieImg(movie.poster_path, 300)
  //const imageUrl = movie.poster_path ? "https://image.tmdb.org/t/p/w300" + movie.poster_path : placeHolder;
  return (
    <li className={styles.movieCard}>
      <Link className={styles.under} to={"/movies/" + movie.id}>
        <img width={230} height={345} className={styles.movieImage} src={imageUrl} alt={movie.title} />
        <div className={styles.title}>{movie.title}</div>
      </Link>
    </li>
  )
}

```

4.7.- DETALLES PELÍCULA:

En los detalles de cada película podemos ver información como la imagen de la película, nombre, género y descripción, además de otros apartados que veremos más adelante, lo hemos hecho de la siguiente manera:

Hacemos llamada a la API para traernos la info de la película en concreto:

```

useEffect(() => [
  setIsLoading(true)

  get("/movie/" + movieId).then(data => {
    setIsLoading(false)
    setMovie(data);
  })
;
  get("/movie/" + movieId + "/videos").then(data => {
    setkey(data.results[0].key);
  })
// movieKey().then((datos1) => setkey(datos1))
;
  prueba(movieId).then((datos) => setout(datos))
  // prueba().then((datos) => setout(datos))
  //setout(prueba());
  //getComment()
], [movieId])

```

Desplegamos la información:

```
const imageUrl = getMovieImg(movie.poster_path, 500);
return <div>
  <div className={styles.detailsContainer}>
    <img className={styles.col + " " + styles.movieImage} src={imageUrl} alt={movie.title} />

    <div className={styles.col + " " + styles.movieDetails}>
      <p className={styles.primero}>
        <strong> Título:</strong> {movie.title}
      </p>
      <p>
        <strong> Descripción:</strong> {movie.overview}
      </p>
      <p>
        <strong> Género:</strong> {movie.genres.map(genre => genre.name).join(', ')}
      </p>
      <p>
        <strong> Fecha de estreno:</strong> {movie.release_date}
      </p>
    </div>
  </div>

<DraggableDialog mkey={key}/>
```

4.8.- TRAILER:

En este apartado también podemos ver el trailer de la película que hemos elegido dentro de una ventana arrastrable.

Lo hemos hecho de la siguiente manera:

Importamos la ventana arrastrable que nos da MUI core:

```
import * as React from 'react';
import Button from '@mui/material/Button';
import Dialog from '@mui/material/Dialog';
import DialogActions from '@mui/material/DialogActions';
import DialogContent from '@mui/material/DialogContent';
import DialogTitle from '@mui/material/DialogTitle';
import Paper from '@mui/material/Paper';
import Draggable from 'react-draggable';
import ReactPlayer from 'react-player'

function PaperComponent(props) {
  return (
    <Draggable
      handle="#draggable-dialog-title"
      cancel={['[class*="MuiDialogContent-root"]']}
    >
      <Paper {...props} />
    </Draggable>
  );
}

export function DraggableDialog(prop) {
  console.log(prop)
  const [open, setOpen] = React.useState(false);
  //seturl('https://www.youtube.com/watch?v=' + prop)

  const handleClickOpen = () => {
    setOpen(true);
  };
}
```

Utilizaremos un paquete llamado REACT player para poder reproducir video en nuestro proyecto.

Introducimos el video dentro de la ventana:

```

};

return [
  <div>
    <Button style={center} variant="contained" onClick={handleClickOpen}>
      Ver trailer
    </Button>
    <Dialog
      open={open}
      onClose={ handleClose }
      PaperComponent={PaperComponent}
      aria-labelledby="draggable-dialog-title"
    >
      <DialogTitle style={{ cursor: 'move' }} id="draggable-dialog-title">
        Arrastra el video donde quieras!
      </DialogTitle>
      <DialogContent>
        <ReactPlayer
          url={'https://www.youtube.com/watch?v=' + prop.mkey}
          width='480px'
          playing= {true} />
      </DialogContent>
      <DialogActions>
        <Button onClick={ handleClose }>Cerrar</Button>
      </DialogActions>
    </Dialog>
  </div>
];

```

Le pasamos la prop al componente con la key del video:

Consulta de la key:

```

;
get("/movie/" + movieId + "/videos").then(data => {
  setkey(data.results[0].key);
})

```

Pasamos por props:

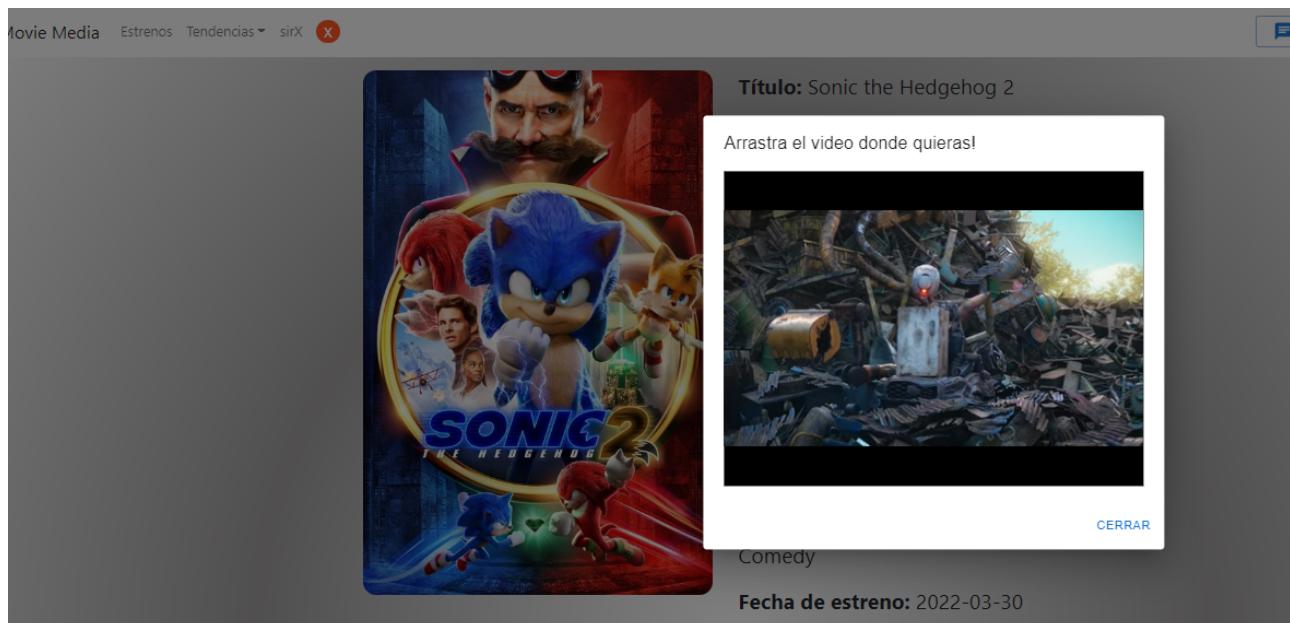
```

      </p>
    </div>
  </div>

  <DraggableDialog mkey={key}/>
  <div className={styles.commentsContainer}>
    <h3>Comentarios:</h3>
    <hr/>

```

El resultado sería el siguiente:



4.9.- COMENTARIOS:

Tienen el siguiente aspecto:

A screenshot of a comment section for a movie review. The section has a header "Comentarios:" and a text input field labeled "Comentario". Below the input field, there is a rating scale with five stars and a green "Publicar" button. Two comments are listed: one from user "h" and one from user "s". Each comment includes a profile picture, a star rating (5 stars), and the comment text.

Lo hemos hecho de la siguiente manera:

Llamada a la api en la que hacemos una búsqueda en mongoDB mediante el ID de la película:

```
export function prueba(prop) {
  return axios.get(API_URL + 'comenta/' + prop).then(res => {
    return res.data;
  });
}
```

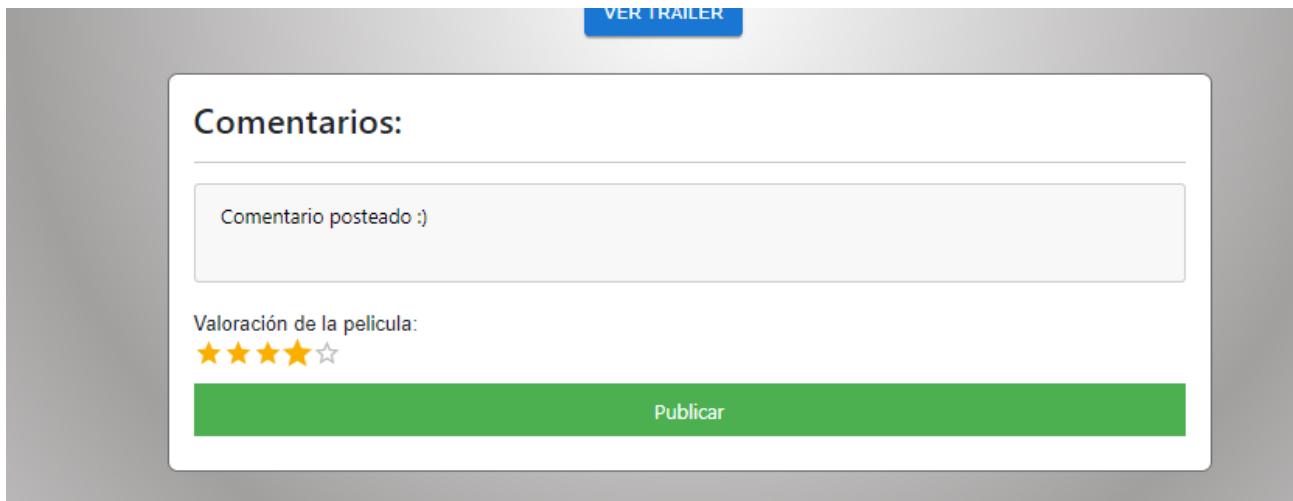
Llamamos al método dentro del estado y cargamos el estado:

```
// movieKey().then((datosI) => setKey(datosI))
;
prueba(movieId).then((datos) => setout(datos))
// prueba().then((datos) => setout(datos))
//setout(prueba());
//getComment()
```

Le hacemos el map a la información que llega:

```
function returnComments(data){
  if (data){
    return data.map((item) => {
      return [
        <div>
          <Box sx={{ width: '100%', height: 150, flexGrow: 1 }}>
            <Grid>
              <Item>
                <Stack direction="row" spacing={2}>
                  <Avatar sx={{ bgcolor: deepOrange[500], alt={item.usuario} src='../default-placeholder.png' aria-owns={open ? 'mouse-over-popover' : undefined} aria-haspopup="true" onMouseEnter={handlePopoverOpen} onMouseLeave={handlePopoverClose} }>
                    <Popover id="mouse-over-popover" sx={{ pointerEvents: 'none', }} open={open} anchorEl={anchorEl} anchorOrigin={{ vertical: 'bottom', horizontal: 'left', }} transformOrigin={{ vertical: 'top', horizontal: 'left', }} onClose={handlePopoverClose} disableRestoreFocus>
                      <Typography sx={{ p: 1 }}>{item.usuario}</Typography>
                    </Popover>
                  <br/>
                  <Rating name="read-only" value={item.value} readOnly />
                </Stack>
              </Item>
            </Grid>
          </Box>
        </div>
      ];
    });
  }
}
```

Para postear un comentario lo haremos de la siguiente manera:



Como podemos ver este cuenta con una valoración en estrellas la cual se guarda con el comentario, lo haremos de la siguiente manera:

Formulario con interacciones onChange y onClick:

```
<form>
  <textarea className={styles.textarea} type="text" placeholder="Comentario" maxLength="40" onChange={a} />

  <Box sx={{'& > legend': { mt: 2 }, }}>
    <Typography component="legend">Valoración de la pelicula:</Typography>
    <Rating
      name="simple-controlled"
      value={value}
      onChange={(event, newValue) => {
        setValue(newValue);
      }}
    />
  </Box>

  <button type="reset" className={styles.button} onClick={b}>Publicar</button>
</form>
```

Funciones onChange y onClick:

```

        }

        const b = async function handleClick(){
            const cookies = new Cookies();

            let values = {
                usuario: cookies.get('usuario'),
                movieId: movieId,
                comment: comment,
                value: value
            }
            await axios.post(API_URL + 'comentario', values)
            setTimeout(window.location.reload(), 10000);
            //window.location.reload();

        }

        const a = async function handleChange(e){
            console.log(e)
            setcomment(e.target.value)
            console.log(comment);
            console.log(movieId)
        }
        function returnComments(data){
```

Post en la restApi:

```

//Recibimos el id de la pelicula, el id del usuario y el comentario
private postComment = async (req: Request, res: Response) => {
    const { usuario, movieId, comment, value } = req.body
    console.log(req.body)
    await db.conectarBD()
    const dSchema = {
        usuario: usuario,
        movieId: movieId,
        comment: comment,
        value: value
    }
    const oSchema = new Comments(dSchema)
    console.log(oSchema)
    await oSchema.save()
        .then((doc: any) => res.send(doc))
        .catch((err: any) => res.send('Error: ' + err))
    await db.desconectarBD()
}

//Recibir los comentarios de la pelicula
```

Resultado en BD:

myFirstDatabase.comments

STORAGE SIZE: 36KB TOTAL DOCUMENTS: 27 INDEXES TOTAL SIZE: 36KB

Find

Indexes

Schema Anti-Patterns 0

Aggregation

Search Indexes •

FILTER { field: 'value' }

QUERY RESULTS: 1-20 OF MANY

```
_id:ObjectId("628613e06f6ba3b543c85c61")
usuario:"santi"
movieId:"752623"
comment:"No me gusto en absoluto"
value:2
```

4.10.- TENDENCIAS Y ESTRENOS:

Páginas de estrenos y tendencias, estas páginas serían similares al movieDetails genérico solo que en ellas se realiza una consulta diferente:

Tendencias diarias:

```
useEffect(() => {
  const searchUrl = "/trending/movie/day?page=" + page;
  get(searchUrl).then(data => {
    setDay((prevMovies) => prevMovies.concat(data.results));
    setHasMore(data.page < data.total_pages);
  });
}, [page])

return (
  <InfiniteScroll
    dataLength={day.length}
    hasMore={hasMore}
    next={() => setPage((prevPage) => prevPage + 1)}
    loader={<Spinner />}
  >
    <p className={styles.pageTitle}>Tendencias diarias</p>
    <ul className={styles.movieGrid}>
      {day.map((movie) =>
        <MovieCard key={movie.id} movie={movie}/>
      )}
    </ul>
  </InfiniteScroll>
)
```

Tendencias semanales:

```
useEffect(() => {
  const searchUrl = "/trending/movie/week?page=" + page;
  get(searchUrl).then(data => {
    setWeek((prevMovies) => prevMovies.concat(data.results));
    setHasMore(data.page < data.total_pages);
  });
}, [page])

return (
  <InfiniteScroll
    dataLength={week.length}
    hasMore={hasMore}
    next={() => setPage((prevPage) => prevPage + 1)}
    loader={<Spinner />}
  >
    <p className={styles.pageTitle}>Tendencias Semanales</p>
    <ul className={styles.movieGrid}>
      {week.map((movie) =>
        <MovieCard key={movie.id} movie={movie}/>
      )}
    </ul>
  </InfiniteScroll>
)
}
```

Estrenos:

```
useEffect(() => {
  const searchUrl = "/movie/upcoming?page=" + page;
  get(searchUrl).then(data => {
    setupComing((prevMovies) => prevMovies.concat(data.results));
    setHasMore(data.page < data.total_pages);
  });
}, [page])

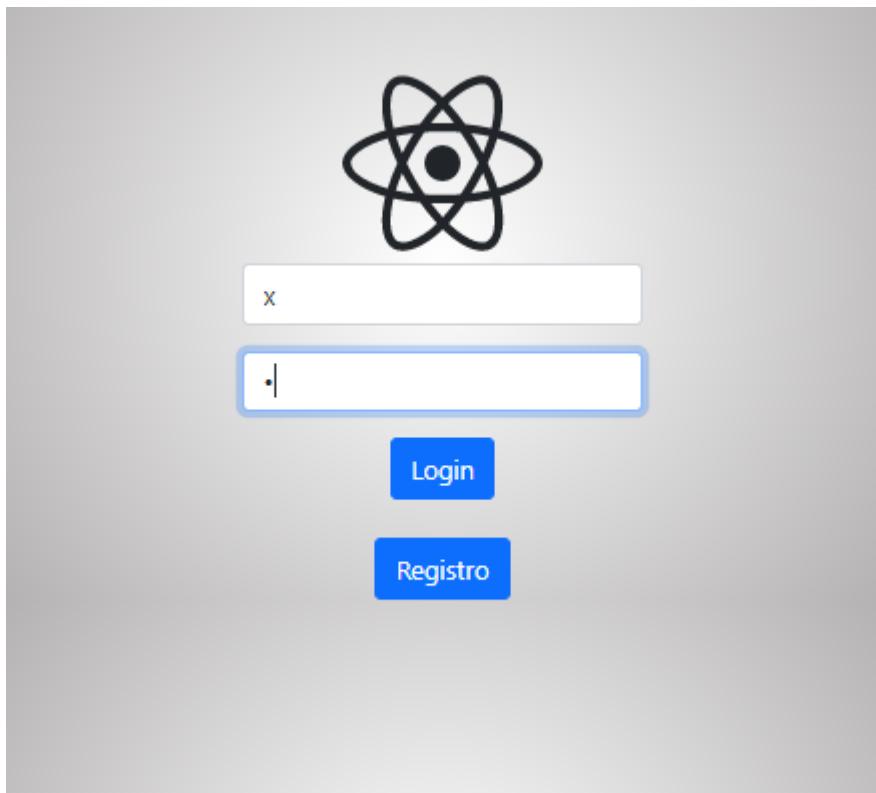
return (
  <InfiniteScroll
    dataLength={upComing.length}
    hasMore={hasMore}
    next={() => setPage((prevPage) => prevPage + 1)}
    loader={<Spinner />}
  >
    <p className={styles.pageTitle}>Últimos estrenos</p>
    <ul className={styles.movieGrid}>
      {upComing.map((movie) =>
        <MovieCard key={movie.id} movie={movie}/>
      )}
    </ul>
  </InfiniteScroll>
)
}
```

CAPÍTULO 5: PRUEBAS DE EJECUCIÓN

5.1.-INTRODUCCIÓN.

En el siguiente apartado se mostrarán capturas de pantalla sobre la ejecución de la aplicación:

5.2.- LOGIN.



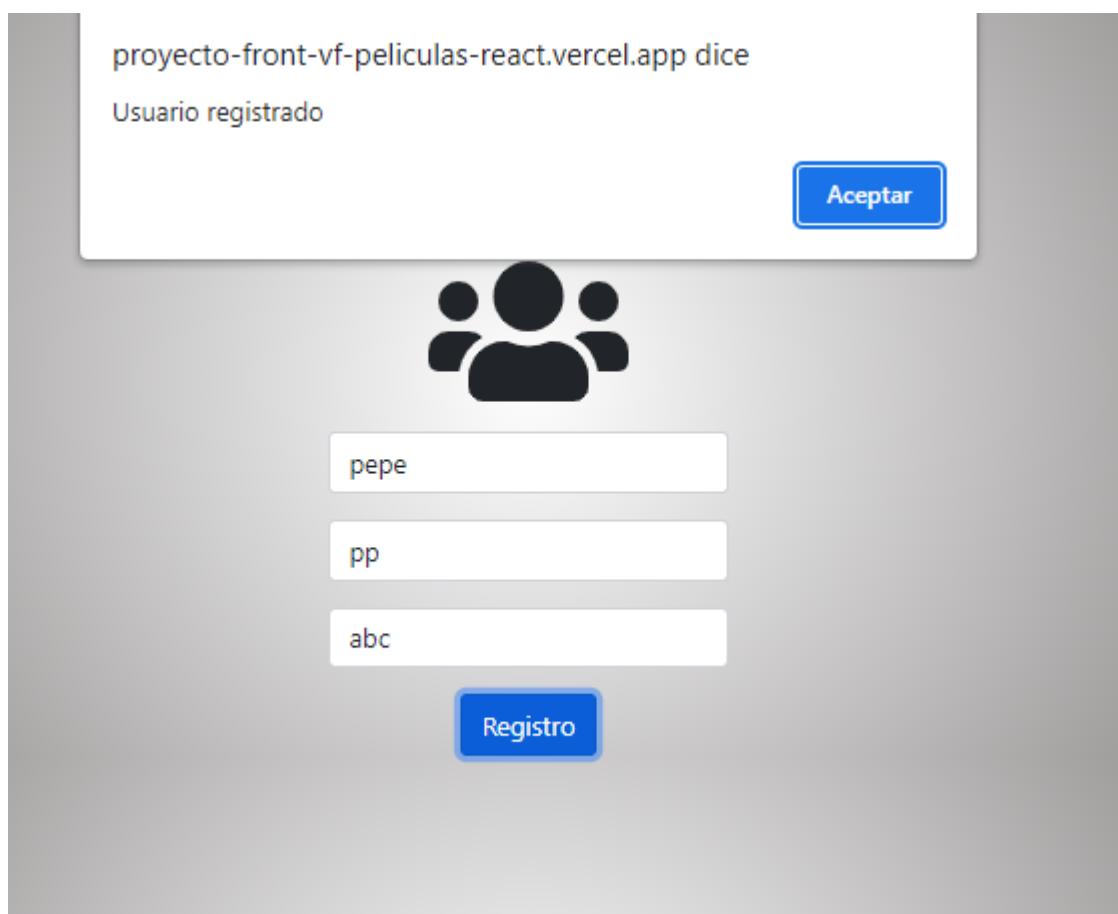
Cookies creadas:

Cookies en uso

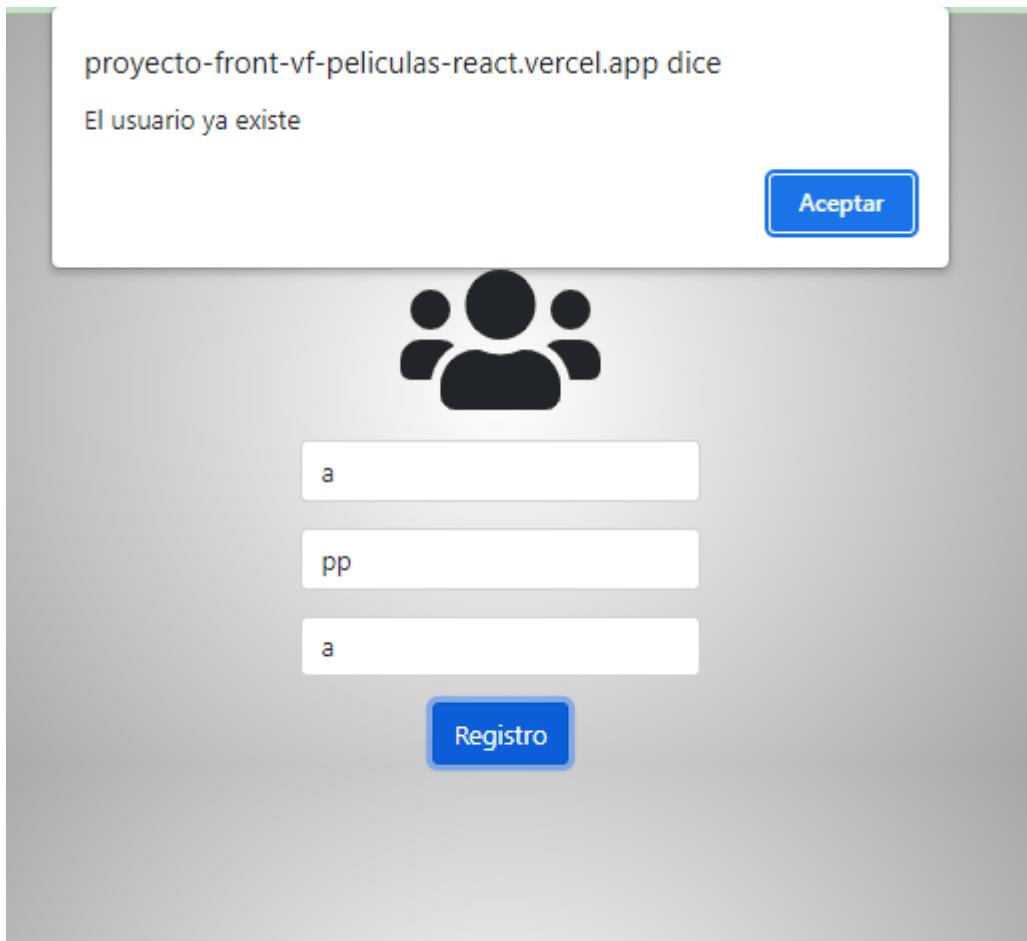
| Permitido | Bloqueado |
|--|--------------------------------------|
| Se han permitido las siguientes cookies al visitar esta página | |
| <ul style="list-style-type: none">• _ga• _ga_JTTS1M0D7Q• estado• nombre• usuario | |
| Nombre | no se ha seleccionado ninguna cookie |
| Contenido | no se ha seleccionado ninguna cookie |

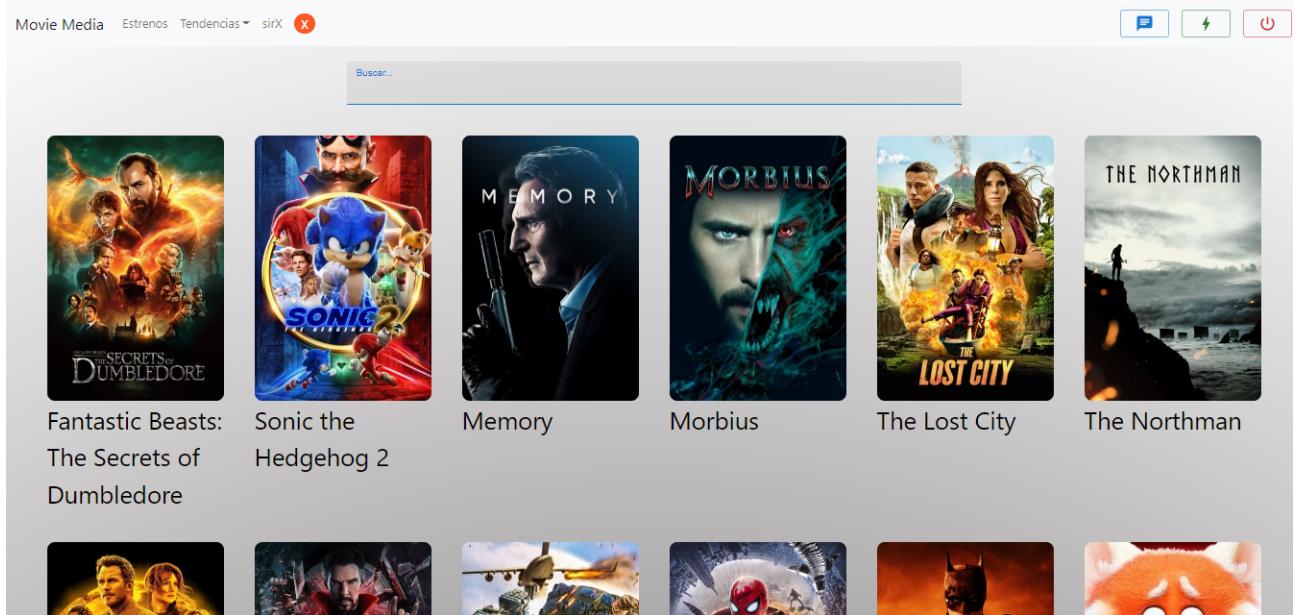
5.3.- REGISTRO.

Creando nuevo usuario:



Usuario ya existente:



5.4.- PANTALLA PRINCIPAL.

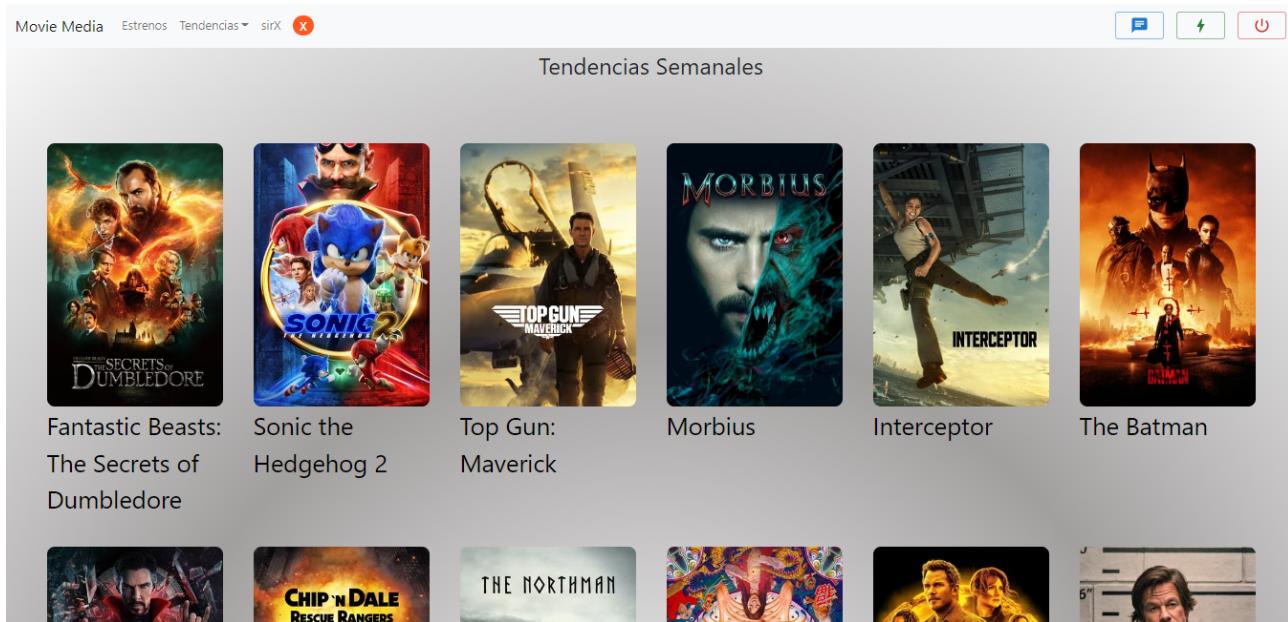
5.4.- PANTALLA ESTRENOS, WEEK, Y DAILY.

Últimos estrenos

| | | | | | |
|---|---|---|---|---|---|
|  |  |  |  |  |  |
| Memory | The Lost City | Jurassic World Dominion | A Day to Die | Top Gun: Maverick | My Hero Academia: World Heroes' Mission |

Tendencias diarias

| | | | | | |
|---|---|---|---|---|---|
|  |  |  |  |  |  |
| Interceptor | Fantastic Beasts: The Secrets of Dumbledore | Last Seen Alive | Jurassic World Dominion | Top Gun: Maverick | Hollywood Stargirl |



5.5.- TOOLBAR CON INFO DEL USUARIO.



5.6.- DETALLES DE LA PELÍCULA.

Movie Media Estrenos Tendencias sirX X



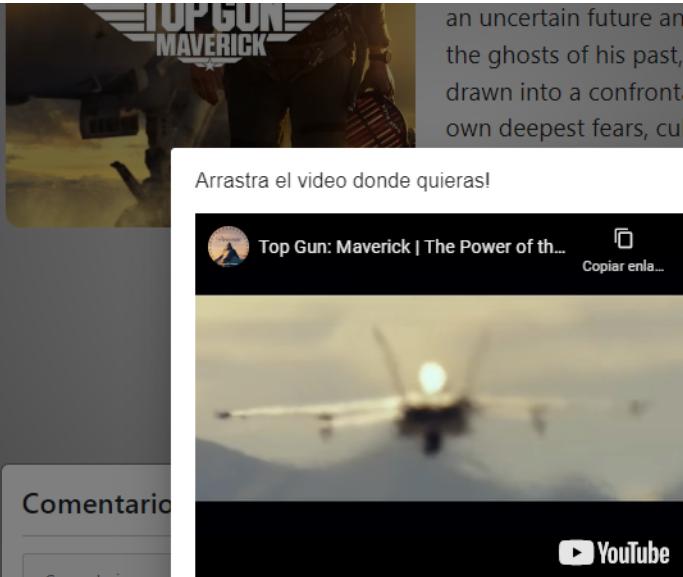
Título: Top Gun: Maverick

Descripción: After more than thirty years of service as one of the Navy's top aviators, and dodging the advancement in rank that would ground him, Pete "Maverick" Mitchell finds himself training a detachment of TOP GUN graduates for a specialized mission the likes of which no living pilot has ever seen. Facing an uncertain future and confronting the ghosts of his past, Maverick is drawn into a confrontation with his own deepest fears, culminating in a mission that demands the ultimate sacrifice from those who will be chosen to fly it.

Género: Action, Drama

Fecha de estreno: 2022-05-24

5.7.- TRAILER.



an uncertain future and confronting the ghosts of his past, Maverick is drawn into a confrontation with his own deepest fears, culminating in a

the ultimate will be

-05-24

Arrastra el video donde quieras!

Comentario

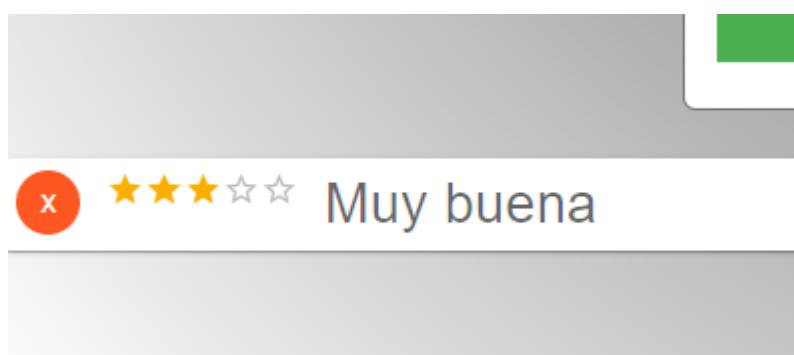
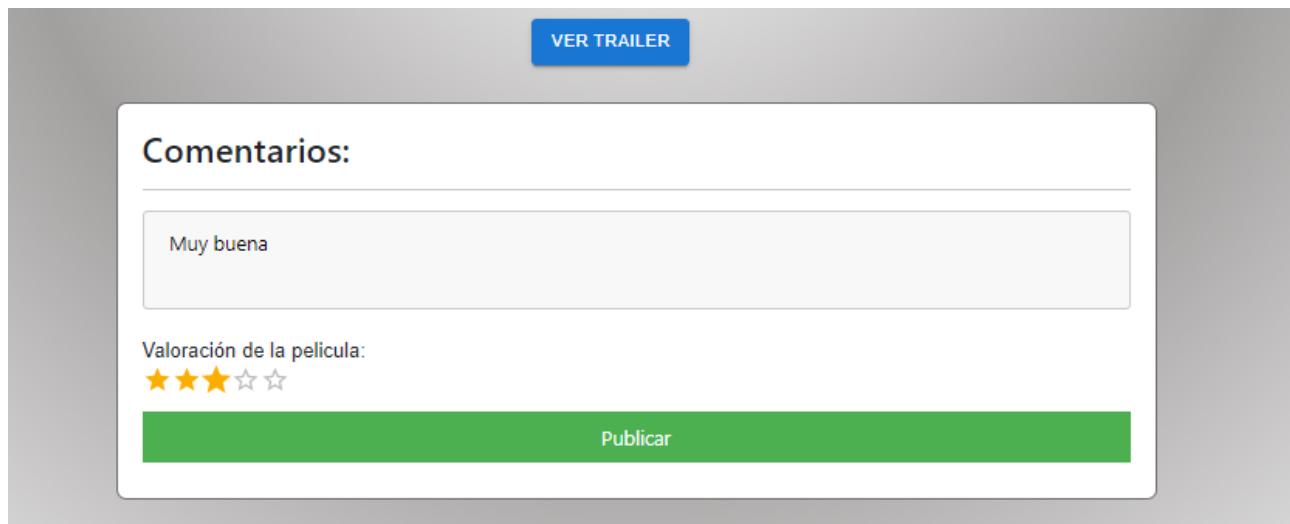
Comentario

Valoración de la película:

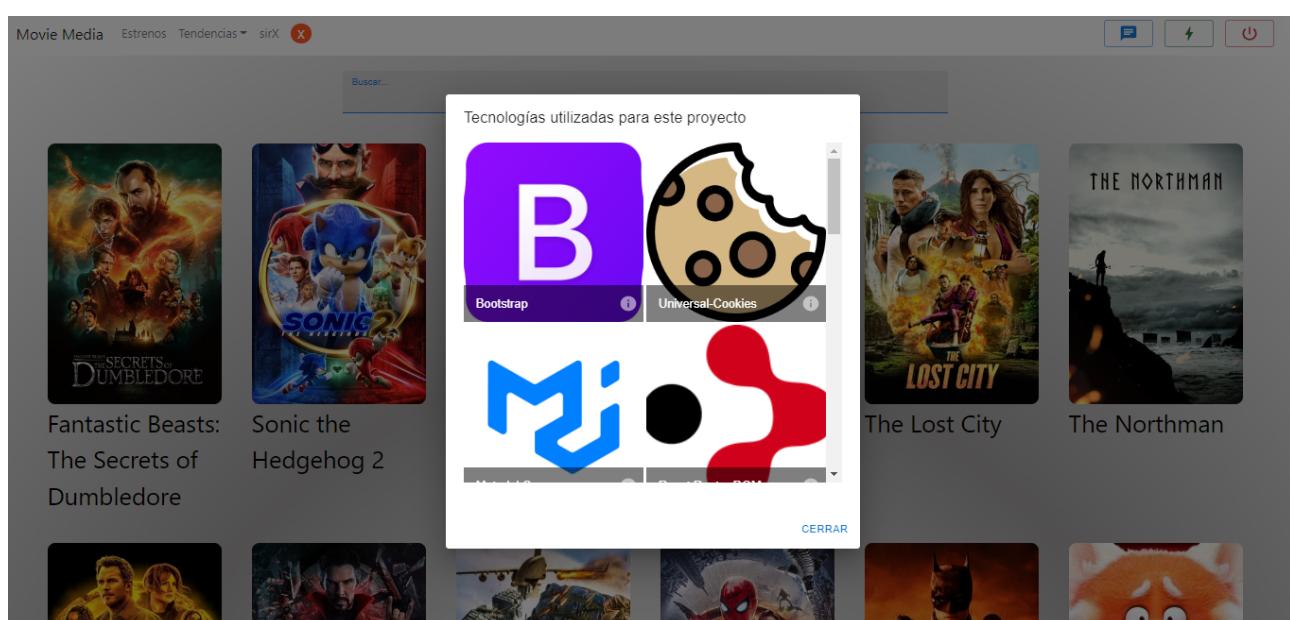
★ ★ ★ ★ ★

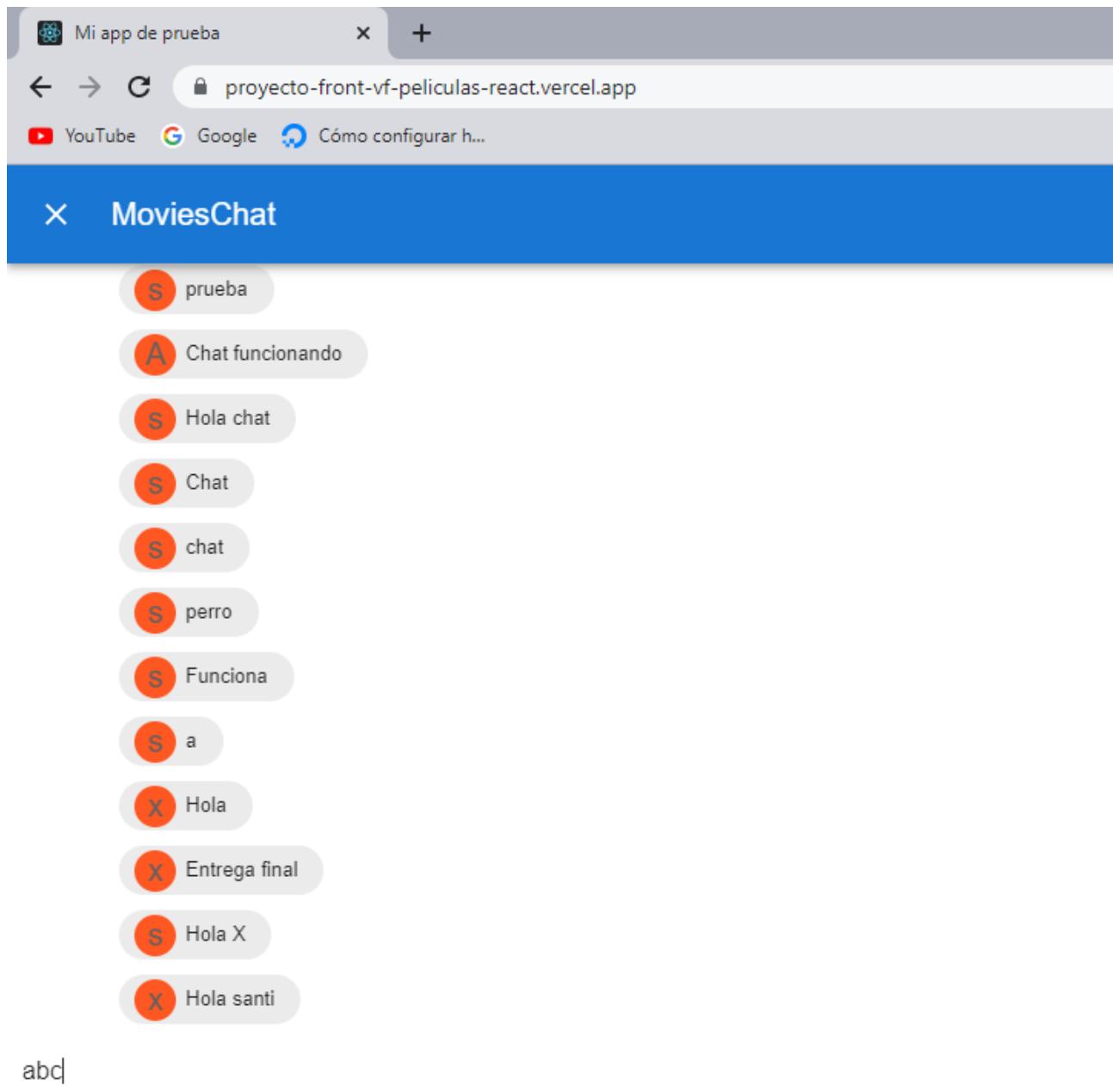
CERRAR

5.8.- COMENTARIOS.



5.7.- TECNOLOGÍAS.



5.8.- CHAT EN DIRECTO.

CAPÍTULO 6: CONCLUSIONES.

6.1.- CONCLUSIONES

Desde mi experiencia personal este proyecto me ha sido de gran utilidad para aprender a usar la librería REACT la cual actualmente es la que tiene más uso en el mercado laboral del FRONT-END.

Además he podido construir una aplicación la cual usaré como portfolio para mis futuras aplicaciones como desarrollador front-end en REACT

6.2.- PROPUESTAS FUTURAS.

Mi propuesta personal con vistas de futuro es seguir formándome en la programación web con el fin de conseguir trabajo en este nicho de mercado.

CAPÍTULO 7: BIBLIOGRAFÍA Y REFERENCIAS

7.1.-REFERENCIAS BIBLIOGRÁFICAS.

REACT: <https://reactjs.org/>

JS: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

TS:<https://www.typescriptlang.org/>

React-Bootstrap: <https://react-bootstrap.github.io/>

MUI-CORE: <https://mui.com/>

Universal cookie: <https://www.npmjs.com/package/universal-cookie>

React-router-dom: <https://v5.reactrouter.com/web/guides/quick-start>

MongoDB: <https://www.mongodb.com/>

Node.js: <https://nodejs.org/en/>

TMDB: <https://www.themoviedb.org/>

Mongoose: <https://www.themoviedb.org/>

Firebase: <https://firebase.google.com/>

Express: <https://expressjs.com/>

Youtube: <https://www.youtube.com/>

StackOverFlow: <https://es.stackoverflow.com/>

ANEXO 1: MANUAL DE INSTALACIÓN

Para la instalación de este proyecto es indispensable **tener instalado NODE.js** en una versión actual.

Una vez hecho **tomaremos el archivo .rar** donde está el proyecto comprimido.

Descomprimimos el proyecto y desde la consola nos movemos a su directorio raiz.

Ejecutamos **npm install** para instalar paquetes y dependencias.

Ejecutamos **npm start** para compilar el código con babel