

MeetSport



Nombre: Andrés Aleu Núñez

Curso: 2ºDAW B

Repositorio: <https://github.com/Andresalnz/meetSport-finalProjectDAW>

Introducción	3
Objetivo	3
Idea Inicial	3
Tecnologías utilizadas	3
React	3
Grommet	4
Nodejs	4
Expressjs	4
MongoDB y MongoDB Atlas	4
Aplicaciones similares	4
Descripción	5
Instalación y preparación	5
4. Guía de estilos y prototipado	6
Guía de estilos	6
Diseño (mockups)	7
Inicio de sesión	7
Crear cuenta	7
Pantalla Principal	8
Perfil	8
5. Diseño	9
6. Desarrollo	10
7. Pruebas	11
8. Despliegue	11
9. Manual de uso	11
Inicio de sesión	11
Crear cuenta	12
Crear publicación	13
Pantalla Principal	14
Perfil	14
Mis publicaciones	15
10. Conclusiones	15
Comparación con la idea inicial	15
Mejoras futuras	15
11. Índice tablas e imágenes	16
Uso del contexto de react	16
Ejemplos de customHooks para hacer peticiones con el uso de fetch	16
Ejemplos de controladores, para dar respuesta a las peticiones	17
Autorización por token	18
12. Bibliografía	19

1. Introducción

Objetivo

El objetivo de este proyecto era un lugar donde, deportistas pudieran “llamar” a otros deportistas para jugar un partido, correr juntos, echar el rato haciendo deporte... en definitiva conocer a otras personas con tus mismos intereses deportivos.

Idea Inicial

Cada usuario al loguearse, puede crear una publicación para buscar compañeros para hacer deporte juntos. Y poder apuntarse a cualquier publicación de cualquier usuario.

Si por ejemplo yo, como usuario, voy a jugar un partido de baloncesto con más personas, pero solo somos 8. Puedo crear una publicación donde comunique que busco 2 personas para completar los equipos y jugar un partido de baloncesto, en Cádiz, en IES Rafael Alberti, a las 6 de la tarde, el 10 del 3, gratis.

Esta publicación puedo verla tanto en la pantalla principal, como en mi perfil donde puedo eliminarla.

Tecnologías utilizadas

- Parte Front-end
 - React

Elegí, principalmente, esta biblioteca de javascript para crear interfaces de usuario, porque era la que habíamos visto en clase, además de haberla visto por mi cuenta en otros proyectos.

También por otras razones:

- **Por tener una amplia comunidad** y poder encontrar más fácilmente, la solución a cualquier error. Y aprender a cómo se realizan ciertas cosas.
- **Por estar compuesta por componentes**, ya que es muy sencillo de crearlos, pasarle información, ser reutilizables para usarlos en toda la app y poder mostrar una cosa u otra según convenga. Además gracias a esto la aplicación es más escalable y fácil de mantener.
- **Porque esta librería utiliza el último estándar de javascript**, lo que aporta al código de mayor legibilidad.
- **Se utiliza la asincronía y se realizan peticiones a una RestApi con mayor claridad.**

- Grommet

Librería de componentes.

Usandola, consigo:

- Realizar las partes en las que se compone la app con más rapidez. Con los estilos que haga falta.
- Mayor accesibilidad de los componentes.
- Código ordenado y con mayor legibilidad.
- Diseños responsive.

- Parte Back-end

- Nodejs

Entorno de ejecución para javascript orientado a eventos asíncronos.

La filosofía detrás de Node.JS es hacer programas que no bloqueen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles durante la espera

Características:

- Utiliza el motor V8
- Programación asíncrona, definiendo las acciones a realizar mediante “callbacks”.
- Podremos captar eventos como por ejemplo, data (datos enviados por una petición)
- Implementa protocolo HTTP

- Expressjs

Framework web para nodejs. Provee de los métodos habituales HTTP y varios tipo de middleware para poder controlar tanto errores, como para realizar el ciclo de solicitud-respuesta.

- MongoDB y MongoDB Atlas

Base de datos NoSQL, basada en documentos que almacena sus datos en JSON, por lo que, se hace mucho más rápido acceder a los datos, así como una mayor naturalidad de lectura.

Utilizando MongoDB Atlas, usando un clúster (set de servidores esperando a tener las bases de datos, además de tener réplicas), para poder crear mi base de datos en la nube.

Y, usando **mongoose** para poder crear esquemas, consultas, validaciones...

Aplicaciones similares

La idea surgió de una aplicación, no se si sigue en funcionamiento, de decathlon que hacía lo mismo, ponía en contacto a deportistas.

2. Descripción

La app web comienza mostrando todas las publicaciones creadas por cualquier usuario, tanto si estás logueado como si no. Dependiendo si estás logueado o no en el header se mostrarán unos links u otros.

Si no estás logueado se mostrará tres links, uno ir a la pantalla principal, otro para poder loguearte y otro para crearte una cuenta. Aunque también aparece el botón para ir a crear publicación, esté, al no estar logueado el usuario, navegará hacia el login.

- **Pantalla de login:** Formulario que pide usuario y contraseña para entrar
- **Pantalla de registro:** Formulario que pide username, email, contraseña, ciudad en la que vives y deporte favorito

Si estás logueado se mostrará tres links, uno para crear una publicación y rellenar toda la información necesaria y mostrarla en la pantalla principal, otro para ir a la pantalla principal y otro para ir a tu perfil donde se encuentra una pequeña información sobre el usuario.

- **Perfil:** Se compone de varias dos secciones.
 - La primera sección donde podemos ver el nombre de usuario y su email. Además de dos botones para cerrar sesión o para darse de baja y borrar el usuario.
 - En la segunda sección podemos ver las publicaciones creadas por el usuario logueado.

Por último, esta app está creada siguiendo la arquitectura Rest Api.

3. Instalación y preparación

Lo primero fue crear un repositorio en github y llevar un control de versión con git. También he utilizado VisualStudio code, ya que es claro y cómodo de usar, ofrece extensiones que lo hacen completo y, además tiene herramientas que hace muy fácil el uso de git.

Al clonar, hay que añadir el archivo .env en el back, con los siguientes datos:

```
MONGO_DB_URI =  
mongodb+srv://andresAdmin:yMSZyGt9fJFF9@clusterproyecto.bd6cl.mongodb.net/meet  
Sport?retryWrites=true&w=majority  
TOKENSECRET = tokenProject
```

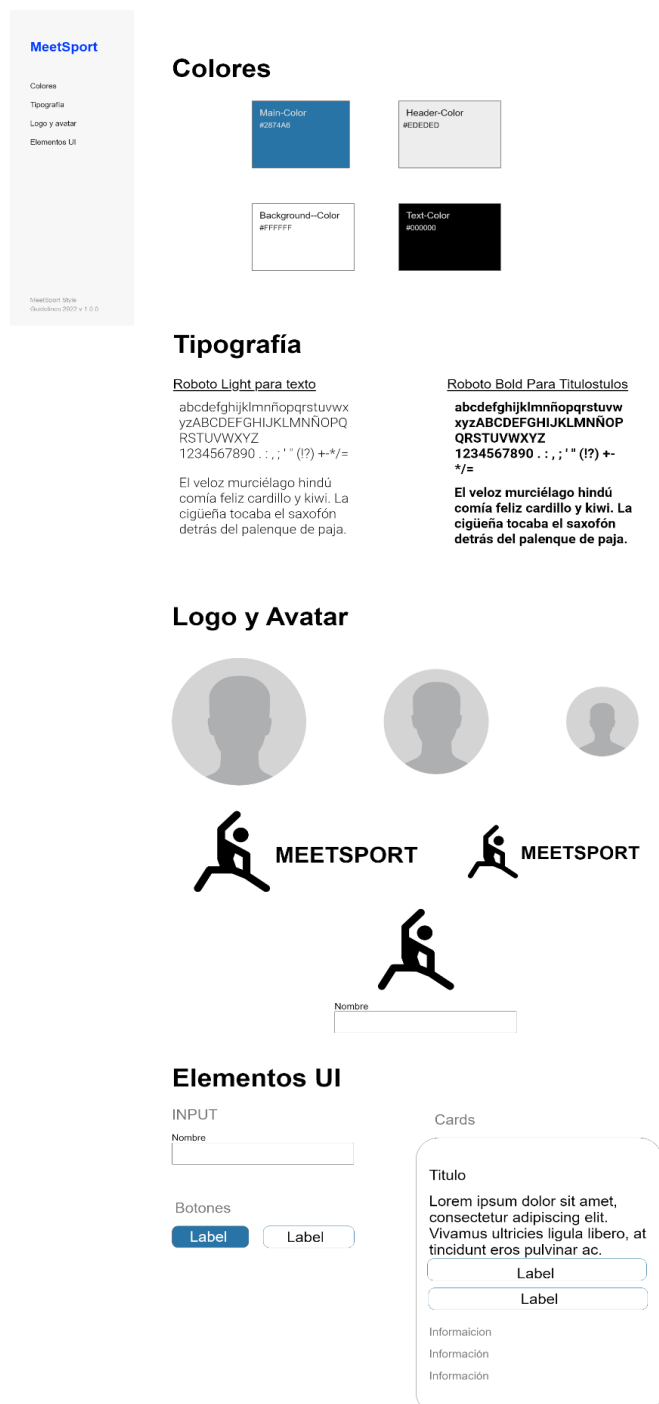
Luego hay dos maneras de ponerlo en marcha:

En las dos maneras hay que realizar **npm install** dentro de las dos carpetas. cd front/back y npm install

1. **(Recomendable)** Abrir una terminal integrada en visual Studio, entrar en la carpeta back y ejecutar **npm run dev**. Esto ejecutará la parte back y front gracias a la herramienta, concurrently.
2. Abriendo una terminal integrada en Visual Studio, entrar en la carpeta front (cd front) y ejecutar **npm start**. Hacer lo mismo con la carpeta back, es decir, abrir otra terminal integrada en visual Studio y entrar en la carpeta back (cd back) y ejecutar en este caso, **npm run dev** (Aunque esto dará error que el puerto 3000 ya está corriendo).

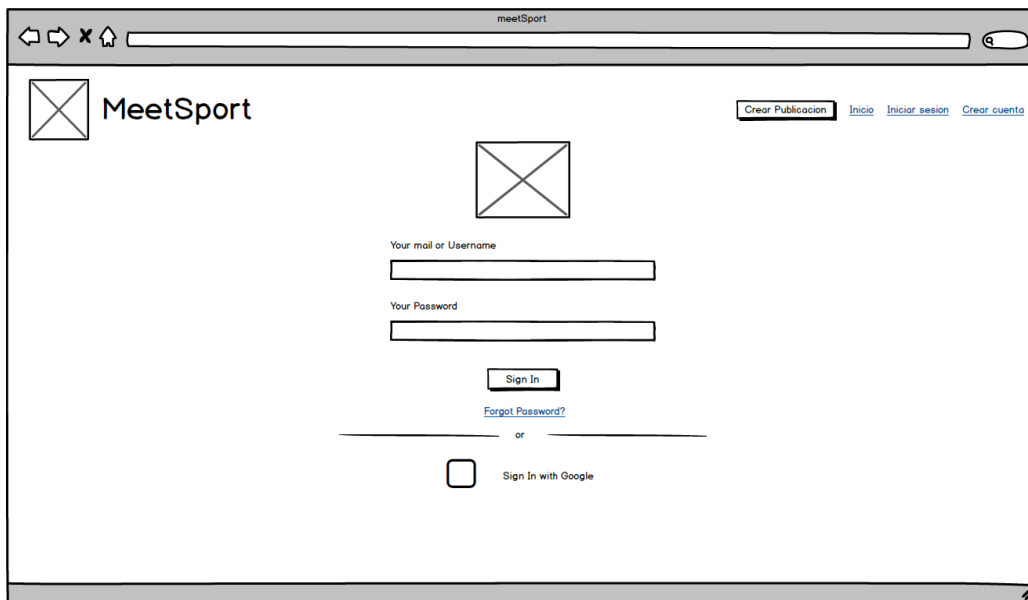
4. Guía de estilos y prototipado

Guía de estilos



Diseño (mockups)


Inicio de sesión



meetSport

MeetSport

Crear Publicacion Inicio Iniciar sesion Crear cuenta



Your mail or Username

Your Password

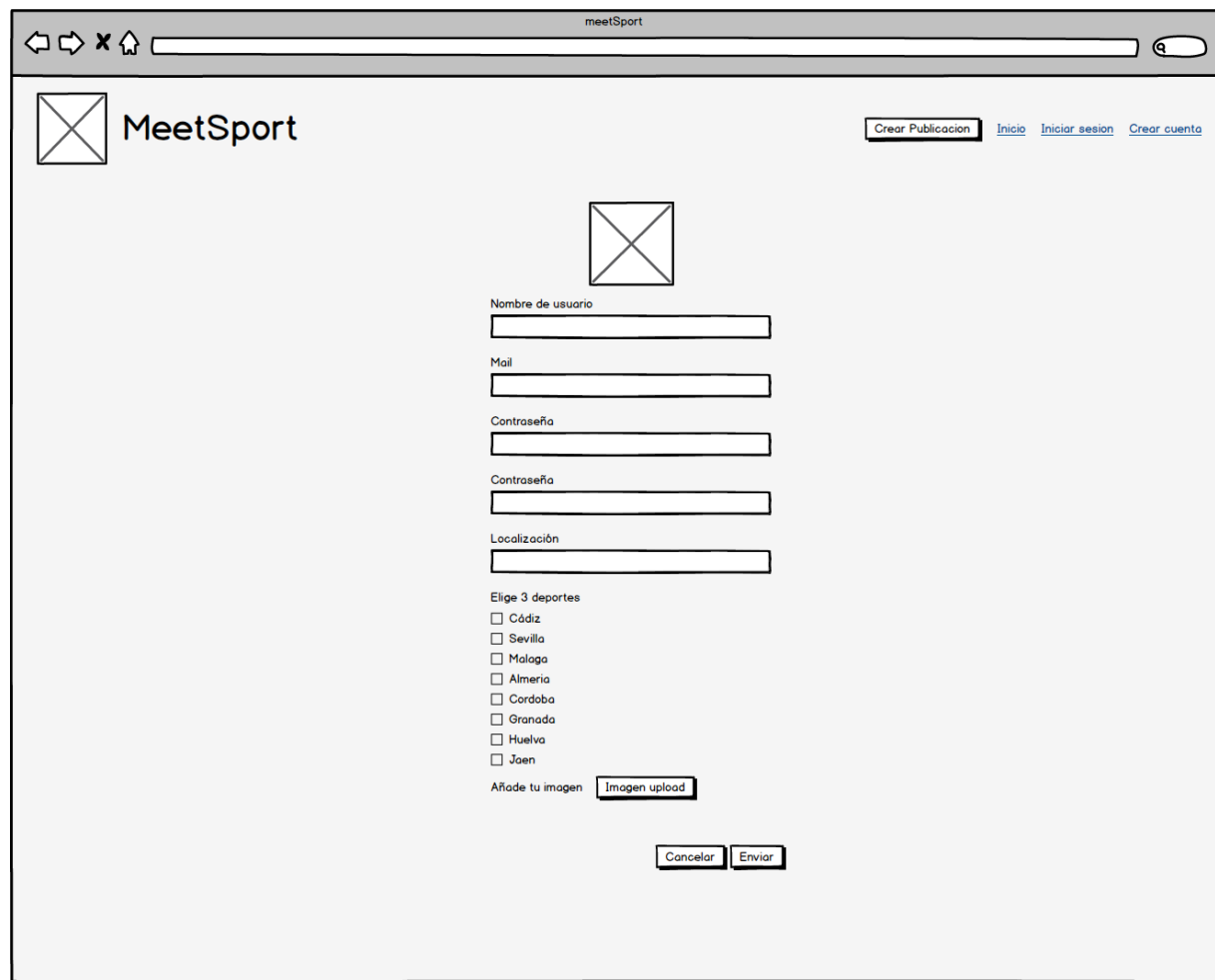
Sign In

[Forgot Password?](#)

or

☐ Sign In with Google


Crear cuenta



meetSport

MeetSport

Crear Publicacion Inicio Iniciar sesion Crear cuenta



Nombre de usuario

Mail

Contraseña

Contraseña

Localización

Elige 3 deportes

☐ Cádiz

☐ Sevilla

☐ Malaga

☐ Almeria

☐ Cordoba

☐ Granada

☐ Huelva

☐ Joen

Añade tu imagen

Pantalla Principal

meetSport

Crear Publicación

[Inicio](#) [Perfil](#)

Title of the post

Reservar

Placeholder text

Cádiz, 23/04/2021 a las 21:00

Se buscan 5 personas

Price: 150€

Join up

Title of the post

Reservar

Placeholder text

Cádiz, 27/04/2021 a las 21:00

Se buscan 5 personas

Price: Gratis

Join up

Title of the post

Reservar

Placeholder text

Cádiz, 23/04/2021 a las 21:00

Se buscan 5 personas

Price: 150€

Join up

publication

Crear Publicación

Título

Descripción de la actividad

Localización

☐ Ciudad 1

☐ Ciudad 2

☐ Ciudad 3

☐ Ciudad 4

☐ Deporte 1

☐ Deporte 2

☐ Deporte 3

☐ Deporte 4

Hora

Fecha

Participantes

Precio

Cancel

Crear publicación

Perfil

meetSport

Crear Publicación

[Inicio](#) [Perfil](#)

Mis datos

Nombre de usuario

Andrés

Email

ejemplo@gmail.com

Localización

Cádiz

Salir

Editar

meetSport

Crear Publicación

[Inicio](#) [Perfil](#)

Mis datos

Mis publicaciones

Title of the post

Reservar

Placeholder text

Cádiz, 23/04/2021 a las 21:00

Se buscan 5 personas

Price: 150€

Join up

Title of the post

Reservar

Placeholder text

Cádiz, 27/04/2021 a las 21:00

Se buscan 5 personas

Price: Gratis

Join up

Title of the post

Reservar

Placeholder text

Cádiz, 23/04/2021 a las 21:00

Se buscan 5 personas

Price: 150€

Join up

5. Diseño



6. Desarrollo

Lo primero que hice fue conectar la base de datos a mi aplicación y conexión a expressjs, además de crear el archivo index en el back, punto de entrada a la app.

Después empecé a desarrollar fueron, los esquemas de usuario y publicaciones y los controladores de cada uno con todos los endpoints para realizar las peticiones HTTP y probar tanto en insomnia, como en la extensión de visual studio, Rest.

Luego pasé al front, donde realicé el componente header, la pantalla principal y la modal para crear una publicación. Además de los hooks para hacer las peticiones de listar las publicaciones y crear publicaciones.

Cuando terminé, pasé a hacer la pantalla de Iniciar sesión y crear cuenta, con sus respectivos hooks para hacer las peticiones al servidor. Cuando estas pantallas estuvieron acabadas, hice la última que quedaba, la de perfil, usando el hook para listar las publicaciones del usuario.

Luego pase a generar el token, cuando el usuario iniciará sesión, en este punto me encontré con el problema de que sí conseguía generar el token en el back, pero no conseguía capturarlo desde la parte front, mediante las cookies. Así que buscando información, me encontré con el sessionStorage, que lo utilicé para guardar el token y el contexto de react, donde pude setear la información con el token y poder utilizarlo donde me interesara. Con esto conseguí, renderizar el componente header con distintos links, según si está logueado o no, además de “proteger” el boton de crear publicación, consiguiendo que si no esta logueado te redirija hacia el inicio de sesión

Cuando ya conseguí generar y obtener el token, pasé a relacionar las colecciones con el método populate de mongoose. Aquí me encontré con otro problema, ya que leía la documentación y buscaba otros ejemplos y mi código parecía estar bien. Consulté con Javier Ortega, y me explicó lo que faltaba, pasar el id de una colección a la otra para saber qué colección estaba relacionando. Por ejemplo, cuando creaba una publicación y la relacionaba con la de usuario tuve que pasar el id de usuario a la colección de publicaciones. Así conseguía que cada publicación tuviera su usuario y que cada usuario tuviera un listado de publicaciones. Además de pasar los endpoints de crear usuario y crear publicación de promesas a async/await.

Gracias al sessionStorage y al contexto de React utilizado anteriormente, pude guardar y obtener el id de usuario y utilizarlo donde se necesitara.

En este momento completé las colecciones con dos más (localizaciones y deportes) las cuales relacioné con usuarios y publicaciones. Además de añadir el listado de localizaciones en crear cuenta y crear publicación y el listado de deporte en crear publicación.

Añadí toda la lógica para poder borrar cualquier publicación creada por el usuario logueado, donde me encontré con otro problema.

Esté surgió al crear una modal de confirmación que se genera al pulsar en el botón de la tarjeta para asegurar que el usuario está seguro de borrarla o de apuntarse. No supe cómo realizar dos acciones distintas para una misma acción de un botón. Entonces, leyendo documentación y viendo algunos ejemplos, comprendí que debía utilizar una función que hiciera lo que deseara y pasarla mediante props hacia el componente de la modal de confirmación, en el que hay otra función que según lo que le llegue hace una acción u otra.

Por último hice la lógica para cerrar sesión y borrar usuario.

Cómo herramienta de control de versiones he utilizado git, ya que es el más utilizado actualmente, cuenta con buena documentación y es fácil integrarlo con visual studio.

7. Pruebas

Por cada componente y/o página terminada en react y su lógica en la parte back, realizaba pruebas manualmente para comprobar que hacía lo que esperaba.

8. Despliegue

Para poner en marcha el proyecto he utilizado **npm (Node package manager)**, ya que es lo que se utiliza tanto en react y node por defecto.

Como dije anteriormente con el comando **npm run dev** dentro de la carpeta back ponemos en funcionamiento el proyecto.

9. Manual de uso

Veremos con varias capturas, como moverse por la aplicación web.

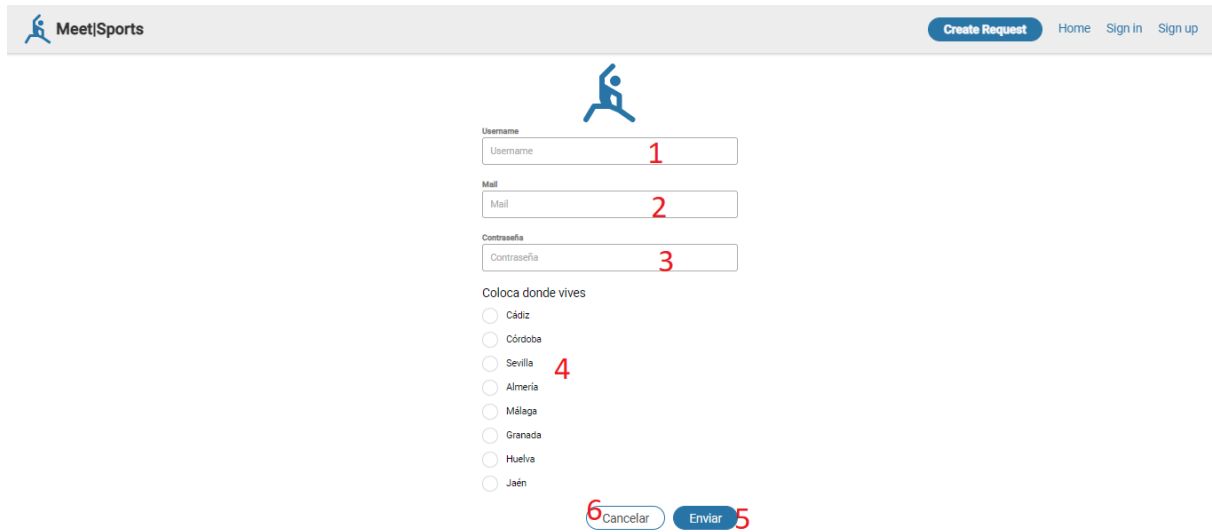
Al entrar en la aplicación nos veremos con la pantalla de login. Donde puedes acceder a loguearte o a crear una cuenta.

1. Input para escribir tu nombre de usuario
2. Input para escribir tu contraseña
3. Botón para verificar si los datos son correctos, si no se mostrará un aviso.

◆ Email o password incorrecto

Si no tienes cuenta, puedes acceder a la pantalla de crear cuenta desde el link del header.

Crear cuenta



MeetSports

Create Request Home Sign in Sign up

Username 1

Mail 2

Contraseña 3

Coloca donde vives

☐ Cádiz

☐ Córdoba

☒ Sevilla 4

☐ Almería

☐ Málaga

☐ Granada

☐ Huelva

☐ Jaén

6 Cancelar 5 Enviar

1. Input para escribir tu nombre de usuario
2. Input para escribir tu email
3. Input para escribir tu contraseña
4. Elegir tu localización
5. Botón para enviar los datos
6. Botón para cancelar los datos y deja en blanco todos los inputs

Cuando te creas la cuenta y te logas, ya puedes crear una publicación.

Crear publicación

Create a request

Title

Basket session with friends... 1

Description of activity

Nos hacen falta 3 tios más para un parti... 2

Location

Campo hondo 3

☐ Baloncesto

☐ Futbol

☐ Balonmano

☐ Runner

☐ Ciclismo

☐ Tennis

☐ Cádiz

☐ Córdoba

☐ Sevilla

☐ Almería

☐ Málaga

☐ Granada

☐ Huelva

☐ Jaén

4

dd/mm

5

hh:mm ap

6

Participants

11 7

Price

1.50 8

10 Cancel

9 Send request

1. Input para introducir el título
2. Input para introducir la descripción
3. Input para introducir la localización
4. Para señalar para qué deporte va destinada la publicación y señalar en qué ciudad está la localización.
5. Input para introducir la fecha
6. Input para introducir la hora
7. Input para introducir los participantes que se busca
8. Input para colocar el precio

Cuando creas una publicación está sale tanto en la pantalla principal, como en tu perfil.

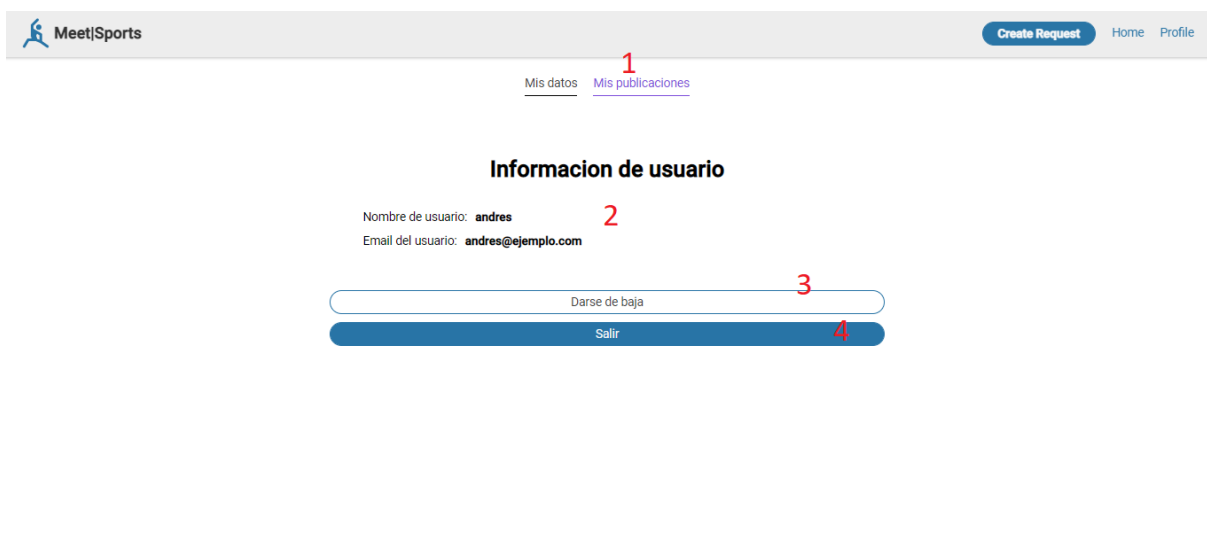
Pantalla Principal



1. Listado de publicaciones de todos los usuarios
2. Botón para unirse a la publicación
3. Link que redirige al perfil
4. Botón que muestra la modal para crear una publicación

Al darle al Link de perfil este redirige hacia el perfil.

Perfil



1. Tab que lleva a mirar las publicaciones creada por el usuario registrado
2. Información del usuario
3. Botón para darse de baja y borrar el usuario y sus publicaciones
4. Botón para salir, que borra del sessionStorage el token y el userId

Mis publicaciones



1. Tab que lleva a la pantalla anterior, perfil.
2. Listado de publicaciones creadas por el usuario
3. Botón para eliminar la publicación. Redirige hacia la principal.

10. Conclusiones

Comparación con la idea inicial

En comparación de la idea inicial con el resultado final, ha quedado algunas cosas por hacer:

- Lógica del botón apuntarse
- Mostrar las publicaciones a las que se ha apuntado el usuario
- Poder actualizar usuario
- Poder actualizar publicación

Mejoras futuras

1. Hacer funcionar el botón para que los usuarios puedan apuntarse a las publicaciones y no solo ver el listado.
2. Hacer que se pueda actualizar publicaciones y usuarios.
3. Modo oscuro.
4. Buscador para filtrar por deporte, ubicación o usuario.
5. Poder logearte con alguna red social
6. Introducir un chat o algún sistema de comunicación para los usuarios que se apunten a una publicación.
7. Hacer que el usuario pueda elegir más de un deporte al crear su cuenta

11. Índice tablas e imágenes

```
export const AuthContext = createContext(null)

export const AuthProvider = ({children}) => {
  const valueToken = sessionStorage.getItem('token') //
  const valueUserId = sessionStorage.getItem('userId')
  const [user, setUser] = useState({token: valueToken, userId: valueUserId});

  return (
    <AuthContext.Provider value={{user, setUser}}>
      {
        children
      }
    </AuthContext.Provider>
  )
}

export const RequireAuth = ({children}) => {
  const location = useLocation()
  const {user} = useContext(AuthContext)
  if(!user.token) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }
  return children
}

export const useAuth = () => useContext(AuthContext)
```

```
function App() {
  return (
    <AuthProvider>
      <Grommet theme={theme}>
        <Router>
          <Routes>
            <Route path="/" element={<Layout />} />
            <Route index element={<Home />} />
            <Route path="home" element={<Home />} />
            <Route path="create-request" element={<RequireAuth> <CreateRequest /></RequireAuth>} />
          </Routes>
          <Route path="create-request" element={<Home />} />
          <Route index element={<RequireAuth> <CreateRequest /></RequireAuth>} />
          <Route path="sign-up" element = {<SignUp />}</Route>
          <Route path="login" element = {<SignIn />}</Route>
          <Route path="profile" element = {<Profile />}</Route>
        </Routes>
      </Router>
    </Grommet>
  </AuthProvider>
);
}
```

Ejemplos de customHooks para hacer peticiones con el uso de fetch

```
export default function useListRequest() {
  const [listRequest, setListRequest] = useState([])
  useEffect(() => {
    fetch('http://localhost:3001/publication')
      .then(result => result.json())
      .then(result => {
        setListRequest(result)
      })
  }, [])
  return listRequest
}

export default function useRemoveUser() {
  const [userId, setUserId] = useState(null);
  useEffect(() => {
    if (userId) {
      async function RemoveUserwithId(url = '', dataId = {}){
        console.log('1')
        await fetch(url, {
          method: 'DELETE',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify(dataId)
        })
      }
      console.log('ideeeee ', userId)
      if (userId) {
        RemoveUserwithId(' http://localhost:3001/user/delete', {id: userId})
        sessionStorage.removeItem('token')
        sessionStorage.removeItem('userId')
        document.location.href = 'http://localhost:3000/login';
      }
    }
  }, [userId])
  return setUserId
}
```

```
export default function useCreateUser() {
  const [request, sendRequest] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    async function postUser(url = '', data = {}) {
      await fetch(url, {
        method: 'POST',
        mode: 'cors',
        headers: {
          'Content-Type': 'application/json',
          'Access-Control-Allow-Origin': '*'
        },
        body: JSON.stringify(data)
      });
    }

    if (request) {
      postUser('http://localhost:3001/user/signup', request);
      navigate('/login')
    }
  }, [request]);

  return sendRequest;
}
```


Desde la parte back podemos poner los siguientes ejemplos

Ejemplos de controladores, para dar respuesta a las peticiones

Esta primera (login) genera el token si la credenciales son válidas

```
router.post('/signin', async (request, response) => {
  const { body } = request
  const { username, password } = body
  const user = await userModel.findOne({username})
  const passwordCorrect = user === null ? false : await bcrypt.compare(password,user.passwordHash)

  if (!(user && passwordCorrect)) {
    response.status(401).send({error:"Invalido"})
  }

  const userForToken = {
    id:user._id,
    username:user.username,
  }
  const token = jwt.sign(userForToken,process.env.TOKENSECRET, {
    expiresIn: 60*60*24*7
  })

  response.send({
    username:user.username,
    id: user.id,
    token
  })
})

module.exports = router
```

```
//Update user
router.put('/account', (request, response) => {
  const {body} = request
  const {id, username, mail, location, sports} = body
  const options = { new: true, rawResult: false } //rawResult: Para verif
  const filter = { _id:id }
  const updateUser = {
    username,
    mail,
    location,
    sports
  }
  userModel.findOneAndUpdate(filter, updateUser, options)
    .then(result => { response.status(200).send(result) })
    .catch(err => { response.send(err.name) })
})

//Delete user
router.delete('/delete', (request,response, next) => {
  const {id} = request.body
  userModel.findByIdAndDelete({_id:id})
    .then(result => { response.status(200).send(result) })
    .catch(error => next(error))
})

//search user by name or sport
router.get('/:search',(request, response) => {
  const {search} = request.params
  userModel.find({$or:[{username:search},{sports:search}]})
    .then(result => { response.status(200).send(result) })
    .catch(err => { response.send(err.name) })
})
```

```
//Create user
router.post('/signup', async (request, response, next) => {
  const {body} = request
  const {username, password, mail, sports, image, locationId} = body
  const salt = 10
  const passwordHash = await bcrypt.hash(password, salt)

  const filter = {name:locationId}
  const location = await locationModel.findOne(filter)
  console.log(location)

  const newUser = new userModel({
    username,
    passwordHash,
    mail,
    location,
    sports,
    image,
    locationId: location._id
  })

  try {
    const savedUser = await newUser.save()
    location.user = location.user.concat(savedUser)
    await location.save()
    response.status(200).send(savedUser)
  } catch (error) {
    next(error)
  }
})
```

```
router.get('/', (request, response, next) => {
  userModel.find({}).populate('publications')
    .then(result => {
      response.send(result)
    })
    .catch(err => {
      next(err.name)
    })
})

router.get('/:id', (request, response, next) => {
  console.log('aquí estamos')
  const {id} = request.params
  userModel.findById({_id:id})
    .then(result => {
      response.status(200).send(result)
    })
    .catch(err => {
      next(err.name)
    })
})
```

Autorización por token

```
module.exports = (request, response, next) => {  
  
  const authorization = request.get('authorization')  
  
  let token = ''  
  if (authorization && authorization.toLowerCase().startsWith('bearer ')){  
    token = authorization.split(' ')[1]  
  }  
  
  const decodeToken = jwt.verify(token, process.env.TOKENSECRET)  
  
  if(!token || !decodeToken.id) {  
    return response.status(401).send({error:"No tienes token o es invalido"})  
  }  
  
  const {id:userId} = decodeToken  
  request.userId = userId  
  
  next()  
}
```

```
router.delete('/delete', userToken, (request, response, next) => {  
  console.log('first')  
  const {id} = request.body  
  console.log('id publi,', id)  
  const filter = { _id:id }  
  const options = { new: true, rawResult: true } //rawResult: Para ver el resultado original  
  publicationModel.findByIdAndDelete(filter, options)  
    .then(result => { response.status(200).send(result) })  
    .catch(error => { next(error) })  
})
```



Usuarios

Username: andres

Password: 12345

Username: rafa

Password: rafael

Username: jose

Password: jose

12. Bibliografía

Documentación de mongoose. <https://mongoosejs.com/>

Documentación de mongodb. <https://docs.mongodb.com/>

Base de datos en la nube con mongodb Atlas.

<https://cloud.mongodb.com/v2/616d97af2000bb07d70fe586#metrics/replicaSet/61ec50064df763345e5b8be1/explorer/meetSport/publications/find>

Documentación de React. <https://es.reactjs.org/docs/getting-started.html>

Documentación de react router. <https://reactrouter.com/docs/en/v6>

Documentación de la librería grommet <https://v2.grommet.io/>

Storybook de la librería grommet <https://storybook.grommet.io/?path=/story/all--all>

Documentación de npm, <https://www.npmjs.com/>

Documentación de Expressjs. <http://expressjs.com/>

Documentación de node. <https://nodejs.org/es/>

Publicación sobre node. <https://desarrolloweb.com/articulos/caracteristicas-nodejs.html>

