

INVESTIGACIÓN DE NUEVOS LENGUAJES Y SU USO EN CREACIÓN WEB



Autora: Natividad Fernández Alcalá

Año: 2022

Instituto: IES Al-Ándalus

Ciclo: Desarrollo de aplicaciones web

Índice

| | |
|--|----|
| 1.- Introducción..... | 3 |
| 2.- Análisis del entorno..... | 3 |
| 3.- Análisis del sistema actual..... | 3 |
| 4.- Solución propuesta..... | 4 |
| 5.- Planificación temporal del desarrollo del proyecto..... | 6 |
| 6.- Estudio de la viabilidad del proyecto..... | 7 |
| 7.- Documentación del diseño e implementación de la solución adoptada..... | 7 |
| 8.- Código fuente documentado..... | 15 |
| 9.- Manual de configuración y funcionamiento de la aplicación..... | 40 |
| 10. Manual de usuario..... | 45 |
| 11.- Bibliografía y fuentes de información..... | 49 |

1. Introducción

Se había detectado una carencia en la recogida y gestión de datos en el ámbito de las empresas objeto de prácticas a realizar por el alumnado y el instituto.

Los pocos datos registrados se recogían en una hoja de excel siendo necesario un sistema más preciso y dinámico además de intuitivo y de manejo simple.

Se aborda este proyecto con el desconocimiento de los lenguajes React, Next js y GraphQL y la gestión de base de datos MongoDB, encontrándonos con la doble tarea de profundizar en su conocimiento y redactar un proyecto en base a la investigación de estos lenguajes, lo que nos obligaba a movernos en el ensayo error constantemente.

Ha sido esta situación tan complicada como satisfactoria por el grado de autoformación que me ha permitido y que suma naturalmente a lo que ya se había aplicado en proyectos anteriores.

2.- Análisis del entorno

Esta aplicación se desarrolla en base a las necesidades de recogida y registro de datos en cuanto a las empresas ofertantes de prácticas y los ciclos formativos que son de aplicación para las mismas.

Siendo de utilización para el instituto IES Al-Ándalus y con posibilidad de ser utilizado por otras instituciones.

3.- Análisis del sistema actual

Actualmente se trabaja sobre una página de excel, donde se reflejan de forma estática los datos necesarios tal y como se refleja en la imagen de abajo, obligando a hacer cada vez que se incorpora una empresa de prácticas su propia hoja de excel.

Como se puede apreciar no existe ningún tramo en el que aparezca expresamente los ciclos formativos que se pueden practicar en la empresa.

Este diseño no nos permite filtrar las empresas por los ciclos formativos que pueden ofrecer.

Siendo susceptible este procedimiento de excel de perderse o borrarse con más facilidad, al mismo tiempo de poder introducir datos erróneos al carecer del proceso de validación de datos.

| | |
|---------------------|--|
| Nombre Empresa | |
| CIF | |
| Representante legal | |
| NIF | |
| email | |
| Teléfono | |
| | |
| Tutor laboral | |
| NIF | |
| email | |
| Teléfono | |
| | |
| CONTACTOS | |
| Nombre Apellidos | |
| Email | |
| Teléfono | |
| | |
| Nombre Apellidos | |
| Email | |
| Teléfono | |
| | |
| Observaciones | |
| | |

4.-Solución propuesta

Detectadas las carencias de este sistema se propone una mejora de la recogida, archivo y visualización de los datos, eliminando la rigidez de la página de excel y creando una página web totalmente dinámica y versátil.

Tecnologías existentes para el desarrollo de la página web:

Node.js → Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.



GraphQL → Es un lenguaje de consulta y un tiempo de ejecución del servidor para las interfaces de programación de aplicaciones (API); su función es brindar a los clientes exactamente los datos que solicitan y nada más.



Gracias a GraphQL, las API son rápidas, flexibles y sencillas para los desarrolladores. Incluso se puede implementar en un entorno de desarrollo integrado (IDE) conocido como GraphiQL. Como alternativa a REST, GraphQL permite que los desarrolladores creen consultas para extraer datos de varias fuentes en una sola llamada a la API.

Apollo Server → Es una implementación de servidor GraphQL para JavaScript, en particular para el Node.js plataforma. Es compatible con muchos marcos populares de Node.js.

Apollo Client → Es un cliente JavaScript para GraphQL diseñado poder crear componentes que hagan uso de GraphQL. Estos componentes podrán obtener y mostrar datos o también realizar cambios o mutaciones cuando ocurran ciertas acciones.

Apollo Server nos da básicamente 3 cosas:

- Nos da una forma de describir nuestros datos con un esquema.
- Proporciona el marco para resolutores, que son funciones que escribimos para obtener los datos necesarios para cumplir con una solicitud.
- Facilita el manejo autenticación para nuestra API.



MongoDB → Es un sistema de base de datos NoSQL, orientado a documentos y de código abierto.

En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.



MongoDB Atlas → Es un servicio de Cloud Database (o Base de Datos en la Nube), que te permite crear y administrar tu BBDD Mongo desde cualquier lugar del mundo, a través de su plataforma.

Mongoose → Mongoose es una librería para Node.js que nos permite escribir consultas para una base de datos de MongoDB, con características como validaciones, construcción de queries, middlewares, conversión de tipos y algunas otras, que enriquecen la funcionalidad de la base de datos.



Next.js → Es un framework de JavaScript que nos permite crear fácilmente sitios web de React listos para salir a producción.



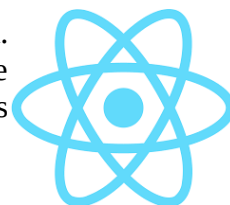
Tailwind → Es un framework CSS que permite un desarrollo ágil, basado en clases de utilidad que se pueden aplicar con facilidad en el código HTML y unos flujos de desarrollo que permiten optimizar mucho el peso del código CSS.



Formik → Es una librería declarativa, intuitiva y adaptable para formularios con React, además utiliza Yup para la validación de estos formularios.



React → Ayuda a crear interfaces de usuario interactivas de forma sencilla. Diseña vistas simples para cada estado en tu aplicación, y React se encargará de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien.



SweetAlert → Es un plugin de jQuery y con el cual podremos dar un aspecto profesional a los mensajes que lancemos a los usuarios acorde a las tendencias actuales. Además, tenemos la posibilidad de configurar el plugin de muchas formas diferentes.



Materiales necesarios para el desarrollo del proyecto:

- Imprescindible en este caso formación en React, Next js, MongoDB, Apollo, GraphQL, Formik, Tailwind que hasta el momento no se tenía.
- Un ordenador y dos pantallas

Recursos Humanos

- Una programadora (a poder ser que no este estresada)

5.- Planificación temporal del desarrollo del proyecto

PLANIFICACIÓN TEMPORAL DEL DESARROLLO DEL PROYECTO

| Desarrollo de la aplicación web. | Puesta a punto de la web. | Fase de pruebas . | Formación de usuarios finales . | Modificaciones o ampliaciones futuras . | Plan de mantenimiento de la web . |
|----------------------------------|-------------------------------------|-------------------------------------|------------------------------------|--|---|
| 2 semanas | 1 semana | 4 / 5 días | 2/ 3 días | A petición del instituto | A contar a partir final fase pruebas |
| | Se hará trabajando en el instituto. | Se hará trabajando en el instituto. | Se formará a los usuarios actuales | La modificaciones o ampliaciones futuras derivadas de la ampliación de las necesidades del instituto serán objeto de estudio | Como norma general se chequeará la aplicación cada tres meses el primer año, o a demanda del instituto. Después anualmente, o a demanda del instituto |

6.- Estudio de la viabilidad del proyecto.

Hasta ahora el instituto no había intentado implementar una web con las características que aquí se han gestionado, para responder a una necesidad real de crear una base de datos que permita la conectividad, agilidad y modificación de los datos recogidos.

Este nuevo sistema permite una mejor concentración de los datos, siendo una solución fiable y sostenible. No necesitando de una capacidad técnica costosa y de fácil implementación y de un uso muy accesible e intuitivo.

No se le aplican costes puesto que la web proviene de un estudio de investigación aplicado a la necesidad detectada en el instituto.

A largo plazo el mantenimiento y actualización no suponen una gran inversión ni económica ni en tiempo.

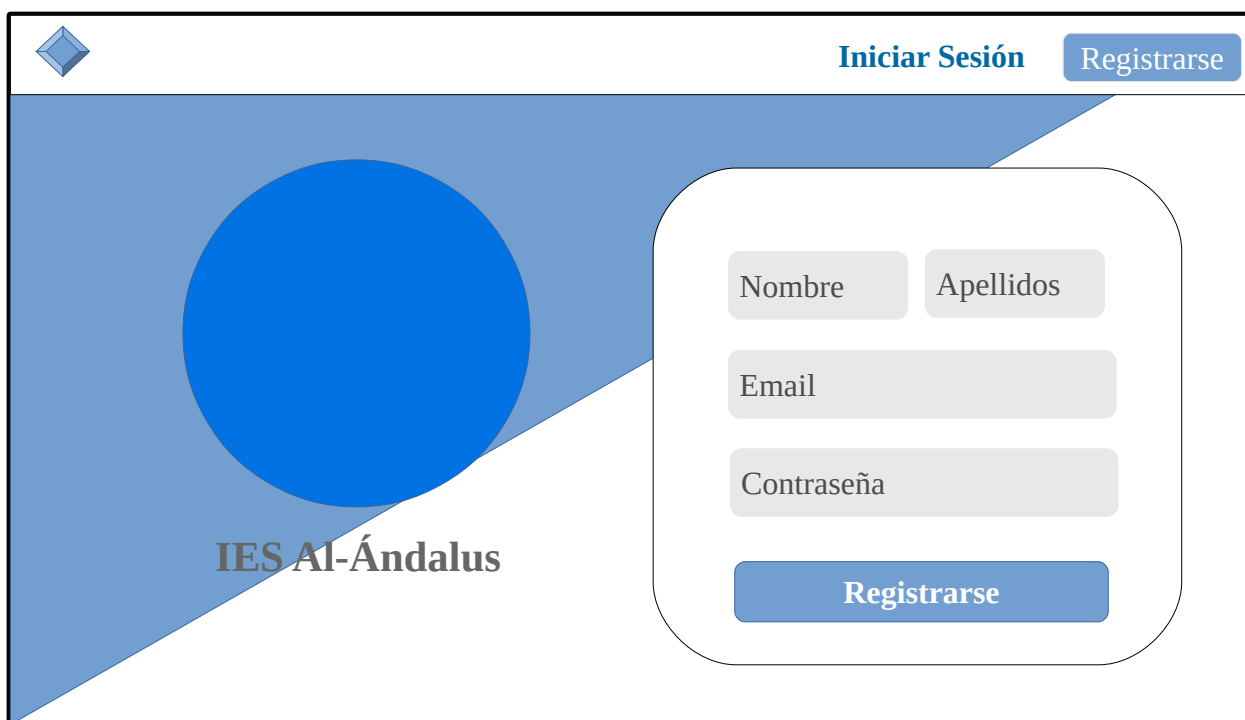
7.- Documentación del diseño e implementación de la solución adoptada.

- *Diseño de la interfaz Web*

Iniciar Sesión

The image shows a web interface design for a login page. At the top, there is a navigation bar with a blue diamond logo on the left, the text "Iniciar Sesión" in blue, and a blue button labeled "Registrarse". Below the navigation bar, the main content area has a light blue background with a large blue circle on the left. To the right of the circle, there is a white rounded rectangle containing the login form. The form has two input fields: "Email" and "Contraseña". Below the "Contraseña" field, there is a link that says "¿Has olvidado la contraseña?". At the bottom of the form is a blue button labeled "Iniciar Sesión". The text "IES Al-Ándalus" is written in a serif font at the bottom left of the main content area.

Registrarse




The image shows a web interface for IES Al-Ándalus. On the left, there is a blue circle logo and the text "IES Al-Ándalus". On the right, there is a registration form with the following fields: "Nombre" (Name), "Apellidos" (Surnames), "Email", and "Contraseña" (Password). Below these fields is a blue button labeled "Registrarse". At the top right, there are two buttons: "Iniciar Sesión" (Login) and "Registrarse" (Register).

Pantalla de Inicio



The image shows the home page of the IES Al-Ándalus system. On the left, there is a sidebar with a blue circle logo and the text "IES Al-Ándalus". Below the logo, there is a menu with the following items: "Inicio" (Home), "Empresas" (Companies), "Ciclos" (Cycles), and "Configuración" (Configuration). On the right, there is a main content area with the text "!Hola;" (Hello!). Below this, there is a large blue cloud graphic. In the center, there is a box with the text "Gestión Prácticas" (Practice Management) and "Formación Profesional" (Vocational Training). Below this box is a blue button labeled "Registrarse" (Register). To the right of the button, there is a link labeled "Ciclos" (Cycles). At the top right, there is a button labeled "Cerrar Sesión" (Logout).

Empresas



IES Al-Ándalus

Inicio

Empresas

Ciclos


Configuración
















Empresas

Nueva Empresa

Ciclos

Buscador




| Nombre columnas base de datos | | | |
|-------------------------------|---|---|---|
| Datos |  |  |  |
| ... |  |  |  |
| ... |  |  |  |
| ... |  |  |  |
| ... |  |  |  |

Cerrar Sesión

 Información Empresa  Editar  Eliminar

Insertar/ Modificar: Empresas



IES Al-Ándalus

Inicio

Empresas

Ciclos

Configuración

Nueva/ Editar Empresa

Cerrar Sesión

Datos empresa

Datos representante


Ciclos

Añadir/ Actualizar

NOTA:

Eliminar es un modal que nos aparece preguntándonos si queremos borrar o no la empresa.

Información Empresa



IES Al-Ándalus

Inicio

Empresas

Ciclos

Configuración

Nueva/ Editar Empresa

Cerrar Sesión

Nombre empresa

Datos empresa

Datos

Datos representante

Datos

Empleados

Empleado 1


Empleado 2

Ciclos

Datos

INFORMACIÓN: En la sección de empleados, nos aparece un listado de todos los empleados que están asignados a esa empresa, además, en esa sección podemos añadir nuevos empleados, editarlos o borrarlos.

Ciclos



IES Al-Ándalus


- Inicio
- Empresas
- Ciclos**
- Configuración


Ciclos

Cerrar Sesión



Nuevo Ciclo


Buscador








Nombre Corto Ciclo
Nombre
Largo Ciclo







Nombre Corto Ciclo
Nombre
Largo Ciclo






Nombre Corto Ciclo
Nombre
Largo Ciclo



 *Editar*  *Eliminar*

Insertar/ Modificar: Ciclos




IES Al-Ándalus

- Inicio
- Empresas
- Ciclos**
- Configuración

Nuevo/ Editar Ciclo

Cerrar Sesión



Abreviatura Ciclo

Rellenar datos

Nombre Completo Ciclo

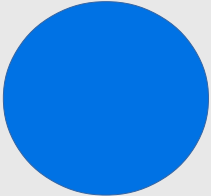
Rellenar datos

Añadir/ Actualizar

NOTA:

Eliminar es un modal que nos aparece preguntándonos si queremos borrar o no la empresa.

Configuración



IES Al-Ándalus

Inicio

Empresas

Ciclos

Configuración

Información Usuario

Cerrar Sesión

Nombre

Apellidos

Email

Datos

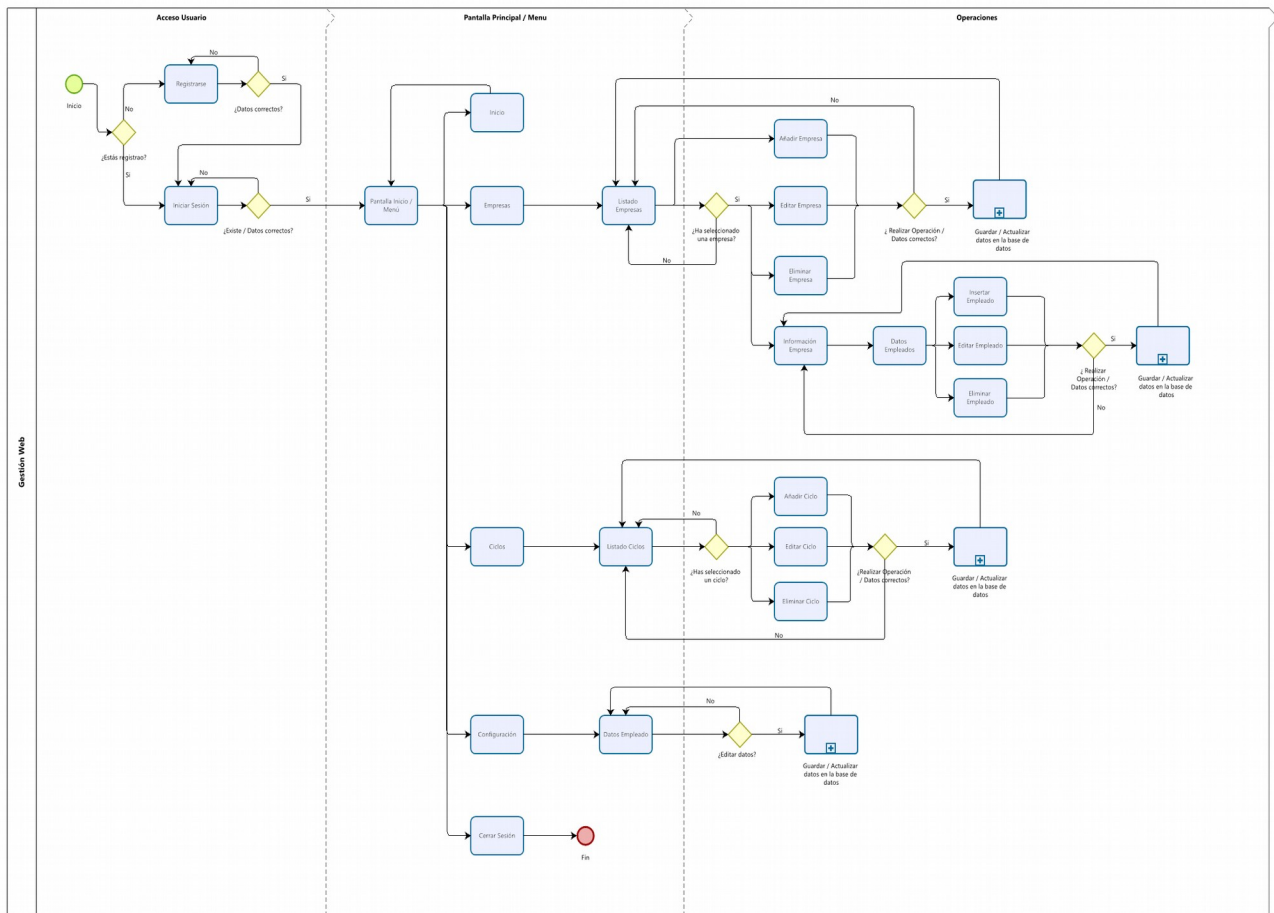
Datos

Datos

INFORMACIÓN:

En la sección de Configuración, además de mostrarnos nuestros datos como usuario tambien podemos modificar nuestros datos.

- **Diseño del Diagrama de Flujo de Datos Web**



- **Diagrama de la Base de Datos BDPracticas**

He utilizado MongoDB y MongoDB Compass para la gestión de la base de datos.

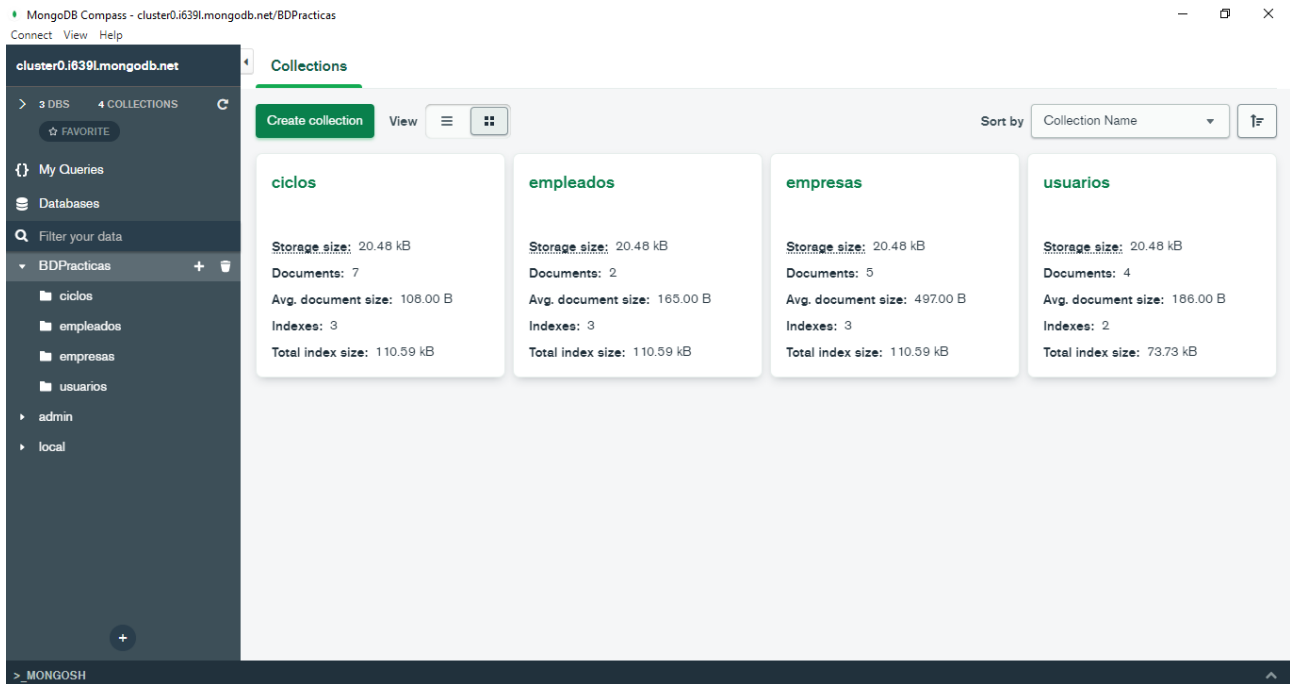
¿Qué es MongoDB y MongoDB Compass?

MongoDB es una base de datos de documentos que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado.

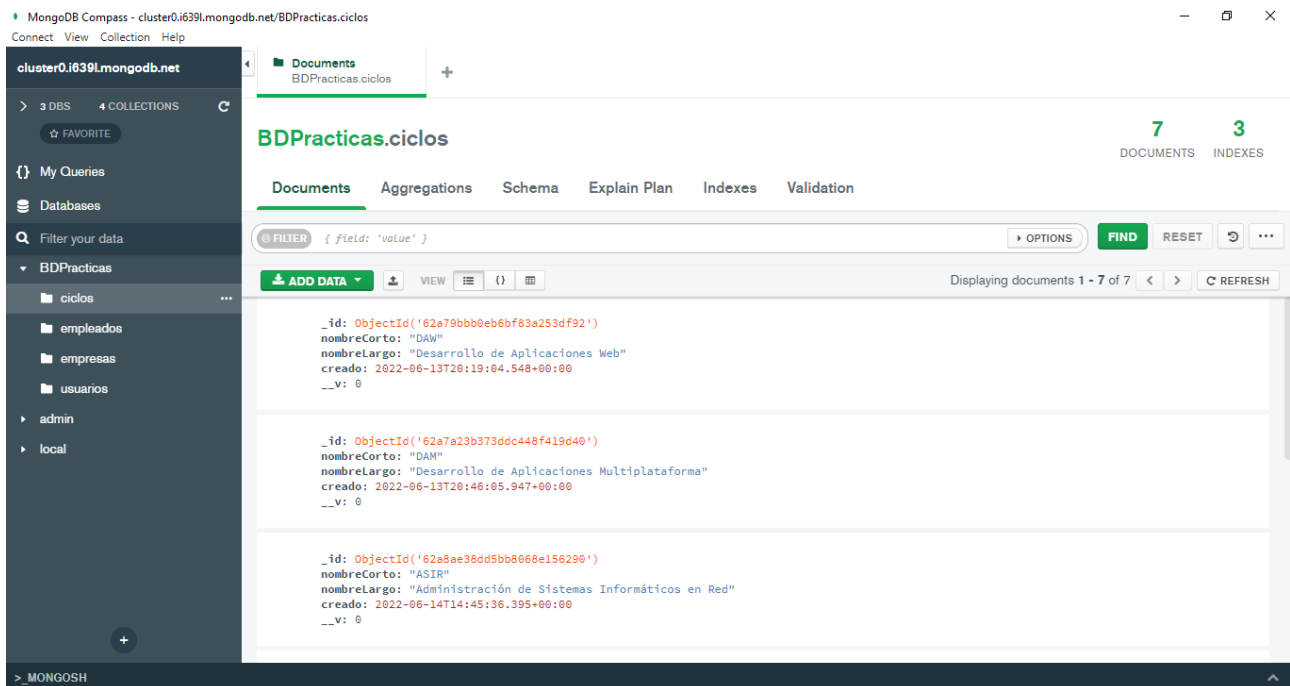
Esta es una base de datos NoSQL orientada a documentos. Se utiliza para almacenar volúmenes masivos de datos. A diferencia de una base de datos relacional SQL tradicional, MongoDB no se basa en tablas y columnas.

Los datos se almacenan como colecciones y documentos.

MongoDB Compass es una poderosa GUI para consultar, agregar y analizar los datos de MongoDB en un entorno visual.



Así es como se ven los datos organizados internamente.

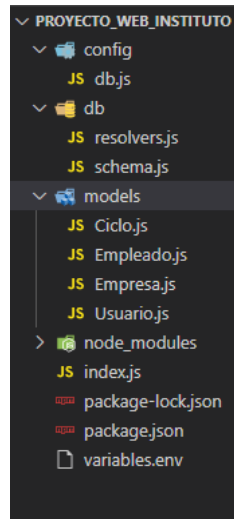


8.- Código fuente documentado.

El proyecto está dividido en dos carpetas, **Proyecto_Web_Instituto** que es la parte del servidor y **webinstituto** que es la parte del cliente

Voy a comenzar a explicar el código fuente de la parte del servidor.

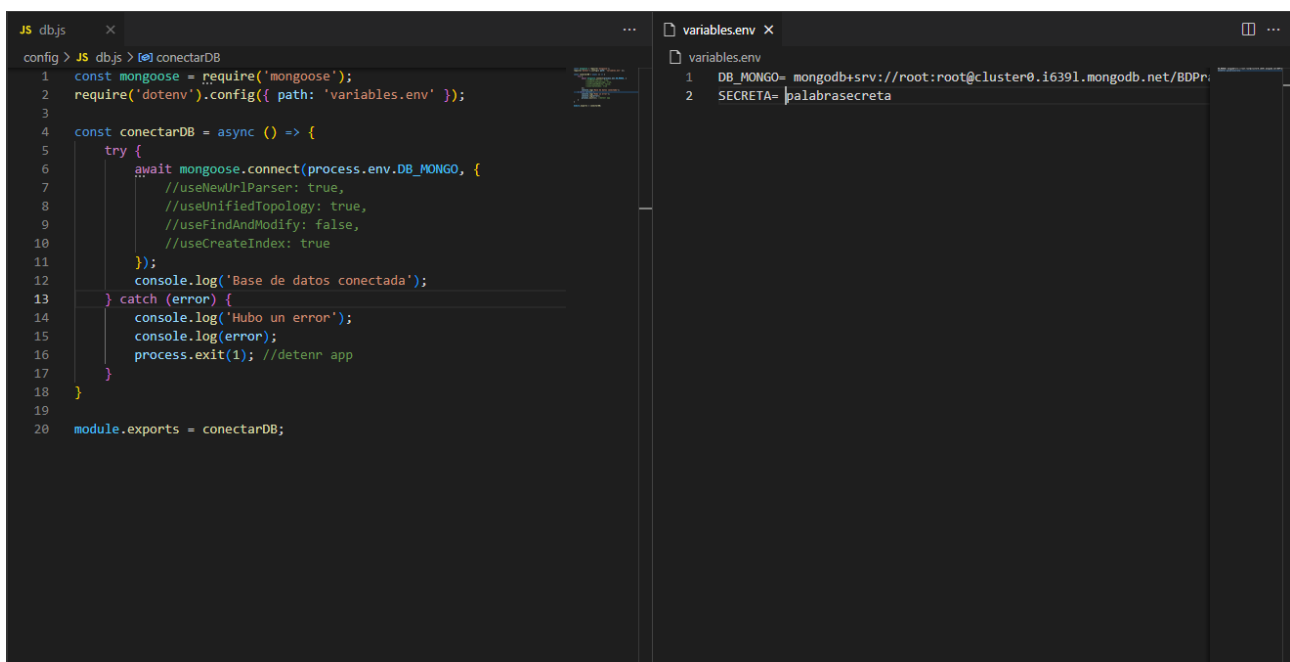
Estructura de carpetas Proyecto_Web_Instituto



1.- Carpeta Config:

1.1.- db.js y variables.env:

En **config/db.js** este es el archivo encargado de realizar la conexión con la base de datos y en **variables.env** almacenamos las variables constantes a las que se le hace llamada para poder conectarnos a la base de datos, se le pasa una **URL** y una **palabra secreta**.



1.2.-Index.js:

Realiza la conexión con Apollo Server y la base de datos (método conectar de db.js). Además creamos el token del usuario que recién se haya logueado. Esta conexión es la que luego se enlazará con Apollo Client.

```
JS index.js  X
JS index.js > ...
6   require('dotenv').config({ path: 'variables.env' });
7
8
9   //Conectar a la Base de Datos
10  conectarDB();
11
12  // servidor
13  const server = new ApolloServer({
14    typeDefs,
15    resolvers,
16    context: ({req}) => {
17      //console.log(req.headers['authorization'])
18
19      //console.log(req.headers)
20
21      const token = req.headers['authorization'] || '';
22      if (token) {
23        //console.log('Holas?');
24        try {
25          const usuario = jwt.verify(token.replace('Bearer ', ''), process.env.SECRETA);
26
27          console.log(usuario);
28
29          return {
30            usuario
31          }
32        } catch (error) {
33          console.log('Hubo un error');
34          console.log(error);
35        }
36      }
37    }
38  });
```

2.- Carpeta db

2.1.- Schema.js:

Cuando realizamos una consulta a un servidor de GraphQL, el entorno de ejecución valida que los campos que solicitamos existan, que la consulta sea válida, que los objetos que solicitamos tengan indicados los sub campos que requerimos, etc.

Para poder hacer esto, es necesario que definamos qué objetos y campos pueden consultarse de nuestro servicio de GraphQL, sin esta definición, ni el cliente ni el entorno de ejecución podrían validar o indicarnos qué podemos y qué no podemos solicitar de un servicio web de GraphQL.

Sistema de tipado en GraphQL (types)

El bloque fundamental de el schema en GraphQL son los tipos. A través del sistema de tipado de GraphQL podemos definir las estructuras de nuestros datos y el tipo de cada propiedad de la que se compone nuestro servicio web.

En general, podemos decir que existen dos elementos importantes en el sistema de tipado, los tipos objetos y los tipos escalares, los primeros definen las estructuras de nuestro servicio web.

```
JS schema.js X
db > JS schema.js > ...
1  const {gql} = require('apollo-server');
2
3  // Schema
4  const typeDefs = gql`
5
6      type Usuario {
7          id: ID
8          nombre: String
9          apellido: String
10         email : String
11         password : String
12         creado: String
13     }
14
15     type Token {
16         token: String
17     }
18
19     type Empresa {
20         id: ID
21         nombre: String
22         cif: String
23         representante: String
24         nif: String
25         telefono: Int
26         direccion: String
27         ciclos: [EmpresaCiclo]
28         creado: String
29     }
30`
```

Las estructuras objeto están compuestas por sub campos que pueden ser de dos tipos, otras estructuras objeto para especificar relaciones entre nuestros datos y los tipos escalares.

Los tipos escalares vienen predefinidos en GraphQL, aunque podemos definir más, y son estos:

Int: Un entero de 32 bits Float: Un valor decimal con punto flotante. value. String: Una secuencia de caracteres Boolean: Un valor que puede ser verdadero o falso ID: Un identificador único de la estructura

Eventualmente, cualquier estructura que definamos en nuestro esquema, tienen que representarse con alguno de estos tipos escalares, en este caso tenemos varios types por lo que explicaré uno de ellos.

type Empresa: Considero que es el más completo, en el podemos encontrar:

- **id:** tipo **ID** (este dato no es necesario que nosotros lo creamos a la hora de hacer inserciones ya que la base de datos nos crea un id por defecto).
- **nombre, cif, representante, nif, direccion y creado:** tipo **String**, son cadenas de caracteres.

- **telefono:** tipo **Int**, solo permitirá enteros. En caso de introducir caracteres nos saltarán errores, lo mismo pasa con los demás campos, con que uno de ellos no sea como en la estructura, nos saltarán errores indicándonoslo.
- **ciclos:** tipo **Array**, este campo almacenará información sobre estos ciclos que están declarados en otro type llamado **EmpresaCiclo**, este type simplemente recoge un id y dos String.

Los types están muy bien controlados por lo que campos de más o de menos, esto nos lo notifica rápido.

```

30
31     type Empleado {
32         id: ID
33         nombre: String
34         apellido: String
35         nif: String
36         telefono: Int
37         email: String
38         empresa: ID
39         creado: String
40     }
41
42     type Ciclo {
43         id: ID
44         nombreCorto: String
45         nombreLargo: String
46         creado: String
47     }
48
49     type EmpresaCiclo {
50         id: ID
51         nombreCorto: String
52         nombreLargo: String
53     }
54
55     input UsuarioInput {
56         nombre: String!
57         apellido: String!
58         email: String!
59         password: String!
60     }
61

```

Los input son los datos que recogemos cuando por ejemplo hacemos un formulario y recogemos esos datos, pues bien, estos datos son validados por estos input, podemos ver que hay **!**, esto significa que el campo que lo contenga, este es obligatorio, por lo que si ese campo está vacío no saldrán mas errores.

```

62     input AutenticarInput {
63         email: String!
64         password: String!
65     }
66
67     input EmpresaInput {
68         nombre: String!
69         cif: String!
70         representante: String!
71         nif: String!
72         telefono: Int!
73         direccion: String!
74         ciclos: [EmpresaCicloInput]
75     }
76
77     input EmpleadoInput {
78         nombre: String!
79         apellido: String!
80         nif: String!
81         telefono: Int!
82         email: String!
83         empresa: ID!
84     }
85
86     input EmpresaCicloInput {
87         id: ID!
88         nombreCorto: String!
89         nombreLargo: String!
90     }
91
92     input CicloInput {
93         nombreCorto: String!

```

Type Query

Es donde definimos las operaciones de consulta que se pueden realizar a nuestro servicio web, por ejemplo, como observamos tenemos varias queries, estas consultas las iremos utilizando con la realización del proyecto.

Pero para poner un poco en contexto, por ejemplo tenemos obtenerUsuario : Usuario, esto lo que hará será que obtenerUsuario estará definido en el resolvers en la parte de Query y este nos devolverá un Usuario. Por como bien dice el nombre de la query, obtenemos el Usuario.

En otra queries podemos ver que pasamos parámetros ID obligatorios que devolvemos un solo resultado o un array. (Podemos orientarnos con los parámetros, al pasar un id sabemos que vamos a buscar un dato con el mismo id que buscamos).

```
type Query {  
  #Usuarios  
  obtenerUsuario : Usuario  
  
  #Empresas  
  obtenerEmpresas : [Empresa]  
  obtenerEmpresa(id: ID!) : Empresa  
  obtenerEmpleadosEmpresa(id: ID!) : [Empleado]  
  obtenerEmpresaCiclo(id: ID!) : [Empresa]  
  
  #Ciclos  
  obtenerCiclos : [Ciclo]  
  obtenerCiclo(id: ID!) : Ciclo  
}
```

type Mutation

Para operaciones que alteren la información de nuestro servicio web, tenemos el tipo Mutation, donde enlistamos precisamente estas operaciones, las de modificaciones, que pueden ser crear nuevos registros, actualizarlos, reordenarlos o eliminarlos.

Fuera de la separación entre operaciones de consulta y operaciones de modificación, los tipos Mutation y Query son muy similares.

```
type Mutation {  
  # Usuarios  
  nuevoUsuario(input: UsuarioInput) : Usuario  
  autenticarUsuario(input: AutenticarInput) : Token  
  actualizarUsuario(id: ID!, input: UsuarioInput) : Usuario  
  
  # Empresas  
  nuevaEmpresa(input: EmpresaInput) : Empresa  
  actualizarEmpresa(id: ID!, input: EmpresaInput) : Empresa  
  eliminarEmpresa(id: ID!) : String  
  
  #Empleados  
  nuevoEmpleado(input: EmpleadoInput) : Empleado  
  actualizarEmpleado(id: ID!, input: EmpleadoInput) : Empleado  
  eliminarEmpleado(id: ID!) : String  
  
  #Ciclos  
  nuevoCiclo(input: CicloInput) : Ciclo  
  actualizarCiclo(id: ID!, input: CicloInput) : Ciclo  
  eliminarCiclo(id: ID!) : String  
}
```

2.2.- Resolvers.js:

Los resolvers es lo que utiliza Apollo Server para saber como procesar las operaciones GraphQL.

Apollo Server necesita saber cómo completar los datos para cada campo en su esquema para que pueda responder a las solicitudes de esos datos. Para lograr esto, utiliza resolvers.

Un resolver es una función que es responsable de completar los datos de un solo campo en su esquema. Puede completar esos datos de cualquier manera que defina, como obtener datos de una base de datos de back-end o una API de terceros.

Los nombre que reciben las queries y los mutation, son los nombres que hemos definido en el schema.

```
// Resolvers
const resolvers = {
  Query: {
    obtenerUsuario: async (_, { }, ctx) => { ...
  },
    obtenerEmpresas: async () => { ...
  },
    obtenerEmpresa: async (_, { id }) => { ...
  },
    obtenerEmpresaCiclo: async (_, { id }) => { ...
  },
    obtenerEmpleadosEmpresa: async (_, { id }) => { ...
  },
    obtenerCiclos: async (_, { }) => { ...
  },
    obtenerCiclo: async (_, { id }) => { ...
  }
},
```

Query:

Muestro una sola query por que al final todas son parecidas.

Es las queries cuando vamos a buscar un dato tenemos que comprobar si existe o no. En caso de que no exista lanzamos una excepción y si existe, devolvemos la empresa que hemos encontrado a través de id obtenido por parámetro.

```
},
obtenerEmpresa: async (_, { id }) => {
  // revisar si la empresa existe o no
  const empresa = await Empresa.findById(id);

  if (!empresa) {
    throw new Error('Empresa no encontrada');
  }

  return empresa;
},
```

Mutation:

```
Mutation: {
  nuevoUsuario: async (_, { input }) => { ...
},
  autenticarUsuario: async (_, { input }) => { ...
},
  actualizarUsuario: async (_, { id, input }) => { ...
},
  nuevaEmpresa: async (_, { input }) => { ...
},
  actualizarEmpresa: async (_, { id, input }) => { ...
},
  eliminarEmpresa: async (_, { id }) => { ...
},
  nuevoEmpleado: async (_, { input }) => { ...
},
  actualizarEmpleado: async (_, { id, input }) => { ...
},
  eliminarEmpleado: async (_, { id }) => { ...
},
  nuevoCiclo: async (_, { input }) => { ...
},
  actualizarCiclo: async (_, { id, input }) => { ...
},
  eliminarCiclo: async (_, { id }) => { ...
}
```

NuevaEmpresa:

En el caso de añadir una nueva empresa, debemos de validar los campos que recibimos por input, que son los que tenemos definidos en el schema, uno de los input importantes a validar, son el nif y el cif.

Para el nif lo que hago es llamar al método validarDNI() donde nos devuelve true o false dependiendo de si el nif es correcto o no, entendemos por correcto es que los números de cif pertenezcan a la letra de este.

En caso de no ser correctos los datos en cualquier caso, siempre lanzaremos errores, que se considera como lanzar excepciones.

Una vez comprobamos los datos añadiremos una empresa con .save(), podemos ver que este método llama a Empresa.save() este Empresa es el modelo que utilizará la base de datos para basarse a la hora de como guardar la estructura de los datos.

```
nuevaEmpresa: async (_, { input }) => {
  const { cif, nif } = input;

  if(!validarDNI(nif)){
    throw new Error('El nif no es correcto');
  }

  //Revisar si el usuario ya está registrado
  const existeEmpresa = await Empresa.findOne({ cif });

  const existeRepresentante = await Empresa.findOne({ nif });

  if (existeEmpresa) {
    throw new Error('La empresa ya está registrada');
  }

  if (existeRepresentante) {
    throw new Error('El nif ya está en uso');
  }

  try {
    const nuevaEmpresa = new Empresa(input);
    // almacenar en la bd
    const resultado = await nuevaEmpresa.save();

    return resultado;
  } catch (error) {
    console.log(error);
  }
}
```

ActualizarEmpresa:

Como hacemos anteriormente, siempre comprobamos los datos obtenidos, en caso de error, se lanzaran mensajes diciéndonoslo.

En actualizarEmpresa también tenemos en cuenta que debemos de validar que un cif que se actualice puede ser que ya exista en la base de datos y o mismo pasaría con el nif, por lo que en todas las actualizaciones comprobamos que los nuevos datos modificados no existan.

Para realizar la actualización utilizamos el método findOneAndUpdate(), le pasamos como parámetro el id de la empresa que queremos actualizar, de segundo parámetro los datos que obtenemos por el input y de tercer parámetro indicamos true, esto es que nos devuelva la empresa ya modificada.

```
actualizarEmpresa: async (_, { id, input }) => {
  // revisar si la empresa existe o no
  let empresa = await Empresa.findById(id);

  const { nif , cif} = input;

  // Buscamos la empresa que tenga el nif/cif que pasamos por input
  let EmpresaNifId = await Empresa.findOne({ nif });
  let EmpresaCifId = await Empresa.findOne({ cif });

  // Un usuario con el nif existe
  if (EmpresaNifId != null) {
    // Comprobamos que no solo seamos nosotros con ese nif y
    // lanzamos excepcion
    if (id != EmpresaNifId.id) {
      throw new Error('Este nif ya está registrado');
    }
  }

  // Un usuario con el cif existe
  if (EmpresaCifId != null) {
    // Comprobamos que no solo seamos nosotros con ese cif y
    // lanzamos excepcion
    if (id != EmpresaCifId.id) {
      throw new Error('Este cif ya está registrado');
    }
  }

  if(!validarDNI(nif)){
    throw new Error('El nif no es correcto');
  }
}
```

```

if (!empresa) {
  throw new Error('Empresa no encontrada');
}

// guardarlo en la base de datos
empresa = await Empresa.findOneAndUpdate({ _id: id }, input, { new: true });

return empresa;

```

Eliminar Empresa:

Como siempre hacemos, buscamos la empresa si existe o no y además borraremos todos los empleados que pertenecen a esa misma empresa, ya que si no existe esa empresa, para que nos sirvan los datos de los empleados.

En este caso como MongoDB no es en cascada, no pasa nada si se borra la empresa y se queda los empleados, pero serían datos innecesarios en nuestra base de datos.

Para eliminar utilizamos el método `findOneAndDelete()`, solo pasamos el parámetro `id`.

```

eliminarEmpresa: async (_, { id }) => {
  // revisar si la empresa existe o no
  let empresa = await Empresa.findById(id);

  if (!empresa) {
    throw new Error('Empresa no encontrada');
  }

  // Buscamos los empleados que pertenecen a esta empresa
  const empleados = await Empleado.find({});
  const empleadosEmpresa = empleados.filter(empleado => empleado.empresa == id);

  // Eliminamos todos los empleados que pertenecen a esa empresa
  if(empleadosEmpresa != null){
    empleadosEmpresa.map( async empleado => {
      console.log(empleado.id)
      await Empleado.findOneAndDelete({ _id: empleado.id });
    })
  }

  //console.log('AQUI');
  //Eliminar
  await Empresa.findOneAndDelete({ _id: id });

  return "Empresa Eliminada";
},

```

3.- Carpeta Models

3.1.- Modelos:

Son un conjunto de funciones para leer/escribir datos de un determinado tipo de GraphQL mediante el uso de varios conectores. Los modelos contienen lógica empresarial adicional, como comprobaciones de permisos, y suelen ser específicos de la aplicación.

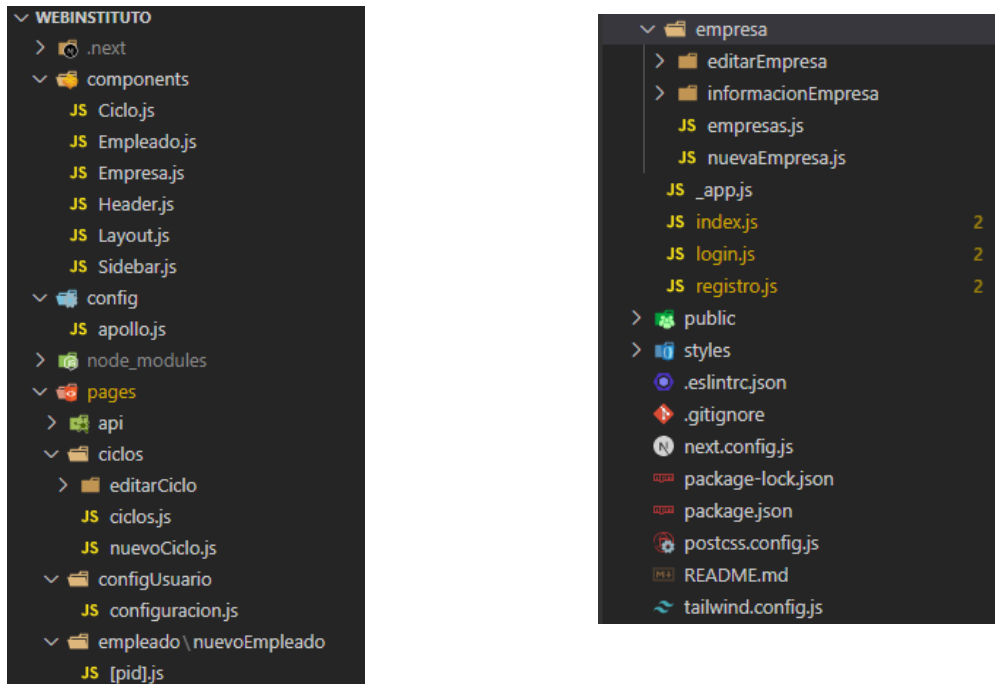
Requerimos mongoose que es Mongoose es una librería para Node.js que nos permite escribir consultas para una base de datos de MongoDB.

```
const mongoose = require('mongoose');

const EmpresasSchema = mongoose.Schema({
  nombre:{
    type: String,
    required: true,
    trim: true
  },
  cif:{
    type: String,
    required: true,
    trim: true,
    unique: true
  },
  representante:{
    type: String,
    required: true,
    trim: true
  },
  nif:{
    type: String,
    required: true,
    trim: true ,
    //unique: true
  },
  telefono:{
    type: Number,
    required: true,
    trim: true
  },
  direccion:{
    type: String,
    required: true,
    trim: true
  },
  ciclos: {
    type: Array
  },
  creado: {
    type: Date,
    default: Date.now()
  }
});

module.exports = mongoose.model('Empresa', EmpresasSchema);
```


Estructura de carpetas webinstituto (Next js)



1.- Carpeta Config

1.1.- Apollo.js:

En este archivo realizamos la conexiones de Apollo Server junto a Apollo Client, ahora nos encontramos en la parte del cliente ya que antes hemos explicado la parte del servidor.

```
JS apollo.js X
config > JS apollo.js > [client] > cache
1 import { ApolloClient, createHttpLink, InMemoryCache } from "@apollo/client";
2 //import fetch from "node-fetch";
3 import { setContext } from 'apollo-link-context';
4
5 const httpLink = createHttpLink({
6   uri: 'http://localhost:4000/',
7   //fetch
8 });
9
10 const authLink = setContext((_, { headers }) => {
11
12   // Leer el storage almacenado
13   const token = localStorage.getItem('token');
14
15   return {
16     headers: {
17       ...headers,
18       authorization: token ? `Bearer ${token}` : ''
19     }
20   }
21 });
22
23 const client = new ApolloClient({
24   connectToDevTools: true,
25   cache: new InMemoryCache(),
26   link: authLink.concat(httpLink)
27 });
28
29 export default client;
```

1.2.- _app.js:

Archivo encargado de mostrarnos la aplicación por pantalla, es la raíz de nuestro proyecto ya que todos nuestros .js se muestran a través de esta.

```
JS _app.js  X
pages > JS _app.js > ...
 1  import '../styles/globals.css'
 2  import { ApolloProvider } from "@apollo/client";
 3  import client from '../config/apollo';
 4
 5  function MyApp({ Component, pageProps }) {
 6    return (
 7      <ApolloProvider client={client}>
 8        <Component {...pageProps} />
 9      </ApolloProvider>
10    )
11  }
12
13  export default MyApp
14
```

2.- Carpeta Pages

2.1.- Login.js y Registro.js

- **Registro.js:** Encargado de realizar los registros de nuevos usuario a la Web.

```
JS registro.js 2  X
pages > JS registro.js > Registro
 1  import React, { useState } from 'react';
 2  import { useRouter, Router } from "next/router";
 3  import Layout from '../components/Layout';
 4  import { useFormik } from "formik";
 5  import * as Yup from 'yup';
 6  import { useMutation, gql } from '@apollo/client';
 7  import Link from 'next/link';
 8  import Image from 'next/image';
 9
10  const REGISTRO = gql`
11    mutation Mutation($input: UsuarioInput) {
12      nuevoUsuario(input: $input) {
13        id
14        nombre
15        apellido
16        email
17      }
18    }
19  `;
20
21
22  export default function Registro() {
23
24    //State para el mensaje
25    const [mensaje, guardarMensaje] = useState(null);
26
27    // Mutation para que se registren nuevos usuarios
28    const [nuevoUsuario] = useMutation(REGISTRO);
29
30    // Routing, nos redirige a la ruta que seleccionemos
31    const router = useRouter();
32
33    // Validación del formulario
```

```

33 // Validación del formulario
34 const formik = useFormik({
35   initialValues: {
36     nombre: '',
37     apellido: '',
38     email: '',
39     password: ''
40   },
41   validationSchema: Yup.object({
42     nombre: Yup.string()
43       .required('El Nombre es obligatorio'),
44     apellido: Yup.string()
45       .required('El Apellido es obligatorio'),
46     email: Yup.string()
47       .email('El email no es válido')
48       .required('El email es obligatorio'),
49     password: Yup.string()
50       .required('La contraseña no puede estar vacía')
51       .min(6, 'La contraseña debe de ser de al menos 6 caracteres')
52   }),
53   onSubmit: async valores => {
54
55     const { nombre, apellido, email, password } = valores;
56
57     try {
58       const { data } = await nuevoUsuario({
59         variables: {
60           input: {
61             nombre,
62             apellido,
63             email,
64             password

```

```

64             password
65           }
66         }
67       });
68       console.log(data);
69
70       // Usuario creado correctamente
71       guardarMensaje(`Se creó correctamente el Usuario: ${data.nuevoUsuario.nombre}`);
72
73       setTimeout(() => {
74         guardarMensaje(null);
75         // Redirigir al usuario para iniciar sesión
76         router.push('/login');
77       }, 3000);
78
79     } catch (error) {
80       guardarMensaje(error.message);
81       setTimeout(() => {
82         guardarMensaje(null);
83       }, 3000);
84     }
85   }
86 }
87 });
88
89 const mostrarMensaje = () => {
90   return (
91     <div className="bg-white py-2 px-3 w-full my-3 max-w-sm text-center mx-auto">
92       <p>{mensaje}</p>
93     </div>
94   )
95 }
96

```

```

    return (
      <>
        <Layout>
          <section className="mb-40 background-radial-gradient overflow-hidden">...
            </section>
          </Layout>
        </>
      )
    )
  }
}

```

Explicación código:

1. Importaciones necesarias de métodos.
2. Mutation para registrar un nuevo usuario
3. Function ()

Hay varias constantes que vamos a ver durante toda la explicación del código, por lo que como son la mayoría iguales, dejaré aquí la explicación general de ellas.

Const [mensaje, guardarMensaje]: Encargado de mostrarnos por pantalla los mensajes de error que obtenemos desde la parte del servidor.

Const [nuevoUsuario] = useMutation(Registro): Como recordamos las Mutation son para insertar, actualizar o eliminar, como en este caso vamos a insertar un nuevo usuario utilizamos mutation, junto a la consulta que hemos declarado arriba.

Const router = useRouter(): Se utiliza para navegar entre rutas de los archivos.

Const formik = useFormik:

- **InitialValues:** inicia los valores de los input, en este caso los inicia a vacío, todos los nombres que aparecen en initialValues son los mismo a los que les realizamos las validaciones.
- **ValidationSchema:** validaciones de los input:
 - **.required()** → el valor es obligatorio
 - **.email()** → valia internamente el patron de un email (expresion regular)
 - **.min()** → minimo de caracteres.
- **onSubmit(valores)** → recoge los valores de los input y se realiza la inserción del nuevo usuario (en este caso).
- **SetTimeout():** método que espera 3 segundos antes de enviarnos al login después del registro.

Método mostrarMensaje(): No muestra el mensaje por pantalla, el mensaje es el que hemos obtenido en la constante mensaje.

Return (): dentro de este return encontramos todo el html, no lo muestro en la capturas porque ocupa mucho espacio.

Voy a mostrar un pedazo del codigo del return para ver como funciona formik en los formularios.

En la imagen de abajo podemos ver que se conecta el formulario con formik para así realizar las validaciones, inicializaciones y obtener los datos.

Cada input debe tener onBlur y onChange además de formik.touched y formik.error, ellos son los encargados de que cada valor este conectado.

```

{mensaje && mostrarMensaje()}
<div className="block rounded-lg shadow-lg px-6 py-12 md:px-12" style={{ bac
  <form onSubmit={formik.handleSubmit}>
    <div className="grid md:grid-cols-2 md:gap-6">
      <div className="mb-6">
        <input className="form-control block w-full px-3 py-1.5 text
          id='nombre'
          type='nombre'
          placeholder="Nombre"
          value={formik.values.nombre}
          onChange={formik.handleChange}
          onBlur={formik.handleBlur}
        />
        {formik.touched.nombre && formik.errors.nombre ? (
          <div className='my-2 bg-red-100 border-1-4 border-r
            <p className='font-bold'>Error</p>
            <p>{formik.errors.nombre}</p>
          </div>
        ) : null}
      </div>
    </div>
  </form>

```

- **Login.js:**

En el login encontramos lo mismo que en registro, la única diferencia es que hacemos uso del token que asignamos al usuario. Para ello utilizamos useMutation(Autenticar_usuario)

El onSubmit recoge email y password.

```

const AUTENTICAR_USUARIO = gql`
  mutation AutenticarUsuario($input: AutenticarInput) {
    autenticarUsuario(input: $input) {
      token
    }
  }
`
;

```

```

try {
  const { data } = await autenticarUsuario({
    variables: {
      input: {
        email,
        password
      }
    }
  });
  console.log(data);
  guardarMensaje('Autenticando...')

  // Guardar el token en localStorage
  setTimeout(() => {
    const { token } = data.autenticarUsuario;
    localStorage.setItem('token', token);
  }, 1000);

  // Redireccionar hacia Inicio
  setTimeout(() => {
    guardarMensaje(null);
    router.push('/');
  }, 5000);
}

```

2.2.- Index.js:

Este archivo simplemente muestra código html, con un par de botones que nos llevan a la opción de empresas y ciclo.

2.3.- Carpeta empresa / empleado / ciclos / configUsuario

Todas estas carpetas contienen archivos parecidos con las mismas funciones pero cambiando los nombres y las consultas a las base de datos, por lo tanto para no hacerlo muy tedioso mostraré el código de empresas que es la mas completa.

2.3.1.- Empresa.js

Mostramos por pantalla todas las empresas registradas en una tabla, además podemos añadir nuevas empresas, editarlas, eliminarlas y nos muestra su información más detallada.

Búsqueda de empresas, nos busca por cualquier dato de la tabla y también tenemos un botón que nos filtra por ciclo.

```
const OBTENER_EMPRESAS = gql`
  query Query {
    obtenerEmpresas {
      id
      nombre
      cif
      representante
      nif
      telefono
      direccion
      ciclos {
        id
        nombreCorto
        nombreLargo
      }
    }
  }
`;

const OBTENER_CICLOS = gql`
  query Query {
    obtenerCiclos {
      id
      nombreCorto
      nombreLargo
    }
  }
`;

const OBTENER_EMPRESAS_CICLO = gql`
  query Query($id: ID!) {
    obtenerEmpresaCiclo(id: $id) {
      nombre
      cif
      representante
      nif
      telefono
      direccion
    }
  }
`;
```

```

export default function Empresas() {

  const router = useRouter();

  const [query, setQuery] = useState('');

  const [showOptions, setShowOptions] = useState(null);

  const [refresh, setRefresh] = useState(true);

  const handleClick = () => {
    setShowOptions(!showOptions);
  }

  const [showCiclos, setShowCiclos] = useState(null);

  const { data: dataCiclos, loading: loadingCiclos, error: errorCiclos } = useQuery(OBTENER_CICLOS);

  const { data: dataEmpresasCiclo, loading: loadingEmpresasCiclo, error: errorEmpresasCiclo } = useQuery(OBTENER_EMPRESAS_CICLO,
    {
      variables: {
        id: showCiclos
      }
    });

  // Consulta de Apollo
  const { data, loading, error } = useQuery(OBTENER_EMPRESAS);

  if (loading || loadingCiclos) return 'Cargando...';

  if (loadingCiclos) return null;

  const { obtenerCiclos } = dataCiclos;

  if (!data.obtenerEmpresas) {
    return router.push('../login');
  }

  const keys = ["nombre", "cif", "representante", "direccion"];

  const search = (data) => { ...
}

```

```

const search = (data) => {
  return data.filter((empresa) => keys.some((key) => empresa[key].toLowerCase().includes(query)))
  /*return data.filter(empresa => empresa.nombre.toLowerCase().includes(query) || empresa.cif.toLowerCase().includes(query))*/
}

```

Esta misma clase utiliza componentes, los componentes de React son elementos autónomos que puede reutilizar en una página. Al crear pequeñas piezas de código centradas, las puede mover y reutilizar a medida que su aplicación se amplía. La clave es que son autónomas y centradas, lo que le permite separar el código en piezas lógicas.

En este caso nuestro componente se llama Empresa, pero al igual tenemos componentes de Ciclo, Empleado, etc.

Carpeta Component /Empresa

Este componente nos muestra por pantalla todas las empresas, en vez de tener muchos `<tr>` declarados, recorreremos con **map()** y llamamos al componente, donde simplemente recoge la estructura de uno, pero al recorrer, todas las empresas toman la misma estructura. A la hora de llamar al componente le pasamos un **key= id** ya que tienen que ser únicos y **empleado={empleado}**, para después mostrar todos los valores de ese empleado por pantalla.

Por cada fila que pertenece a una empresa tenemos los botones de editar, eliminar e información de esa empresa.

```
8   const ELIMINAR_EMPLEADO = gql`
9     mutation Mutation($id: ID!) {
10       eliminarEmpleado(id: $id)
11     }
12   `;
13   const OBTENER_EMPLEADOS = gql`
14     query Query($id: ID!) {
15       obtenerEmpleadosEmpresa(id: $id) {
16         id
17         nombre
18         apellido
19         nif
20         telefono
21         email
22       }
23     }
24   `;
25
26   const ACTUALIZAR_EMPLEADO = gql`
27     mutation ActualizarEmpleado($id: ID!, $input: EmpleadoInput) {
28       actualizarEmpleado(id: $id, input: $input) {
29         nombre
30         apellido
31         nif
32         telefono
33         email
34         empresa
35       }
36     }
37   `;
38
39   export default function Empleado({ empleado }) {
40
41     // Mensaje de alerta
42     const [mensaje, guardarMensaje] = useState(null);
43
44     const { nombre, apellido, nif, telefono, email, id, empresa } = empleado;
45
46     const [mostrarComponente, setMostrarComponente] = useState(false);
47
48     const [operativo, setOperativo] = useState(true);
49
50     // Consultar para obtener el empleados
51     const { data, loading, error } = useQuery(OBTENER_EMPLEADOS, {
52       variables: {
53         id: empresa
54       }
55     });
```

```
100   if (loading) return 'Cargando...';
101
102   // Mutation para eliminar empleado
103   const confirmarEliminarEmpleado = () => {
104     Swal.fire({
105       title: '¿Deseas eliminar este empleado?',
106       text: '¡Esta acción no se puede deshacer!',
107       icon: 'warning',
108       showCancelButton: true,
109       confirmButtonColor: '#3085d6',
110       cancelButtonColor: '#d33',
111       confirmButtonText: '¡Sí, Eliminar!',
112       cancelButtonText: 'No, Cancelar'
113     }).then(async (result) => {
114       if (result.isConfirmed) {
115         //console.log('Eliminando...',id);
116         try {
117           // Eliminar por ID
118           const { data } = await eliminarEmpleado({
119             variables: {
120               id
121             }
122           })
123           //console.log(data);
124
125           Swal.fire(
126             'Eliminado',
127             data.eliminarEmpleado, // Al devolvernos un mensjae, los mostramos en el alert
128             'success'
129           )
130         } catch (error) {
131           console.log(error);
132         }
133       }
134     })
135   }
136 }
137
```



```

138 // Modifica el empleado de la BD
139 const actualizarInfoEmpleado = async valores => {
140   const { nombre, apellido, nif, telefono, email } = valores;
141   console.log(valores);
142   try {
143     const { data } = await actualizarEmpleado({
144       variables: {
145         id: id,
146         input: {
147           nombre,
148           apellido,
149           nif,
150           telefono,
151           email,
152           empresa: empresa
153         }
154       }
155     });
156     //console.log(data);
157
158     // Mostrar alerta
159     Swal.fire(
160       'Actualizado',
161       'Información actualizada correctamente',
162       'success'
163     );
164
165     setOperativo(!operativo)
166     setMostrarComponente(!mostrarComponente)
167
168   } catch (error) {
169     guardarMensaje(error.message);
170     setTimeout(() => {
171       guardarMensaje(null);
172     }, 3000)
173   }
174 }
175
176 const mostrarMensaje = () => {
177   Swal.fire({
178     icon: 'error',
179     title: 'Oops...',
180     text: mensaje,
181     footer: '<a href="">Vuelve a comprobar los datos introducidos</a>'
182   })
183 }
184
185

```

2.3.2.- NuevaEmpresa.js y EditarEmpresa:

Ambos archivos son completamente iguales, simplemente uno añade y otro actualiza, en el de añadir la inicialización de valores es vacía y en el de editar, la inicialización de valores es con los datos de la empresa seleccionada

Los datos introducidos por input son validados por formik, que ya explicamos anteriormente. Realizamos un Mutation para añadir/editar la empresa. En caso de datos incorrectos nos saltan excepciones.

El eliminar empresa lo hacemos desde Empresas.js, en la misma fila donde se encuentran los valores de esta, nos aparece el botón de eliminar y lo podemos eliminar desde ahí mismo.

```

JS nuevaEmpresajs X
pages > empresa > JS nuevaEmpresajs > NuevaEmpresa
9
10 const NUEVA_EMPRESA = gql`
11
12   mutation NuevaEmpresa($input: EmpresaInput) {
13     nuevaEmpresa(input: $input) {
14       id
15       nombre
16       cif
17       representante
18       nif
19       telefono
20       direccion
21     }
22   }
23 `;
24
25
26 const OBTENER_EMPRESAS = gql`
27   query Query {
28     obtenerEmpresas {
29       id
30       nombre
31       cif
32       representante
33       nif
34       telefono
35       direccion
36     }
37   }
38 `;
39
40 const OBTENER_CICLOS = gql`
41   query Query {
42     obtenerCiclos {
43       id
44       nombreCorto
45       nombreLargo
46     }
47   }
48 `;
49

```

```

export default function NuevaEmpresa() {

  const router = useRouter();

  // Mensaje de alerta
  const [mensaje, guardarMensaje] = useState(null);

  const { data, loading, error } = useQuery(OBTENER_CICLOS);

  const [selectedOptions, setSelectedOptions] = useState([]);

  const handleSelect = () => {
    console.log(selectedOptions);
  };

  const [nuevaEmpresa] = useMutation(NUEVA_EMPRESA, {
    update(cache, { data: { nuevaEmpresa } }) {
      // obtener el objeto de cache que deseamos actualizar
      const { obtenerEmpresas } = cache.readQuery({ query: OBTENER_EMPRESAS });

      // Reescribimos el cache (el caché nunca se debe modificar)
      cache.writeQuery({
        query: OBTENER_EMPRESAS,
        data: {
          obtenerEmpresas: [...obtenerEmpresas, nuevaEmpresa]
        }
      });
    }
  });

  const formik = useFormik({
    initialValues: {
      nombre: '',
      cif: '',
      representante: '',
      nif: '',
      telefono: '',
      direccion: ''
    },
    validationSchema: Yup.object({
      nombre: Yup.string()
        .required('El nombre de la empresa es obligatorio'),
      cif: Yup.string()
        .matches(/^[ABCDEFGHJKLMNPQRSTUVWXYZ]{7}$/, 'El formato del cif es incorrecto')
        .required('El cif es obligatorio'),
      representante: Yup.string()
        .required('El nombre del representante es obligatorio')
    })
  });
}

```

```

93     .matches(/^[A-Z]{1}[a-z]{1}[0-9]{1}$/, 'El formato del cif es incorrecto')
94     .required('El cif es obligatorio'),
95     representante: Yup.string()
96     .required('El nombre del representante es obligatorio'),
97     nif: Yup.string()
98     .matches(/^[a-zA-Z]{8}$/, 'El formato del dni es incorrecto')
99     .required('El nif es obligatorio'),
100    telefono: Yup.string()
101    .matches(/^[0-9]{9}$/, 'El formato del teléfono es incorrecto')
102    .required('El teléfono es obligatorio'),
103    direccion: Yup.string()
104    .required('La dirección es obligatoria'),
105  },
106  onSubmit: async valores => {
107    console.log(valores);
108
109    let datos = recogerCiclos(selectedOptions);
110
111    if (datos == 0) {
112      guardarMensaje("Debes de seleccionar al menos un ciclo");
113      setTimeout(() => {
114        guardarMensaje(null);
115      }, 3000);
116    } else {
117
118      const { nombre, cif, representante, nif, telefono, direccion } = valores;
119      try {
120        const { data } = await nuevaEmpresa({
121          variables: {
122            input: {
123              nombre,
124              cif,
125              representante,
126              nif,
127              telefono,
128              direccion,
129              ciclos: datos
130            }
131          }
132        });
133
134        router.push('/empresa/empresas'); // Redirreccionar hacia empresas
135      } catch (error) {
136        guardarMensaje(error.message);
137
138        setTimeout(() => {
139          guardarMensaje(null);
140        }, 3000);
141        //console.log(error);

```

```

if (loading) return null;

const { obtenerCiclos } = data;

const recogerCiclos = (opciones) => {
  let array = [];
  opciones.map(a => {
    obtenerCiclos.filter(ciclo => ciclo.nombreCorto === a.nombreCorto ? (
      array.push({ id: ciclo.id, nombreCorto: ciclo.nombreCorto, nombreLargo: ciclo.nombreLargo })
    ) : ());
    console.log(a)
  })

  return array;
}

const mostrarMensaje = () => { ...
}

return (
  <Layout>
    {mensaje && mostrarMensaje()}
  </Layout>
);

```

2.3.3.- InformacionEmpresa.js:

Nos muestra por pantalla todos los datos de la empresa, junto a todos sus empleados, estos empleados se muestran como empresas.js mediante un componente Empleado, por lo tanto nos aparecerán varios empleados o no en una misma empresa, estos empleados se pueden añadir, editar y eliminar.

Ademas de esta información, nos aparecen los ciclos relacionados con esta empresa. El archivo se llama **[pid].js**, **Next js** trabaja con los fichero de esta manera, para interpretar que se está pasando un id por la URL.

```
const OBTENER_EMPRESA = gql`
query ObtenerEmpresa($id: ID!) {
  obtenerEmpresa(id: $id) {
    id
    nombre
    cif
    representante
    nif
    telefono
    direccion
    ciclos {
      id
      nombreCorto
      nombreLargo
    }
  }
}
`;

const OBTENER_EMPLEADOS_EMPRESA = gql`
query Query($id: ID!) {
  obtenerEmpleadosEmpresa(id: $id) {
    id
    nombre
    apellido
    nif
    telefono
    email
    empresa
  }
}
`;

export default function InformacionEmpresa() {

  // obtener el ID actual
  const router = useRouter();
  const { pid } = router.query

  // Consultar para obtener la empresa
  const { data: dataEmpresa, loading: loadingEmpresa, error: errorEmpresa } = useQuery(OBTENER_EMPRESA, {
    variables: {
      id: pid
    }
  })
}
```

```

// Consultar para obtener los empleados de la empresa
const { data: dataEmpleados, loading: loadingEmpleados, error: errorEmpleados } = useQuery(OBTENER_EMPLEADOS_EMPRESA, {
  variables: {
    id: pid
  }
});

if (loadingEmpresa || loadingEmpleados || !pid) return 'Cargando...';

const { obtenerEmpresa } = dataEmpresa;

const { nombre, cif, direccion, representante, nif, telefono } = obtenerEmpresa;

const { obtenerEmpleadosEmpresa } = dataEmpleados;

return (
  <>
    <Layout> ...
  </Layout>
</>
);
}

```

4.- Carpeta component

A parte de los componentes nombrados anteriormente, también tenemos:

4.1.- Layout:

El layout es el encargado de recoger el Header, Sidebar y los ficheros.js de pages y mostrarlos todos cuadrados en la pantalla.

```

JS Layout.js X
components > JS Layout.js > Layout
1 import React from 'react';
2 import Head from 'next/head';
3 import Sidebar from '../components/Sidebar';
4 import Header from '../components/Header';
5 import { useRouter } from 'next/router';
6 import Script from 'next/script';
7
8 const Layout = ({ children }) => {
9
10   // Hook de routing
11   const router = useRouter();
12
13   return (
14     <>
15       <Head>
16         <title>Gestión Prácticas-Empresas</title>
17       </Head>
18       {router.pathname === '/login' || router.pathname === '/registro' ? (
19         <div className='flex flex-col justify-center'>
20           <div>
21             {children}
22           </div>
23         </div>
24       ) : (
25         <div className='min-h-screen'>
26           <svg ...
27         </svg>
28
29         <div className='sm:flex min-h-screen'>
30           <Sidebar />
31           <main className='sm:w-4/5 md:w-4/5 lg:w-4/5 xl:w-4/5 sm:min-h-screen p-5' > { /* sm:w-2/3 xl:w-4/5 sm:min-h-screen p-5*/}
32             <Header />
33             {children}
34           </main>
35         </div>
36       </div>
37     </>
38   );
39 }
40
41 export default Layout;

```

4.2.- Sidebar:

Menú de nuestra aplicación, nos permite ir de una ruta a otra.

```
<li className={router.pathname === "/empresa/empresas" || router.pathname === "/empresa/nuevaEmpresa" || router.pathname === "/empresa/editarEmpresa/[pid]" || router.pathname === "/empresa/eliminarEmpresa/[pid]" ? "active" : ""}>
  <div className="flex items-center">
    <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor" strokeWidth="2">
      <path strokeLinecap="round" strokeLinejoin="round" d="M19 21V5a2 2 0 0-2 2H7a2 2 0 0-2 2v16m14 0h2m-2 0h-5m-9 0h3m2 0h5M9 7h1m-1 4h1m4-4h1m-1 4h1m-5 10v-5a1 1 0 0-1 1h1" />
    </svg>
    <Link href="/empresa/empresas">
      <a className="block pl-2">Empresas</a>
    </Link>
  </div>
</li>
```

4.3.- Header:

Cabecera de la web, se muestra el nombre de la sección en la que nos encontramos y el botón de cerrar sesión.

```
JS Header.js
components > JS Header.js > Header
1  import React from 'react';
2  import { useQuery, gql, ApolloConsumer } from '@apollo/client';
3  import { useRouter } from 'next/router';
4  //import { getSession } from 'next-auth/react';
5
6  const OBTENER_USUARIO = gql`
7    query ObtenerUsuario {
8      obtenerUsuario {
9        id
10       nombre
11       apellido
12     }
13   }
14 `;
15
16  export default function Header() {
17
18    const router = useRouter();
19
20    //query de apollo
21    const { data, loading, error } = useQuery(OBTENER_USUARIO);
22
23    //console.log(data);
24    //console.log(loading);
25    //console.log(error);
26
27    // Proteger que no accedamos a data antes de tener resultados
28    if (loading) return 'Cargando...';
29
30    // Si no hay información
31    if (!data.obtenerUsuario) {
32      router.push('/login');
33    }
34
35    //const { nombre, apellido } = data.obtenerUsuario;
36
37    const cerrarSesion = (usuario) => {
38      localStorage.removeItem('token');
39      usuario.resetStore();
40      router.push('/login');
41    }
42  }
```



```

43 const mensajePagina = (ruta) => {
44
45   if (data.obtenerUsuario) {
46
47     switch (ruta) {
48       case "/":
49         return '¡Hola ' + data.obtenerUsuario.nombre + ' ' + data.obtenerUsuario.apellido + '!';
50         break;
51       case "/empresa/empresas":
52         return 'Empresas';
53         break;
54       case "/empresa/nuevaEmpresa":
55         return 'Nueva Empresa';
56         break;
57       case "/empresa/editarEmpresa/[pid]":
58         return 'Editar Empresa';
59         break;
60       case "/empresa/informacionEmpresa/[pid]":
61         return 'Información Empresa';
62         break;
63       case "/empleado/nuevoEmpleado/[pid]":
64         return 'Nuevo Empleado';
65         break;
66       case "/ciclos/ciclos":
67         return 'Ciclos';
68         break;
69       case "/ciclos/nuevoCiclo":
70         return 'Nuevo Ciclo';
71         break;
72       case "/ciclos/editarCiclo/[pid]":
73         return 'Editar Ciclo';
74         break;
75       case "/configUsuario/configuracion":
76         return 'Información Usuario';
77         break;
78
79       default:
80         break;
81     }
82   }
83
84 }
85
86 //console.log(router.pathname);
87
88 return (

```

```

return (
  <>
  <div className='flex justify-between mb-6'>
    <p className="text-3xl md:text-white lg:text-white xl:text-white font-light mr-2">{mensajePagina(router.pathname)}</p>
    <ApolloConsumer>
      {usuario => (
        <div className="flex items-center lg:ml-auto">
          <button type="button" className="inline-block sm:px-4 px-6 py-2.5 bg-white text-blue-600 font-medium text-xs leading-tight uppercase rounded shadow-md
            data-mdb-ripple="true" data-mdb-ripple-color="light"
            onClick={() => cerrarSesion(usuario)}>Cerrar Sesión</button>
        </div>
      )}
    </ApolloConsumer>
  </div>
  </>
);

```

INFORMACIÓN:

Para las capturas he mostrado solamente una sección de ficheros .js, ya que todas tienen la misma estructura de código.

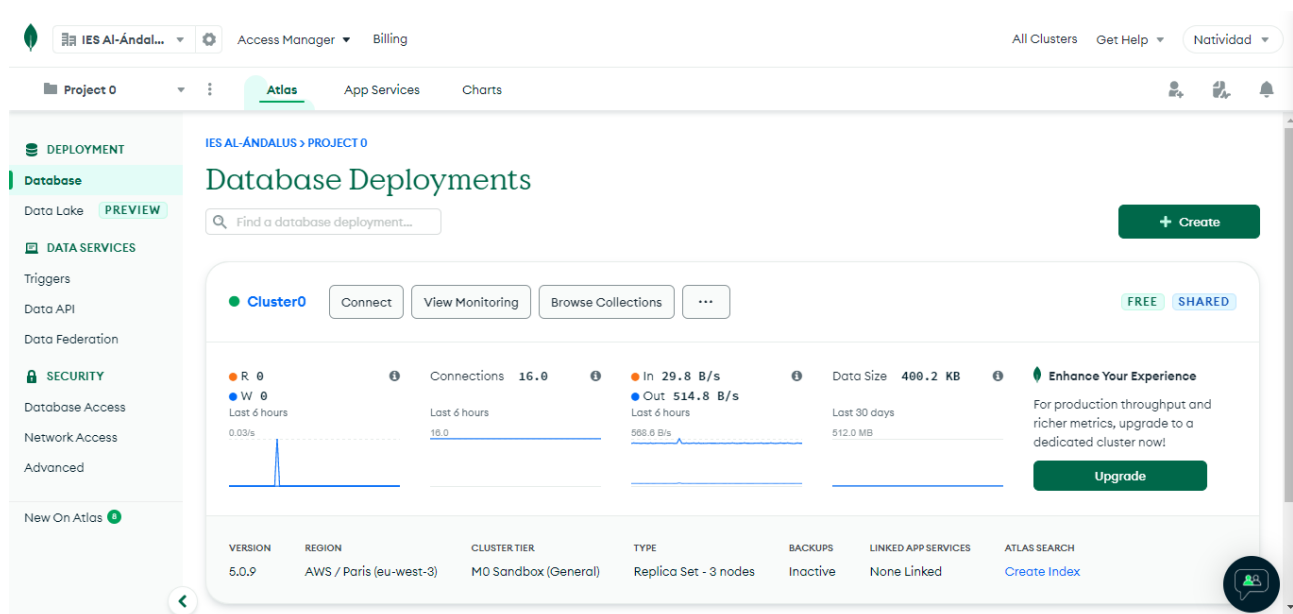
9.- Manual de configuración y funcionamiento de la aplicación.

- Manual de configuración de la Web:
 - Software Windows o Ubuntu
 - Estar registrado en MongoDB Atlas y hacer la instalación de MongoDB Compass para la base de datos.

Instalación de MongoDB Compass

Vamos a la página oficial de MongoDB Atlas y nos registramos

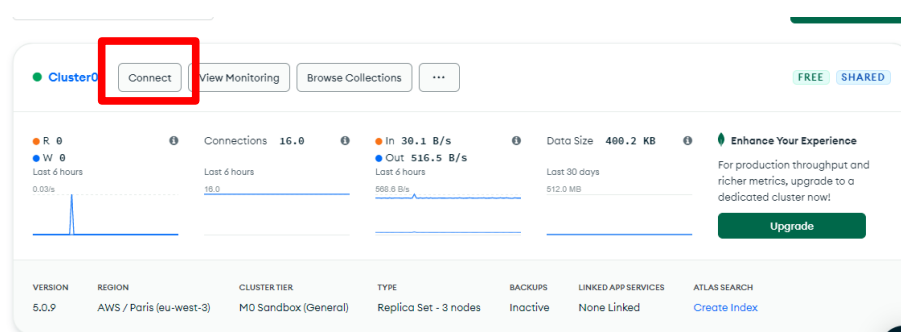
Una vez registrados, debe aparecer algo como esto. (En mi caso ya tengo una base de datos creada por eso me aparece Cluster0)



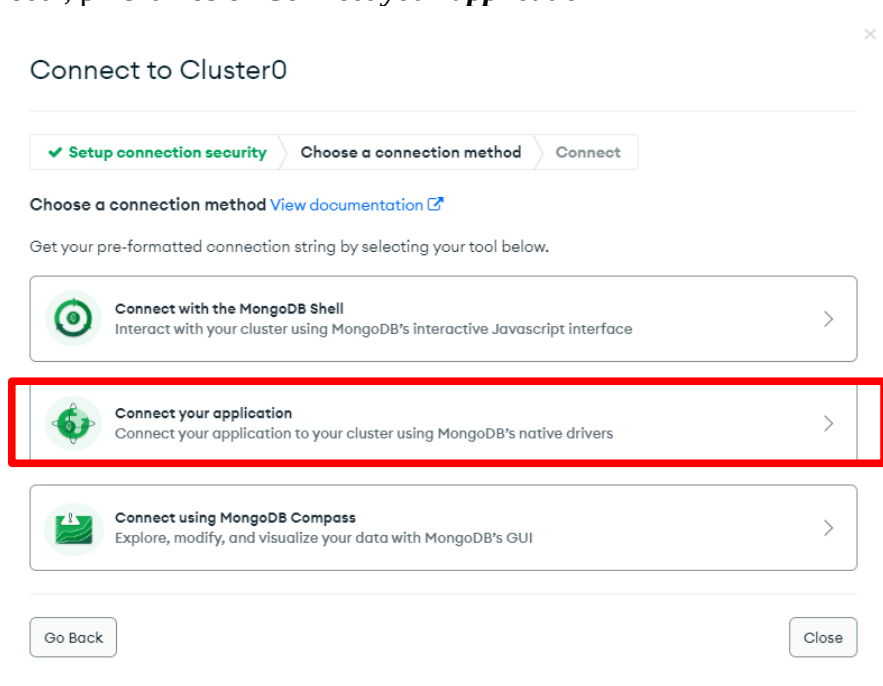
Creamos un cluster pinchando en create, si no aparece create a la derecha, debe de aparecer en el centro, debido a que no se encuentra ningún cluster creado.

Cuando vayamos a crear el cluster, damos todo a siguiente con los valores de por defecto. La web tardará un poco en la creación del cluster .

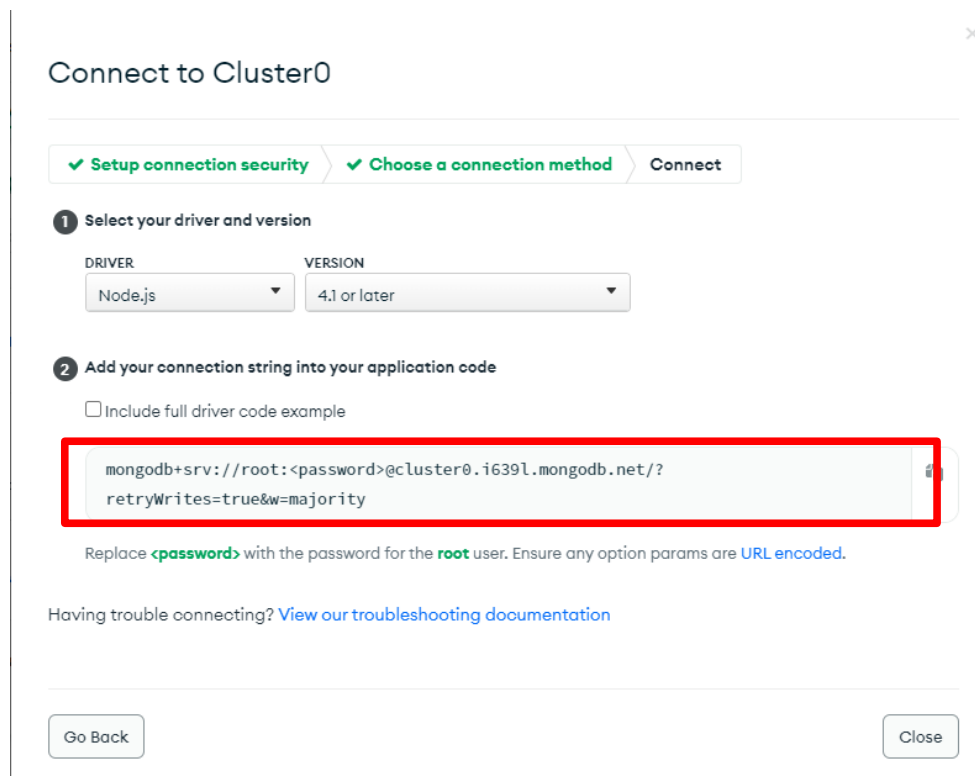
Nuevamente, pinchamos en **Connect**



Aparecerá un modal, pinchamos en **Connect your application**

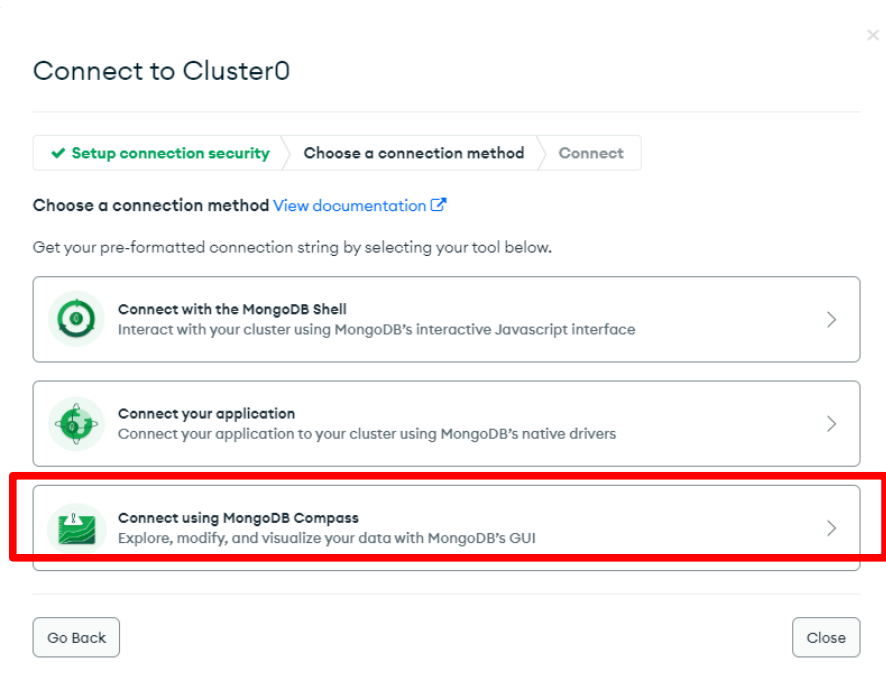


Nos dará una url con nuestro root y la contraseña (si no aparece password es necesario tener una) esta url es la que pondremos en nuestro servidor para conectarnos con la base de datos.

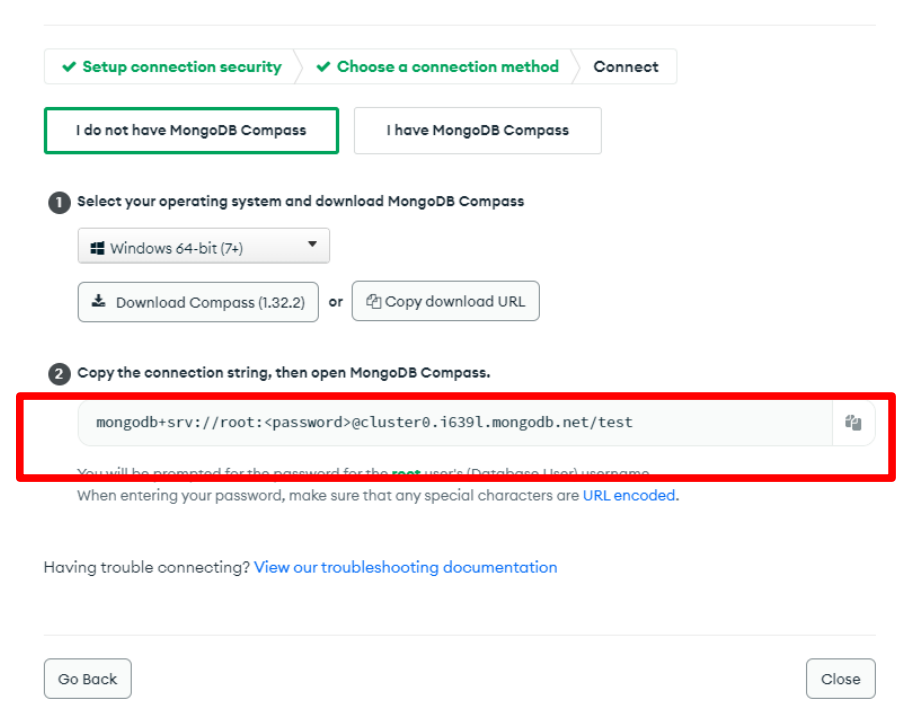


Inciso, tenemos que descargar también MongoDB Compass y seguir el instalador sin complicaciones.

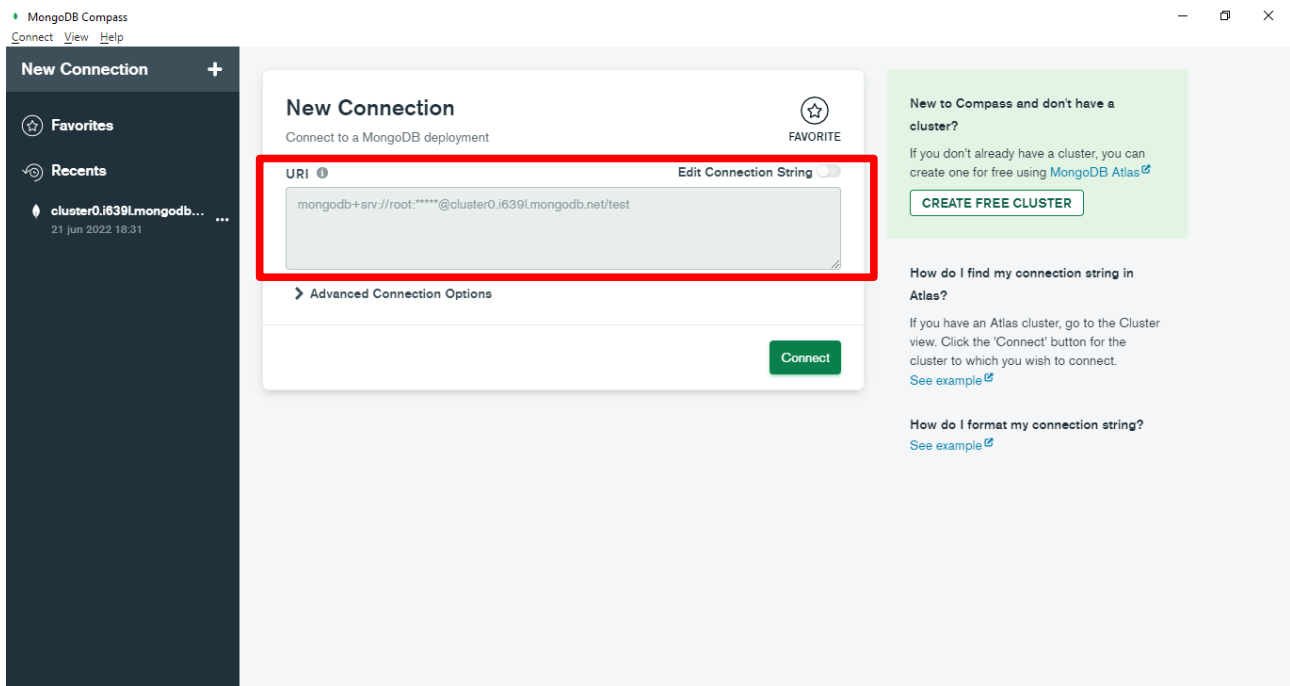
Siguiendo con la instalación de MongoDB Atlas, pinchamos en **Connect using MongoDB Compass**



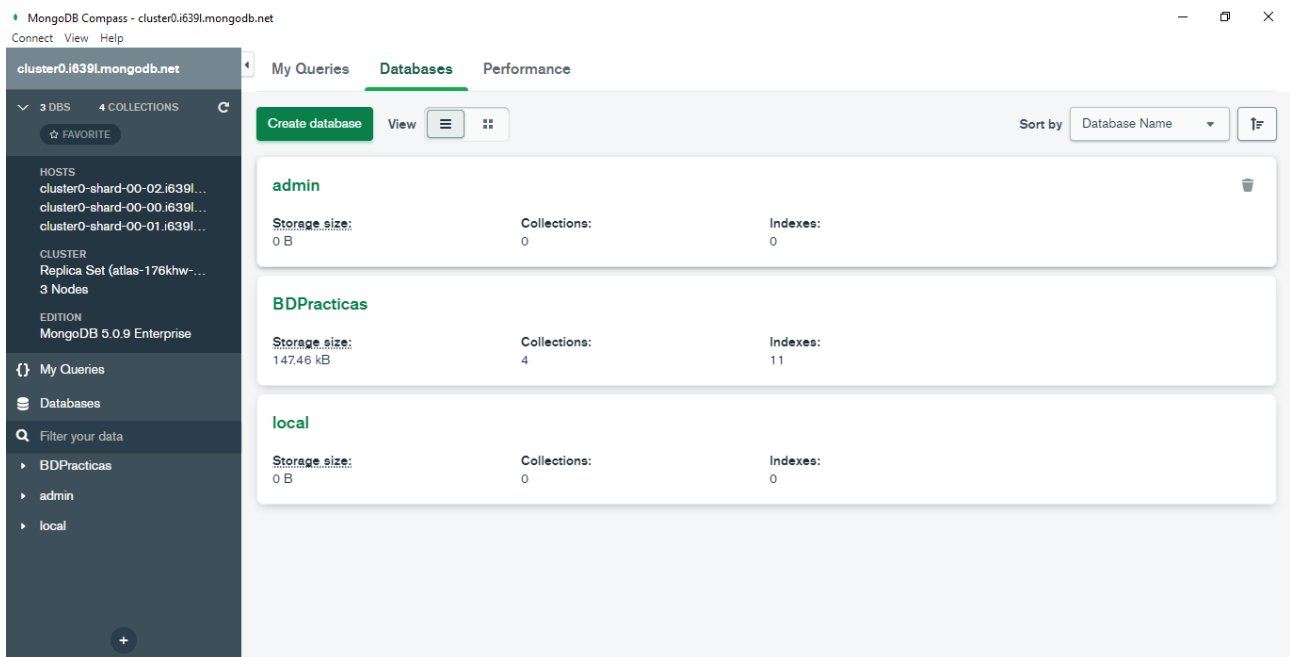
Copiamos esta url y nos dirigimos a MongoDB Compass



Pegamos la url aquí y pinchamos en connect



Nos aparecerá esta pantalla, y pinchamos en crear una base de datos



Ponemos el nombre de la base de datos y de la colección. (Entendemos como colección como si fuera una tabla), sabemos que en MongoDB no hay tablas pero para explicarlo de alguna manera.

Create Database

Database Name

Collection Name

> Advanced Collection Options (e.g. Time-Series, Capped, Clustered collections)

i Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

Cancel Create Database

El proyecto tiene instalado mongoose, que es el encargado de conectar el proyecto con la base de datos de MongoDB y la que nos permitirá consultar, insertar y editar.

Ahora yendo al nuestro proyecto, copiamos la URL que indicamos en **Connect your application** y la pegamos en **variables.env**

```
variables.env X
variables.env
1 DB_MONGO= mongodb+srv://root:root@cluster0.i6391.mongodb.net/BDPracticas
2 SECRETA= palabrasecreta
```

Finalización de la instalación.

NOTA:

Si nos aportan una base de datos, en vez de nosotros crearla en MongoDB Compass, importamos el JSON o CSV que nos han dado, pero recordar que para la hora de la conexión hay que seguir los pasos indicados.

10.- Manual de usuario.

Inicio de Sesión o Registrarse

Información

Si aún no tienes cuenta podrás registrarte.

En cambio si la tienes, tendrás que iniciar sesión

Email y Contraseña

Campos necesarios para poder acceder

Iniciar Sesión

Nos dará acceso a la Web

¿Olvidaste la contraseña?

Podrás cambiar la contraseña por una nueva

La pantalla de registro es muy parecida al login, tienes más campos para introducir. He de decir que en todas las zonas de la web donde aparezcan formularios, todos estos están validados, pues si un dato es incorrecto, la web nos avisará para que lo corrijamos.

Registrarse

Datos a Rellenar

Se tendrá que rellenar todos los campos, pues si no es así, no podrá registrarse correctamente y tener acceso a la web

Registrarse

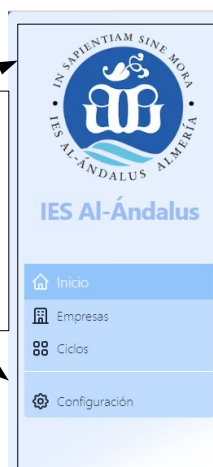
Nos dará acceso a la Web

Esto sería lo primero que encontramos al iniciar sesión.

Pantalla principal / Inicio

Menú

Menú para navegar por la web



¡Hola Nati Fernández!

CERRAR SESIÓN

Cerrar Sesión

Botón al que Siempre tenemos acceso.

Cuando queramos salir de la webs simplemente pinchamos en el.

Mensaje de bienvenida

Imagen

Gestión Prácticas Formación Profesional

EMPRESAS

CICLOS



Refrescar

Al filtrar por ciclos no aparecerán todas las empresas, por lo tanto, refrescamos para que aparezcan todas

En empresas mostramos todas las empresas registradas en la base de datos.

Empresas

Añadir

Añadir una nueva empresa



Empresas

NUEVA EMPRESA

Ciclos

Buscador...

Refrescar

| Nombre | Cif | Representante | Teléfono | Dirección | Acciones |
|------------|-----------|---------------|-----------|----------------|---|
| MarTec | W77775666 | Sofia | 123456789 | Calle Desierto |    |
| Globomatic | C83375666 | juan | 123456789 | calle sol |    |
| ValleTech | B83375575 | juan | 123456789 | calle sol |    |
| InfoAlm | P55375575 | pepe | 123456789 | calle amargura |    |
| Empresita | G83375579 | Fer | 123654789 | Calle |    |

Buscador

Filtrar por cualquier campo que vemos en la tabla, además en el botón izquierdo, podemos filtrar por ciclos

Acciones

Permite Editar, Eliminar y mostrar información de la fila elegida. Si elegimos Editar, redirigirá a un formulario con los datos rellenos de la fila, si elegimos Mostrar, nos muestra una ficha de esta fila y si elegimos Eliminar se abrirá un cuadro de diálogo, para confirmar.

Acciones (Añadir/Editar) Empresa

El formulario añadir/regar y de editar/modificar es completamente igual, excepto que el de editar ya nos autocompleta los campos, podemos cambiarlos si es lo que se quiere. (Se podrá añadir o editar si los campos introducidos son correctos)

Añadir una nueva empresa

Nueva Empresa

Datos Empresa

Nombre Cif Dirección

Datos Representante

Nombre Nif Teléfono

Ciclos

Selecciona algun ciclo

REGISTRAR EMPRESA

Ciclos

Podemos elegir los ciclos, en editar nos muestra lo que tenemos hasta ahora y también se pueden editar marcando el checkbox

Editar una empresa

Editar Empresa

Datos Empresa

Nombre Cif Dirección

Aluvisa G83375579 Calle Benitagla Nº8

Datos Representante

Nombre Nif Teléfono

Eloy Manuel Fernández 44280861L 648905856

Ciclos ☐ ¿Editar Ciclos?

- Desarrollo de Aplicaciones Web
- Desarrollo de Aplicaciones Multiplataforma
- Administración de Sistemas Informáticos en Red

ACTUALIZAR DATOS

Eliminar Empresa

The screenshot shows the 'Empresas' management interface. A confirmation dialog is displayed in the center with the text: '¿Deseas eliminar esta empresa?' and '¡Esta acción no se puede deshacer!'. Below the text are two buttons: '¡Sí, Eliminar!' (blue) and 'No, Cancelar' (red). To the right of the dialog, a callout box asks '¿Eliminar?' and explains: 'En caso de querer eliminar una empresa, nos desaparecerá de la tabla'. The background interface includes a sidebar with 'Inicio', 'Empresas', 'Ciclos', and 'Configuración', and a table of companies with columns for 'Nombre' and 'Dirección'.

Información Empresa / Ficha

Se reflejará todo sobre la fila de la empresa seleccionada

The screenshot shows the 'Información Empresa' page for 'Aluvisa'. The page is divided into several sections:

- Datos Empresa:** Contains the company's Cif (G83375579) and Dirección (Calle Benitagla nº8).
- Información Representante:** Contains the representative's Name (Eloy Manuel Fernández), Nif (44280861L), and Teléfono (648905856).
- Empleados:** Lists the company's employees. The first employee shown is José Miguel Belmonte Rodríguez, with contact information: 27516185C, 665213847, and josemiguel.belmonte.
- Ciclos:** Lists the company's cycles: 'Desarrollo de Aplicaciones Web', 'Desarrollo de Aplicaciones Multiplataforma', and 'Administración de Sistemas Informáticos en Red'.

Callouts provide additional information:

- Ciclos:** 'Se muestran los ciclos con los que trabajan las empresas'.
- Empleados:** 'Podemos Añadir / Editar y Eliminar empleados. Además de mostrarse todos los empleados relacionados con la empresa.'

Ciclos

Añadir
Añadir una nuevo ciclo

Buscador
Filtrar por cualquier campo que vemos en la tabla.

Editar
Nos redirige a una nueva página para editar el ciclo

Eliminar
Nos aparece un modal preguntando si queremos eliminar el ciclo

Tarjetas
Hace la misma función que la tabla de empresas, pero esta vez lo mostramos en tarjetas

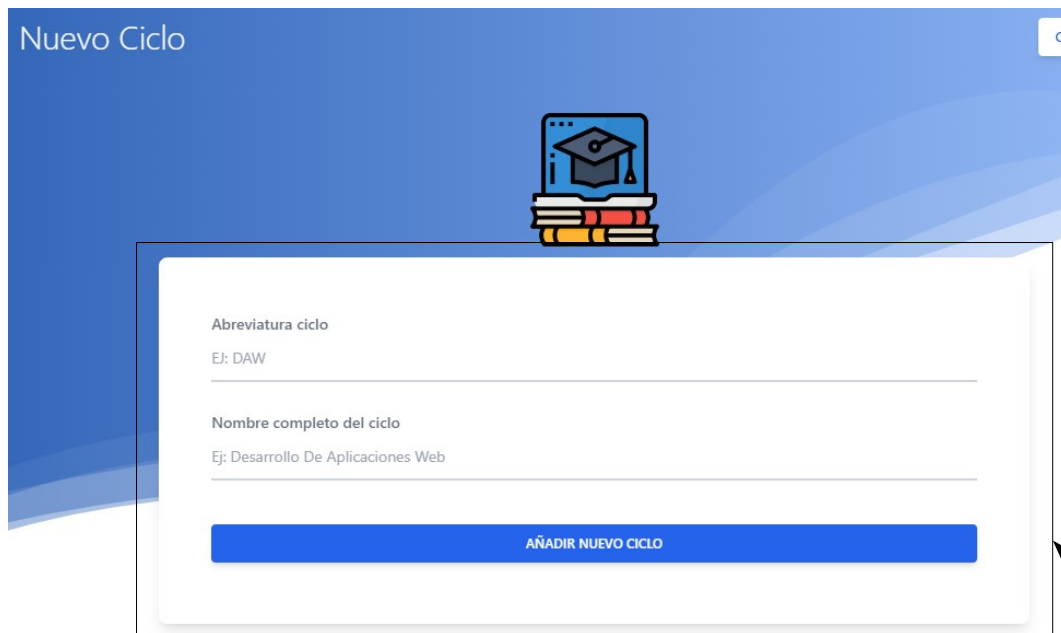
Eliminar Ciclo

¿Eliminar?
En caso de querer eliminar un ciclo, nos desaparecerá de la tabla

Acciones (Añadir / Editar) Ciclo

El formulario añadir/regar y de editar/modificar es completamente igual, excepto que el de editar ya nos autocompleta los campos, podemos cambiarlos si es lo que se quiere. (Se podrá añadir o editar si los campos introducidos son correctos)

Añadir una nuevo ciclo



Nuevo Ciclo

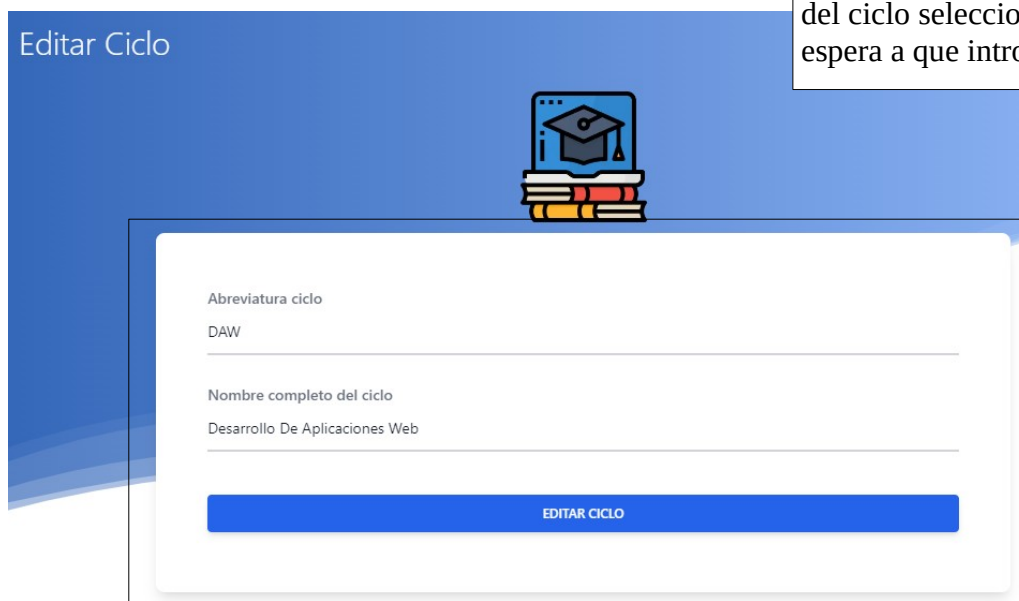
CE

Abreviatura ciclo
Ej: DAW

Nombre completo del ciclo
Ej: Desarrollo De Aplicaciones Web

AÑADIR NUEVO CICLO

Editar un ciclo



Editar Ciclo

CE

Abreviatura ciclo
DAW

Nombre completo del ciclo
Desarrollo De Aplicaciones Web

EDITAR CICLO

Información

Como he comentado, son similares, simplemente editar carga los valores del ciclo seleccionado y nuevo ciclo espera a que introduzcan datos

Configuración / Info. Usuario

Editar Usuario

Botón que nos da la opción de poder modificar nuestros datos como usuario

Información Usuario

Nombre: Nati, Apellidos: Fernández, Email: nati@gmail.com

Botón: Editar Usuario

Información Usuario

Se muestra la información del usuario que está logueado

Configuración / Info. Usuario / Editar

Cambiar Contraseña

En la opción de editar, podemos cambiar la contraseña, si se queda vacía esta contraseña no se cambiará, pero si escribimos en el campo, se validará junto a confirmar contraseña ya que deben de ser iguales

Información Usuario

Nombre: Nati, Apellidos: Fernández, Email: nati@gmail.com

Nueva Contraseña, Confirmar contraseña

Botón: ACTUALIZAR DATOS

Botón: CERRAR SESIÓN

Información

Ahora que estamos en edición, los campos a los que antes no podíamos acceder, ahora si se puede

Aquí estaría completo el manual de usuario.

11.- Bibliografía.

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

<https://tailwindcss.com/>

<https://nextjs.org/docs/api-reference/next/link>

<https://formik.org/>

<https://github.com/jquense/yup>

<https://levelup.gitconnected.com/how-to-run-multiple-queries-at-once-using-graphqls-apollo-client-c24bea52e079>