

Memoria de la práctica final

Diego Fernández Rueda

Elena Martín Naranjo

David Soler Vicén

Enrique Gutiérrez Velasco

Daniel Pastor Miguel

Frontend	3
Desarrollo general y aspecto físico	3
Componentes	3
Button	4
Login (manejo de sesiones)	4
Store	5
Helpers	6
Not-found (manejo de errores)	6
Oferta-informacion	7
Register	7
Table-of-listings	8
Text-input	8
Nueva-oferta	8
Datos a mostrar	9
Navegación	9
Footer	9
Header	9
Navegación entre páginas	10
Páginas	11
Página de inicio del alumno	11
Página de datos de la posición	12
Página de inicio del tutor	13
Página de inicio del responsable	13
Página de login	14
Página de registro	14
Página de generación de informe (tutor)	15
Página de creación de oferta	16
Página de documentación	17
Página de alumnos aceptados en oferta	18
Backend	19
Docker	23

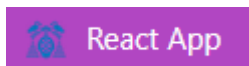
Frontend

Para el desarrollo de la parte de capa de presentación, se ha elegido la opción de la librería de React. Las razones por las que elegir el uso de React para la presentación del frontend han sido, entre otras, la capacidad de acelerar el desarrollo de las aplicaciones. Hemos sido capaces de aprovechar varios componentes listos para crear las funciones de las aplicaciones de una forma más rápida. La modularidad que permite el sistema de módulos hace que esta forma de bloques sea más fácil de integrar al actualizar la aplicación que si no contáramos con ellos. Por otro lado cuenta con la función de recarga activa, la cual permite implementar los cambios en el código y ver el efecto del cambio en la aplicación en el mismo momento. La aplicación está activa y se va actualizando a medida que hacemos cambios. React cuenta con un DOM virtual. Esto resulta útil ya que en caso de cambiar el estado de un componente, el DOM solo cambia este objeto y hace más rápida la compilación

Desarrollo general y aspecto físico

El aspecto físico desarrollado con css

Para el establecimiento del icono presente en cada una de las pestañas que son abiertas de la aplicación web, hemos hecho uso del archivo .ico que se genera al crear un esqueleto para el desarrollo de la web. Este archivo se encuentra en la carpeta “public”, donde también contamos con otros archivos de imágenes para otros iconos en caso de ser necesario.



En el desarrollo de esta práctica cada uno de los componentes, y páginas como tienen un archivo específico del tipo scss. No se realiza un incrustado de estilo mediante un archivo general ya que con nuestro método conseguimos aligerar la carga de datos en cada una de las páginas y componentes.

Componentes

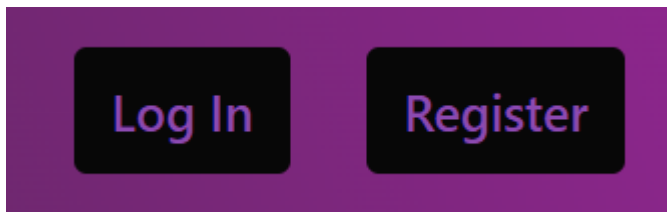
Bajo la carpeta “components” se agrupan aquellos elementos que hemos creado para el uso en las vistas. Estos componentes son reutilizables, es decir, se trata de elementos genéricos que se pueden modificar acorde a las necesidades de la vista.

Los elementos creados son botón, login, not-found oferta-información, register, table-of-listings y text-input.

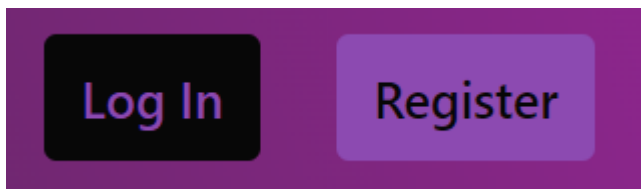
Creando esta carpeta, logramos separar la lógica de cada componente individualmente para hacerlo de una manera modular.

Button

Se trata de un botón simple que se utiliza para ser redireccionados a otra página o que se active una acción desencadenante, como enviar los datos de usuario y contraseña, o recuperar información y mostrarla en pantalla. Junto con el archivo de la lógica, se ha creado un archivo de formato en scss. Este archivo de estilo determina el tamaño, color, y comportamiento cuando el ratón pasa sobre el botón para pulsar. El contenido en texto que el botón pueda tener, se modifica dentro de aquellas vistas que lo utilicen, de tal manera que el mismo componente se utiliza en varias ocasiones.



Cuando se posa el ratón sobre el botón, este cambia de estado.



Este efecto viene definido en el archivo scss:

```
&:hover {  
  background-color: yellow;  
  color: white !important;  
}
```

Login (manejo de sesiones)

Se trata del único componente que se muestra en la página de login. Este está formado por el texto de bienvenida (con formato definido en el archivo login.module.scss), los dos textos de input de email y contraseña y los botones heredados del módulo button del que hablamos más arriba.

Este componente es importante porque va a ser aquel que se encargue de obtener los datos de los usuarios que inician la sesión en la web.

El componente login se encuentra contextualizado dentro del contenedor **<AuthContextProvider>**, el cual provee el servicio onLogin al componente login:

```
const { onLogin } = useContext(AuthContext);
```

Store

onLogin es un método que se encuentra definido en **auth-context.js**. Esta, a su vez se encuentra en la carpeta “store”, en la cual almacenamos los métodos y páginas que nos sirvan para el manejo de sesiones.

El archivo auth-context provee de una serie de métodos para el tratamiento de sesiones, estos elementos se encuentran “escuchando” los eventos que suceden en la web siempre que esos eventos se produzcan bajo el contexto de AuthContext. Por ejemplo, en el formulario login, contenido dentro de lo arriba mencionado, cuenta y hace uso del método para manejo de sesiones de entrada. La constante loginHandler contiene “isLoggedIn”, la cual proporciona una sesión durante el tiempo establecido.

```
const loginHandler = () => {  
  cookieHelper.writeCookie("isLoggedIn", "1", 24 * 60 * 60);  
  setIsLoggedIn(true);  
};
```

El tiempo viene determinado por el tercer parámetro de la función. En este caso, se trata de una sesión de 24h. Es decir, en caso de abrir una sesión y esta no sea cerrada, tenemos 24 horas hasta que esta se cierre automáticamente.

El hecho de que la sesión se encuentre abierta supone que, siempre y cuando el npm start se haya ejecutado y la sesión no se cierre o cambie, siempre que entremos en el localhost de la web, se entrará en el inicio establecido de cada uno de los usuario que haya hecho entrada en el sistema sin necesidad de volver a introducir los credenciales.

Cuando entramos en la página web, con las herramientas de desarrollador, podemos ver que la variable queda almacenada como “1” al iniciar sesión. El almacenamiento de cookies funciona correctamente.

Filter											
Only show cookies with an issue											
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	Sam...	Sam...	Parti...	Pri...
isLoggedIn	1	localhost	/	2023-01-12T09:04:28.000Z	11			Lax			Medi...

Hay que tener en cuenta que la variable que se encuentra oculta en caso de que no se encuentre el componente que estemos usando dentro del contexto de autenticación.

Contamos con los métodos `logoutHandler` y `useEffect`. Sirven para que la sesión termine al cambiar el valor de la variable booleana y para obtener información respectivamente.

`UseEffect` es un hook que utilizamos para actualizar el estado del componente, No necesitamos una API especial para acceder a ella, ya que se encuentra en el ámbito de la función.

```
useEffect(() => {  
  if (!isLoggedIn) {  
    navigate(urlPaths.home);  
  }  
}, [isLoggedIn]);
```

Se implementa un método con el código arriba mostrado que evita el acceso a contenido no autorizado sin una sesión abierta. La variable `isLoggedIn` comprueba si está activada a la hora de poder acceder a la página de entrada. Así mismo, tampoco puede acceder una vez iniciada sesión un usuario a la página de inicio de sesión (Login) en caso de estar ya iniciada.

Helpers

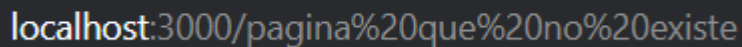
Esta carpeta es también parte del manejo de las cookies y sesiones en la página web. En ella encontramos `cookie.js`

Este archivo contiene las funciones de creación de una cookie, de lectura de los contenidos de esta, y de la eliminación.

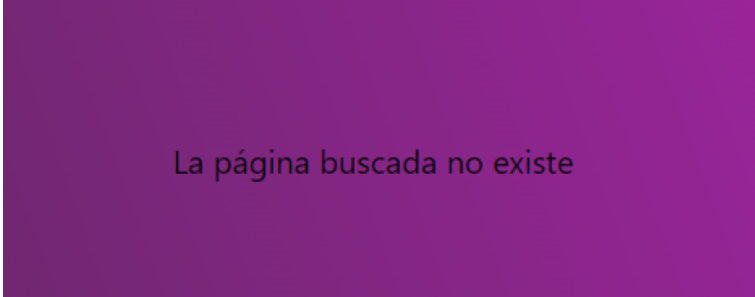
Not-found (manejo de errores)

Para el manejo de errores, se ha desarrollado un componente adicional. A este componente solamente se puede acceder cuando un error salta al acceder a contenido. Por ejemplo, en el caso de que en la barra de navegación se trate de acceder a un servicio en nuestra web no registrado en los links de navegación. La ventaja de usar un componente que sea el que muestra esta información, y no una vista a parte dedicada a ello, es que no hay necesidad de cambiar valores de sesión, el componente se muestra en la vista usada y de esta manera

no se tiene que hacer un link general desde todas las páginas de nuestra web a esta que nos muestra el error.



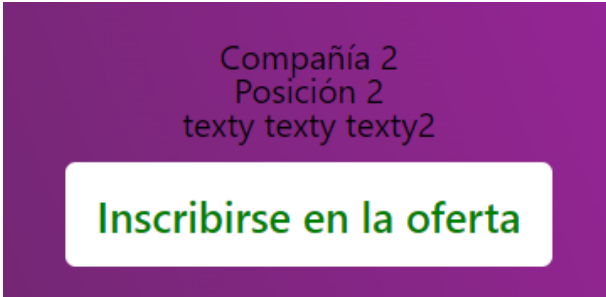
localhost:3000/pagina%20que%20no%20existe



La página buscada no existe

Oferta-informacion

Componente que muestra la información de las ofertas (compañía, posición e información extra). Contamos con un botón del tipo submit, este botón se encargará de pasar al backend la información del alumno para que quede registrado como inscrito en la oferta. Se hace uso de los datos que se encuentran en datos-mostrar para que se pueda ver la información de la oferta y compañía al seleccionar la oferta. Tanto los datos que se muestran como el botón de inscripción aparecen descritos en cuanto a aspecto en el archivo scss.



Compañía 2
Posición 2
texty texty texty2

Inscribirse en la oferta

Register

Componente usado en la vista dedicada al registro de los responsables en la aplicación.

Al igual que el componente de login, este componente se encuentra contextualizado en AuthContext, el cual nos dará acceso al manejo de sesiones en la web.

Se trata de un formulario de registro con los campos de nombre, e-mail y contraseña. Cuando los datos son introducidos y registrados, una nueva sesión es iniciada en la web. Podemos acceder a los contenidos del usuario. De nuevo, como en el login, se hace uso de las cookies y los contextos con sus variables para que funcione correctamente.

Nombre

Tu nombre

Email

example@gmail.com

Contraseña

Registrar

Table-of-listings

Componente usado para mostrar en formato tabla los datos que necesitamos. Hacemos uso del archivo datos-mostrar. El estilo de la tabla se establece en el archivo style.module.scss

Compañía 2	Posición 2
Compañía 3	Posición 3
Compañía 4	Posición 4

El acceso a cada una de las posiciones de la tabla que son mostrados se puede realizar clicando en las celdas de la tabla.

Text-input

Se trata de un componente reutilizable utilizado para la introducción de texto. Obtiene un valor y un tipo. Este componente se utiliza en otras webs como generación de nueva oferta. Es necesario que se importe para su utilización.

Titulo de posicion

Nueva-oferta

Contenedor compuesto de text inputs. Formado por tres cuyos valores se establecen en la clase y obtienen los datos con el uso métodos como setTitulo.


```
const [titulo, setTitulo] = useState("");
const [turnos, setTurnos] = useState();
const [semanas, setSemanas] = useState();
```

Datos a mostrar

Carpeta (datos-mostrar) con almacenamiento de los datos por cada uno de los valores que necesitamos. Cada uno de los valores cuentan con atributos con su respectiva información.

```
{
  companyName: "Compañía 2",
  position: "Posición 2",
  info: "Informacion",
  id: 2,
},
```

Navegación

Footer

Mostramos información que se va a encontrar en todas las páginas que hay en la web. Se encuentra en el fondo inferior de la página. Tenemos un apartado de enlaces importantes. Uno con la información de login a modo de inicio, el otro con el enlace de la documentación del proyecto.

Se muestra otra columna donde están los créditos de los autores del trabajo.

Enlaces importantes

Login

Documentación del trabajo

Trabajo realizado por

Diego Fernández Rueda

Elena Martín Naranjo

David Soler Vicén

Enrique Gutiérrez Velasco

Daniel Pastor Miguel

Header

Se encuentra en la parte superior de todas las páginas de la web. Tiene un diseño cambiante que va a depender del manejo de las sesiones. En un primer momento, sin sesión iniciada, el header va a mostrar dos botones para acceder a las páginas de login y registro.

Login Register

Una vez que se ha hecho cualquiera de las dos, el header cambia su disposición para mostrar dos botones que van a desarrollar funciones muy diferentes.

Inicio

Cerrar sesión

El botón inicio nos conducirá al menú de inicio de cada uno de los usuarios. Esto es, dependiendo del tipo de usuario que haya iniciado la sesión, se conducirá a una vista u otra de la web. El botón de cerrar sesión borrará los datos del usuario en las cookies y la sesión finalizará. Se dirige de nuevo al inicio de la web donde se accede sin la sesión iniciada para hacer login o el registro.

Mostrar esquema de navegación

Navegación entre páginas

La navegación entre páginas se produce gracias a los archivos navigation.js y url-path.js. Estos archivos contienen los enlaces a cada una de las páginas de la web. Es necesario hacer una importación de estos dos archivos en el resto de las webs para poder redirigir a otras webs dentro de la vista el archivo URL paz contiene simplemente la ruta relativa de cada uno de las páginas dentro de la web sin embargo, navigation.js , contiene tanto la ruta relativa como los caminos y los elementos que cada uno de ellos representa.

Con el siguiente código, en navigation.js, obtenemos una plantilla para la creación de una nueva página.

Esta nueva página servirá para la información de las prácticas. Es un método que no solamente se usa para esta página, sino también para la información de los alumnos en cada oferta.

```
<Route path={urlPaths.nuevaOferta} element={<NuevaOferta />} />
<Route path={`${urlPaths.information}/${id}`}
element={<Information />} />
<Route path={`*`} element={<NotFound />} />
```

La plantilla, llamada información, requiere de el id de la oferta para crear la nueva. A continuación, se comprueba si ese id se corresponde con uno real existente. En caso de que no exista dicho id, se muestra por pantalla un mensaje de error materializado en el componente Not Found.

Páginas

Las páginas de nuestra web son cada una de las vistas que componen el conjunto de esta. Entre las que hemos creado se encuentran el inicio del responsable, el inicio del alumno, el inicio del tutor, el login, el registro y los datos. Cada una de estas páginas hacen uso de los componentes reutilizables que hemos mencionado más arriba. Además, cada una de las páginas se encuentran englobadas en distintos contextos como authentication context que nos dará una serie de funcionalidades y variables que nos ayudarán tanto a iniciar sesión como mantenerla y mostrar los datos vinculados a cada uno de los usuarios.

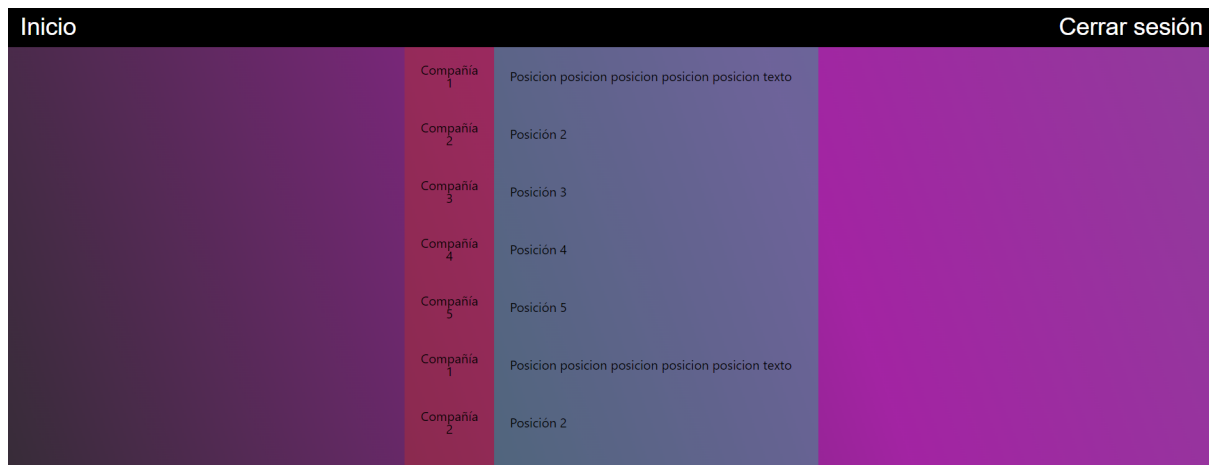
No todas las vistas pueden ser accedidas desde todas las demás. Existe una serie de métodos de acceso a las vistas que se encuentran disponibles por los propios componentes. Decir, una vista cuenta con botones y métodos que nos permiten acceder a otras vistas. Si la sesión ha sido iniciada correctamente respecto a la vista a mostrar, esta vista aparecerá. De lo contrario, el componente de error de página aparecerá, indicando que el componente al que hemos intentado acceder no existe o no se encuentra disponible.

La mayoría de las páginas usadas en esta web son estáticas. Sin embargo, hay webs que son generadas dinámicamente. Es decir dependiendo de la cantidad de datos que manejamos en la carpeta mostrar datos, se generará una nueva vista. Este es el ejemplo de mostrar oferta para los alumnos y mostrar los alumnos en la oferta para los tutores. Las ofertas se pueden encontrar ya inscritas en el sistema o pueden ser generadas por el tutor. En ese caso, la nueva vista será creada para que los alumnos puedan inscribirse.

Tanto el tamaño de la tabla mostrado en el inicio como las webs generadas a partir de las celdas de cada una se generarán automáticamente.

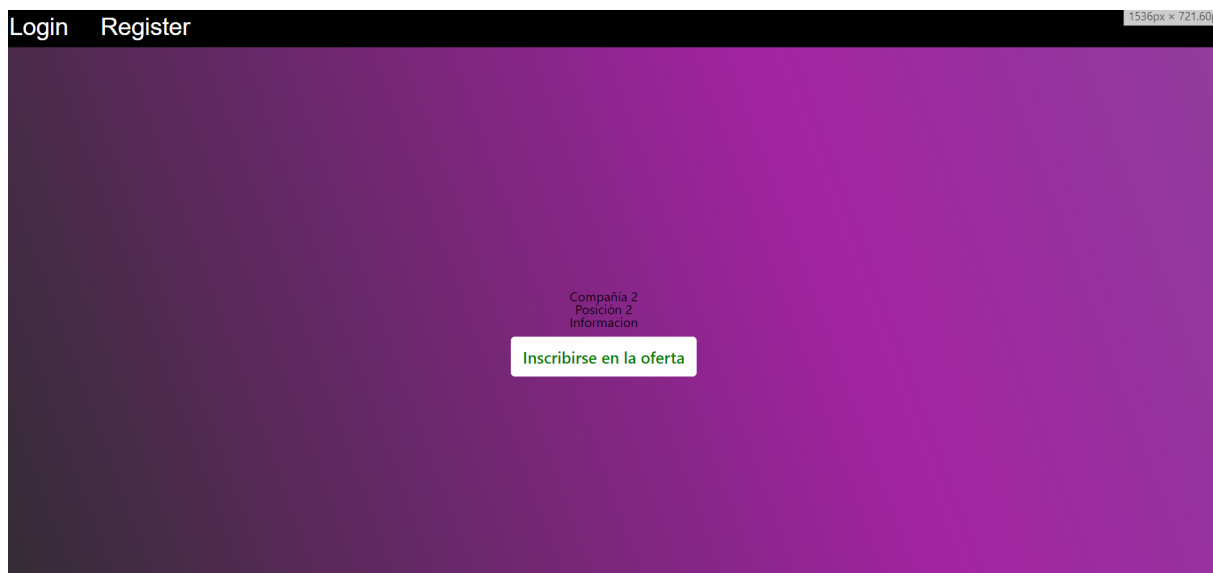
Página de inicio del alumno

La página de inicio del alumno muestra el comienzo de la tabla con las ofertas a las que puede inscribirse. Pulsando sobre cada una de las ofertas en la tabla, se le dirigirá a cada una de las ofertas con su información y la opción de ser inscrito. Esta página constituirá el inicio al que se podrá acceder desde el header en el caso de que la sesión corresponda con un usuario de alumno. Es decir, cuando pulsemos en el botón de inicio con la sesión abierta nos mostrará la página de la que estamos hablando.



Página de datos de la posición

La página de datos de la posición es accedida al pulsar en cada una de las posiciones ofertadas en el inicio del alumno. Cada una de estas vistas es generada automáticamente al ser añadido nuevos datos dentro de la carpeta mostrar datos. Dentro de esta página, mostramos el nombre de la compañía, la descripción de la posición e información. Contamos con un botón que nos permite inscribirnos a la oferta si somos alumnos, el dato que le dará guardado con las credenciales del alumno y se mostrará al responsable cuando lo consulte.



Cada una de las vistas generadas se llamarán como el identificador de cada oferta. En el URL podremos ver que, bajo el apartado Información, el único nombre identificador, será su número.

`localhost:3001/information/2`

Página de inicio del tutor

La página de inicio del tutor supone la vuelta al inicio cuando se pulsa en el botón de Heather en el caso de que se haya iniciado sesión como un usuario tutor. Contiene una lista del tipo componente table-of-listings. En la parte inferior tiene los botones del tipo Button definido también como componente que dirigen a la página de añadir oferta y a la página de registro de un nuevo tutor. Esta última puede ser accedida sin iniciar una sesión como usuario registrado



Por otro lado, contamos también con el botón de generación de un informe sobre un alumno.

Página de inicio del responsable

El portal de inicio de responsable, con una tabla que muestra las mismas ofertas que ha colgado a los alumnos. En cada una de las posiciones ofertadas, se puede pulsar para acceder a la vista de alumnos aceptados en la oferta.

Compañía 1	Posicion posicion posicion posicion posicion texto
Compañía 2	Posición 2
Compañía 3	Posición 3
Compañía 4	Posición 4
Compañía 5	Posición 5
Compañía 1	Posicion posicion posicion posicion posicion texto
Compañía 2	Posición 2

Página de login

La página de login cuenta con el componente login. Este componente es el único que se muestra en la página web. Esta página será la primera a la que accedemos al entrar en la web , una vez accedemos con nuestras credenciales, esta página se abandonará y accederemos al inicio de cada uno de los usuarios.

Bienvenido al portal de prácticas de la universidad de Alcalá de Henares

Email

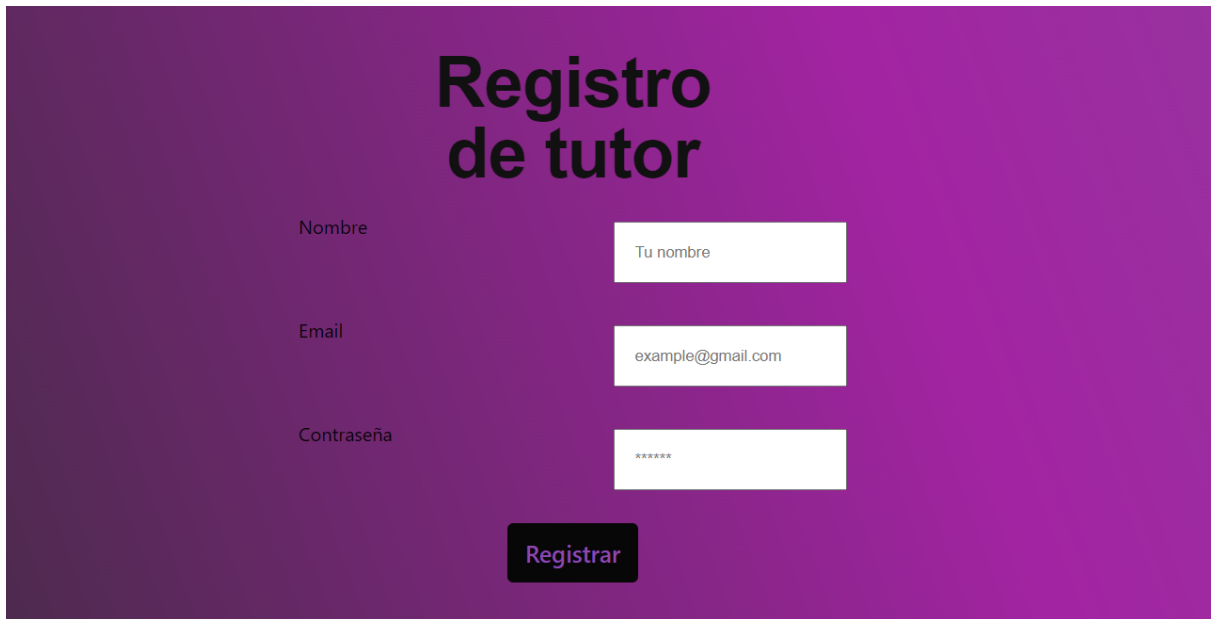
Contraseña

Log In

Register

Página de registro

La página de registro se puede acceder desde el header o desde la propia página de login. Al introducir las credenciales de nombre, e-mail y contraseña, podemos pulsar el botón, activar la cookie que guarda nuestra sesión y navegar por la web. El único componente que está página tiene es el módulo register.jsx.

A registration form titled "Registro de tutor" on a purple gradient background. The form includes three input fields: "Nombre" with the placeholder "Tu nombre", "Email" with the placeholder "example@gmail.com", and "Contraseña" with the placeholder "*****". Below these fields is a black button with the text "Registrar" in white.

Es importante comprobar en esta página web que el manejo de sesiones funciona correctamente. Mientras que en la página de inicio de sesión no puede ser accedida una vez está iniciada, la página de registro siempre está accesible incluso con una sesión abierta. Esto es debido a que la comprobación de presencia de sesión abierta afecta solamente a las páginas restringidas tanto si está abierta la sesión como si no.

Página de generación de informe (tutor)

Esta página es simplemente accesible desde el portal de inicio del tutor. Para la generación del informe añadiendo los datos del alumno la nota y unas notas adicionales. El botón de generar informe es del tipo submit, esto es, obtiene los datos de aquellos valores de los text input y los pasa a los datos capturados.

Generar informe

Generar informe

Página de creación de oferta

Página para la creación de nuevas ofertas en el usuario tutor. Contiene cuatro text inputs heredados de los componentes de valores título, turnos, semanas y alumnos. Un botón del tipo submit activa la función handle Submit que almacena el título, turno, semanas y alumnos. A excepción del valor título los tipos de datos introducidos son del tipo Integer.

Nueva Oferta

Publicar oferta

Página de documentación

En esta página de la web hacemos uso de una funcionalidad que nos ofrece Google Docs. Se nos permite hacer un incrustado dentro de una página web para ser publicado nuestro documento. Una vez se crea la página web y se añade su enlace a la lista de navegación, al botón de entrada en el footer, y se crea el esqueleto, solo falta crear un archivo de estilo para dar formato al documento incrustado.



Creamos el estilo para el iframe, el componente que nos ofrece Google Docs para mostrar el documento. Asignamos una altura y una anchura para que se muestre correctamente en el en la página web.

```
.documentation-container {  
  iframe {  
    width: 50vw;  
    height: 90vh;  
  }  
}
```

Página de alumnos aceptados en oferta

En las páginas previas, se explica de la existencia de un archivo llamado datos mostrar esta carpeta que sirva de almacenamiento de datos temporal, fue creada durante el desarrollo del Front End para simular datos que podemos obtener de la parte del back end. Estos datos son de una naturaleza simple y no cuentan con relaciones entre sí ni entre las tablas que las contienen. Al tratarse de una vista muy compleja de realizar sin estos datos, no se ha podido terminar el trabajo. Tanto la tabla que debería mostrar los datos como el botón de generar una elección de los alumnos dependen exclusivamente de estos datos que no tenemos en este momento.

Backend

Para el backend hemos escogido la opción de implementar servlets de java que se conecta a una base de datos mariadb a través del driver jdbc.

Para la base de datos hemos implementado el patrón Singleton, así evitamos crear múltiples instancias de este objeto.

```
/**
 * Atributo que referencia una base de datos para la implementacion del patron singleton
 */
private static BaseDatosUtil instancia;

private BaseDatosUtil() {

}

public static BaseDatosUtil getInstancia() {
    if (instancia == null) {
        instancia = new BaseDatosUtil();
    }
    return instancia;
}
```

La conexión la realizamos con jdbc como hemos comentado anteriormente:

```
/**
 * Abre una conexion con una base de datos mysql usando el drive jdbc en localhost:3306 y credenciales root:root
 *
 * @throws ClassNotFoundException
 * @throws SQLException
 */
public void abrirConexion() throws ClassNotFoundException, SQLException {
    String driver = "com.mysql.cj.jdbc.Driver";
    String url = "jdbc:mysql://localhost:3306/gestionPracticas?useTimezone=true&serverTimezone=Europe/Madrid";
    String user = "root";
    String password = "root";
    System.out.println("Entramos a conectar con la BBDD");
    if (conexion != null) {
        System.out.println("Ya hay una conexion");
    } else {
        System.out.println("Creamos una nueva conexion");
        Class.forName(driver);
        conexion = DriverManager.getConnection(url, user, password);
        System.out.println("Finalizamos config db");
    }
}
```

A la hora de interactuar con la base de datos hemos seguido la siguiente estructura:

- El método devuelve true si se ha podido realizar la acción exitosamente.
- Devuelve false si no ha sido posible (y elimina las acciones que se hayan podido realizar anteriormente)
- Utilizamos PreparedStatements para facilitar la legibilidad del código.

Ej:

```

public boolean registrarTutor(String nombreUsuario, String contrasena, String nif, String empresaCif, String cargo) throws
String consulta = "INSERT INTO Persona (`usuario`, `contrasena`, `nif`) VALUES (?, ?, ?)";
ps = conexion.prepareStatement(consulta);
ps.setString(1, nombreUsuario);
ps.setString(2, HashPasswordUtil.generarPasswordHash(contrasena));
ps.setString(3, nif);
try {
    ps.executeUpdate();
} catch (SQLIntegrityConstraintViolationException ex) {
    ps.close();
    return false;
}
consulta = "INSERT INTO Tutor (`empresa_cif`, `persona_usuario`, `cargo`) VALUES (?, ?, ?)";
ps = conexion.prepareStatement(consulta);
ps.setString(1, empresaCif);
ps.setString(2, nombreUsuario);
ps.setString(3, cargo);
try {
    ps.executeUpdate();
} catch (SQLIntegrityConstraintViolationException ex) { //si no se puede insertar el tutor porque la empresa no existe,
    consulta = "DELETE FROM Persona where usuario = ?";
    ps = conexion.prepareStatement(consulta);
    ps.setString(1, nombreUsuario);
    ps.executeUpdate();
    ps.close();
    return false;
}
ps.close();
return true;
}

```

Para insertar un tutor, primero insertamos una Persona (devolviendo false si el nombre de usuario ya está utilizado), posteriormente insertamos un Tutor. Sin embargo, si la empresa del tutor no existe, entonces se devuelve false ya que no se ha podido crear el tutor correctamente. Además, debemos eliminar la entrada en la columna Persona insertada anteriormente, para que ese nombre de usuario usado no quede cogido.

Otro ejemplo de método es el de obtener todas las ofertas de prácticas:

```

public ArrayList<OfertaPracticas> getAllOfertasPracticas() throws SQLException {
    ArrayList<OfertaPracticas> ofertasPracticas = new ArrayList();

    statement = conexion.createStatement();
    rs = statement.executeQuery("SELECT * FROM oferta_practicas");
    while (rs.next()) {
        ofertasPracticas.add(
            new OfertaPracticas(
                rs.getInt("id_oferta_practicas"),
                rs.getString("descripcion"),
                rs.getString("titulo"),
                rs.getString("idiomas"),
                rs.getString("requisitos"),
                rs.getDate("fecha_inicio"),
                rs.getString("curso"),
                rs.getInt("duracion"),
                rs.getTime("hora_inicio"),
                rs.getTime("hora_salida"),
                rs.getInt("plazas"),
                rs.getInt("salario"),
                rs.getString("tutor_persona_usuario"),
                rs.getString("tutor_empresa_cif")
            )
        );
    }
    rs.close();
    statement.close();
    return ofertasPracticas;
}

```

En cuanto a los servlets, aquí se muestra el utilizado para iniciar sesión:

```
@WebServlet(name = "InicioSesion", urlPatterns = {"/InicioSesion"})
public class InicioSesion extends HttpServlet {

    private BaseDatosUtil bd;

    @Override
    public void init(ServletConfig cfg) throws ServletException {
        bd = BaseDatosUtil.getInstancia();

        try {
            bd.abrirConexion();
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(Servlet.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SQLException ex) {
            Logger.getLogger(Servlet.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

con un método post que recibe un nombre y un usuario y comprueba si son alumno, tutor o responsable para redirigir a una u otra página.

Nos ha parecido una prioridad encriptar las contraseñas almacenadas en la base de datos para evitar un grave incidente de ciberseguridad:

Mostrar: persona

[Visualizar contenido](#) [Mostrar estructura](#) [Modificar tabla](#) [Nuevo Registro](#)

Mostrar	Condición	Ordenar	Limite 50	Longitud de texto 100	Acción Mostrar
---------	-----------	---------	--------------	--------------------------	-------------------

SELECT * FROM 'persona' LIMIT 50 (0.001 s) [Modificar](#)

<input type="checkbox"/> Modify	usuario	contrasena	nif
<input type="checkbox"/> modificar	admin	1000:f6a9f76f9a90fdd0a45945f7957fb551:5fd52298f6e7c7127576b25a77fb928a53239a8de6756dc9d21380bd615fc2...	xxxxxxxxxx
<input type="checkbox"/> modificar	alba.ramos	1000:21dab995242070b696c7522135d0e549:1a7747b47ba942821beda50c5607ff9063e043f417fa19b7e1f56627eb8df1...	12345678A
<input type="checkbox"/> modificar	alberto.torres	1000:ccc265a7e9d9a25d836fcbbe5acfb151:b1b6755925cbfd832a53173769baf9248472d9c83fd73e3e8b2df73cac806b...	12332456A
<input type="checkbox"/> modificar	alejandro.mendez	1000:f0073d812661d6608865da97f75f9f61:b9f544ad895108b5aae4ebc03d90541da9200382927923741e3dcf29467315...	98798754A
<input type="checkbox"/> modificar	alumno	1000:d3ebd50fddd1495b4968a093596fb5c:f0b6584f3b281c084400b654cee80b8b744d27248c0f830623c4fdd7fd51d...	zzzzzzzzzz
<input type="checkbox"/> modificar	andres.hernandez	1000:9158ebcaa7814414cce66373e3c64d6a:d2f671915ebe9d2fd4c4d2fd8c5449cbdc831d5d7f8c5109f6a492eb978f2...	76547566M
<input type="checkbox"/> modificar	bernardos.lopez	1000:374a407fcbcb59eef0811467cb8e6092:889c63903c5784c10b7f9e588e70c2032fbc7d0a4b59d1429727d63d467b99...	45455524J
<input type="checkbox"/> modificar	carla.vazquez	1000:2a8526b997cb9d95af29c5ab37b95fc1:5263e0739f6d21b70be5a440ae122dd2db5212eeeb1906a84f7ac61c06de37...	00986745B
<input type="checkbox"/> modificar	carlos.cetino	1000:b6a6daaffd07328b2f585f3fa59162fe6:8630d6db993252ea9c59ea2d795c37e243f1f812745cd0f587a28a8ed179dd...	90689954K
<input type="checkbox"/> modificar	cristina.sanchez	1000:fda3bfa9030f90c15c27fb83831d0142:68b829a9080be86c78d63540181b5d15ff97ab334ccc584d34479369747147...	97658976C
<input type="checkbox"/> modificar	daniel.garcia	1000:d2ece4fc0d7f5b93448fb533a44ccb6d:3d390530cd6ef4aa34d777806685289e7f5409a3a0b74d56d64ad82da843d6...	34573689M
<input type="checkbox"/> modificar	elena.gutierrez	1000:5237638f4c46de044ac5908b1005794d:baf498fb10599e4513cc4e63cde85191838b3f342dfab643563c10fa5c4881...	53457824M
<input type="checkbox"/> modificar	ibrahim.alcaide	1000:bb2e108c6f1cdef8ecf81ed52fd56ba5:776d46120eda21b5b76e0c932aebbc74971b979eac87042e95c1196f2d1d52...	48593453H
<input type="checkbox"/> modificar	jaimel.tripi	1000:8db3e64faa1b3c8f84aa0c2b6f4fc203:c1f657d7869ce074d87433154d8a4661e1c16a9ea8529136913b31ce554d47...	77653676H
<input type="checkbox"/> modificar	jesus.berlanga	1000:1fe418117883d738f23a66625554e956:dadc875f4b2357494c8623443dd2f98068909110e04bf282ea03743bf9dfdea...	76547567K
<input type="checkbox"/> modificar	jose.juan	1000:0cd09a15bbcf5a0f8cc16d687218c3b7:5436f98e4a65b56e3ee8cfe7a21c56f330f3358ea2f82b487007adb3de8...	77684645Y
<input type="checkbox"/> modificar	juan.alberto	1000:f628c979d6585894517dde574271ec64:4b9e4bedabc49174ecaf35049a4fb4994a5c15824c644432cf961b8cf3d8ae...	23111232B
<input type="checkbox"/> modificar	juan.carlos.herranz	1000:1e4896a522c355d562f025f9015dd8e0:0dc47478f7b2a28e6e71cecaa8a0d429f38d0aef2049044e1ab663f769a8b...	56435643Y
<input type="checkbox"/> modificar	juan.olmeda	1000:2470a90f220a897fc0252da96e56f6ff:16ffd380a1c2db0cb98c5efaa149fe1bc28511d316e421505e34c617c9bb64...	44534724D
<input type="checkbox"/> modificar	juan.palomo	1000:67c3d02c41721ea30c2213a0b07669e:07a079733684b5cc9df3aeb76e02599864a7068c7f2cc922b6a900a27e02bd...	67546776G
<input type="checkbox"/> modificar	juan.tome	1000:3c6ff51e33be1e6f7d3d9c915c634841:10f32aa3d1487b01015d7b4c0155ceaab25698e4653dede398d09002be9111...	64888832C
<input type="checkbox"/> modificar	lucia.perez	1000:491ba9428bcd3df98cb8c506fc19f77:9608ea50b054d3fc7ae346c11f65e0cf3ea87d618847318ae6882a993696a4...	53559084J
<input type="checkbox"/> modificar	luis.miguel	1000:490d9c892818c632ea1f102497dddd68:2d75d32207d8f122ee869a9fbbd116af11ff9ddf89ade5f395aef1e94f1bba...	54444345M
<input type="checkbox"/> modificar	luis.pinto	1000:9256b9d54ecc617e6bab1c39656974ad:3b64eec76d58c1619155516bfdf7ae98365968412188ead60bf888bf4abc62...	34434232N
<input type="checkbox"/> modificar	marta.saez	1000:4eff5cda9528f510ee6e7f3930a74e8b:38364f8b99edb37ee32cda3413dbe5f1fdb23d5938491c2a8512a44aef2c62...	35464444C

Hemos utilizado un algoritmo de encriptado PBKDF2WithHmacSHA1 de 166 caracteres el cual le hemos sumado un salt de 16 caracteres aleatorios usando el algoritmo SHA1PRNG.

Esto aumenta de manera exponencial el tiempo necesario para crackear las hashes.

```
/**
 * Genera una hash con un algoritmo de encriptado PBKDF2WithHmacSHA1 de 166 caracteres
 * @param password. Contraseña sobre la que se quiere crear una hash
 * @return El hash generado
 * @throws NoSuchAlgorithmException
 * @throws InvalidKeySpecException
 */
public static String generarPasswordHash(String password) throws NoSuchAlgorithmException, InvalidKeySpecException {
    int iterations = 1000;
    char[] chars = password.toCharArray();
    byte[] salt = getSalt();

    PBKeySpec spec = new PBKeySpec(chars, salt, iterations, 64 * 8);
    SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");

    byte[] hash = skf.generateSecret(spec).getEncoded();
    return iterations + ":" + toHex(salt) + ":" + toHex(hash);
}

/**
 * Crea 16 caracteres aleatorios usando el algoritmo SHA1PRNG
 * @return Los caracteres aleatorios
 * @throws NoSuchAlgorithmException
 */
private static byte[] getSalt() throws NoSuchAlgorithmException {
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    byte[] salt = new byte[16];
    sr.nextBytes(salt);
    return salt;
}
```

Luego disponemos de una función que realiza el procedimiento inverso para descryptar la hash y obtener el valor original del string:

```
/**
 * Compara si una cadena de texto simple y otra hashada con el algoritmo PBKDF2WithHmacSHA1 son iguales
 * @param decodedPassword
 * @param hashedPassword
 * @return Si son iguales o no
 * @throws NoSuchAlgorithmException
 * @throws InvalidKeySpecException
 * @throws NumberFormatException. Si se introduce una cadena que no este hashada como argumento de hashedPassword
 */
public static boolean validarPassword(String decodedPassword, String hashedPassword)
    throws NoSuchAlgorithmException, InvalidKeySpecException, NumberFormatException {
    String[] parts = hashedPassword.split(":");
    int iterations = Integer.parseInt(parts[0]);

    byte[] salt = fromHex(parts[1]);
    byte[] hash = fromHex(parts[2]);

    PBKeySpec spec = new PBKeySpec(decodedPassword.toCharArray(),
        salt, iterations, hash.length * 8);
    SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
    byte[] testHash = skf.generateSecret(spec).getEncoded();

    int diff = hash.length ^ testHash.length;
    for (int i = 0; i < hash.length && i < testHash.length; i++) {
        diff |= hash[i] ^ testHash[i];
    }
    return diff == 0;
}
```

Este método se utiliza principalmente a la hora de iniciar sesión:

```

public boolean login(String nombreUsuario, String contrasena) throws SQLException, NoSuchAlgorithmException, InvalidKeySpecException {
    Persona usuario = getUser(nombreUsuario);
    return HashPasswordUtil.validarPassword(contrasena, usuario.getContrasena());
}

```

La última mejora introducida en el proyecto es la creación y recolección de logs. Un ejemplo:

```

/**
 * Cambia la contraseña de un objeto tipo Persona en una base de datos
 *
 * @param persona
 * @param contrasena
 * @throws Exception
 */
public void cambiarContrasena(Persona persona, String contrasena) throws Exception {
    if (persona == null) {
        NewMain.logBBDD.log(Level.WARNING, "El usuario "+persona.getUsuario()+" no existe");
        NewMain.log.log(Level.WARNING, "Error al cambiar la contraseña del usuario"+ persona.getUsuario() );
        throw new Exception("La persona no existe");
    }

    persona.setContrasena(HashPasswordUtil.generarPasswordHash(contrasena));
    String consulta = "update Persona set contrasena = ? where usuario = ?";
    ps = conexion.prepareStatement(consulta);
    ps.setString(1, persona.getContrasena());
    ps.setString(2, persona.getUsuario());
    ps.executeUpdate();
    NewMain.logBBDD.log(Level.INFO, "Contraseña de usuario "+persona.getUsuario()+" cambiada");
    NewMain.log.log(Level.INFO, "Contraseña de usuario "+ persona.getUsuario()+ " cambiada");
    ps.close();
}

```

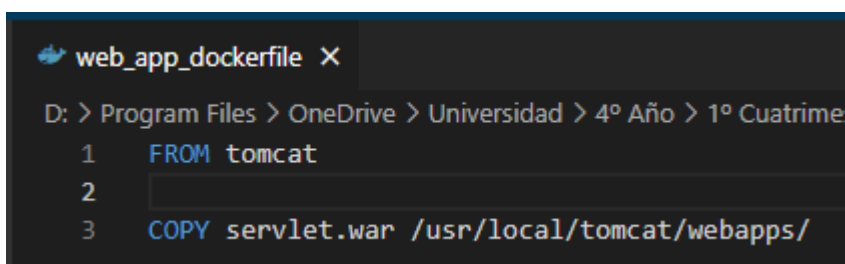
Se generan logs tanto para los errores como para información del estado de ejecución del sistema.

Docker

Hemos utilizado la herramienta de contenedores Docker para desplegar el backend por completo, desde los servlets de java hasta la base de datos.

Para no tener que introducir el proyecto java de servlets compilado como volumen, hemos decidido crear una imagen que tenga ya el fichero introducido para que se inicie con tomcat directamente.

Para ello utilizamos el siguiente Dockerfile:



```

web_app_dockerfile X
D: > Program Files > OneDrive > Universidad > 4º Año > 1º Cuatrime
1 FROM tomcat
2
3 COPY servlet.war /usr/local/tomcat/webapps/

```

Lo ejecutamos para crear la imagen:

```

D:\Program Files\OneDrive\Universidad\4º Año\1º Cuatrimestre\Arquitectura y Diseño de Sistemas Web y C-S\Laboratorio\Prá
ctica final\Base de datos\setup>docker build -f web_app_dockerfile -t servlet-app .
[+] Building 11.5s (7/7) FINISHED
=> [internal] load build definition from web_app_dockerfile 0.0s
=> => transferring dockerfile: 103B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/tomcat:latest 1.8s
=> [internal] load build context 0.2s
=> => transferring context: 13.04MB 0.2s
=> [1/2] FROM docker.io/library/tomcat@sha256:dd920d167352e9e21b297dbe08c54ca35c445c100f38bf9be9a8fa85c9196e7f 8.7s
=> => resolve docker.io/library/tomcat@sha256:dd920d167352e9e21b297dbe08c54ca35c445c100f38bf9be9a8fa85c9196e7f 0.0s
=> => sha256:1dad8c5497f9a16cd95b04d48c7e1cf893bd9f3996101d49e6039d3fc09471aa 12.15kB / 12.15kB 0.0s
=> => sha256:dd920d167352e9e21b297dbe08c54ca35c445c100f38bf9be9a8fa85c9196e7f 1.21kB / 1.21kB 0.0s
=> => sha256:10362751d14315fd94f1afb61b768fa8e48f999766afe3c8438e5c3a59bf01a4 1.79kB / 1.79kB 0.0s
=> => sha256:4d8d923227d8dc300e1ddab1b65c83c0efff7f4a7105d958420872f7138b79735 16.97MB / 16.97MB 0.8s
=> => sha256:eda8241fd25fe52c584da502f21cef5b5067551f7bcfcca58e9a630238f7f558 192.44MB / 192.44MB 5.2s
=> => sha256:35dccabde73de58115ed8db691bd335b82d91b6961c0d1af77691b368b0a8253 175B / 175B 0.8s
=> => extracting sha256:4d8d923227d8dc300e1ddab1b65c83c0efff7f4a7105d958420872f7138b79735 1.6s
=> => sha256:978c906bcdda3ea15fcb8a67356062f1adad5fe2eaedf43a1791858a3bc96a5b 171B / 171B 1.1s
=> => sha256:45999e75f51e89a4e81e2d7a03b01b73da4b198c22ea5fa9e9de965642eb071a 12.67MB / 12.67MB 1.4s
=> => sha256:ecd30916ffc4583ae5c0b563e5cba5fbf6ce4c3ef2431563b4d4f9c05e24348c 131B / 131B 1.5s
=> => extracting sha256:eda8241fd25fe52c584da502f21cef5b5067551f7bcfcca58e9a630238f7f558 2.0s
=> => extracting sha256:35dccabde73de58115ed8db691bd335b82d91b6961c0d1af77691b368b0a8253 0.0s
=> => extracting sha256:978c906bcdda3ea15fcb8a67356062f1adad5fe2eaedf43a1791858a3bc96a5b 0.0s
=> => extracting sha256:45999e75f51e89a4e81e2d7a03b01b73da4b198c22ea5fa9e9de965642eb071a 0.2s
=> => extracting sha256:ecd30916ffc4583ae5c0b563e5cba5fbf6ce4c3ef2431563b4d4f9c05e24348c 0.0s
=> [2/2] COPY servlet.war /usr/local/tomcat/webapps/ 0.7s
=> exporting to image 0.1s
=> => exporting layers 0.1s
=> => writing image sha256:b3a28aaf66165529d8cc006194e286a959f24980a5b52bc05c0457d67c5535b8 0.0s
=> => naming to docker.io/library/servlet-app 0.0s

```

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

Para desplegar el entorno Docker de manera sencilla hemos creado un docker-compose:


```

🐳 docker-compose.yml ●
D: > Program Files > OneDrive > Universidad > 4º Año > 1º Cuatrime
1  version: '3.1'
2
3  services:
4
5      db:
6          image: mariadb:10.9.4
7          restart: always
8          volumes:
9              - ./mariadb-vol-hashed:/var/lib/mysql
10         environment:
11             - MARIADB_ROOT_PASSWORD=root
12         ports:
13             - 3306:3306
14         container_name: mariadb-cont
15
16     servlet-app:
17         image: servlet-app:latest
18         restart: always
19         ports:
20             - 8080:8080
21         depends_on:
22             - db
23         container_name: servlet-app-cont
24
25     adminer:
26         image: adminer:4.8.1
27         restart: always
28         ports:
29             - 8081:8080
30         depends_on:
31             - db
32         container_name: adminer-cont
33

```

El entorno docker cuenta con la base de datos mariadb, la cual tiene persistencia de datos gracias al volumen “mariadb-vol” que se encuentra en la carpeta setup Docker adjuntada.

A parte disponemos de un contenedor con la imagen “adminer” el cual nos permite gestionar la base de datos de manera visual con una interfaz muy intuitiva:

Idioma: Español

MySQL » db » Base de datos: gestionPracticas

Adminer 4.8.1

DB: gestionPracticas

[Comando SQL](#) [Importar](#)
[Exportar](#) [Crear tabla](#)

[registros alumno](#)
[registros empresa](#)
[registros informe_practicas](#)
[registros oferta_practicas](#)
[registros oferta_practicas_has_alun](#)
[registros persona](#)
[registros responsable](#)
[registros tutor](#)

Base de datos: gestionPracticas

[Modificar Base de datos](#) [Esquema de base de datos](#) [Privilegios](#)

Tablas y vistas

Buscar datos en tablas (8)

[Condición](#)

<input type="checkbox"/>	Tabla	Motor [?]	Colación [?]	Longitud de datos [?]	Longitud de índice [?]	Espacio libre [?]	Incremento automático [?]	Registros [?]	Comentario [?]
<input type="checkbox"/>	alumno	InnoDB	utf8mb4_general_ci	16 384	0	0		~ 1	
<input type="checkbox"/>	empresa	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
<input type="checkbox"/>	informe_practicas	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
<input type="checkbox"/>	oferta_practicas	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
<input type="checkbox"/>	oferta_practicas_has_alumno	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
<input type="checkbox"/>	persona	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
<input type="checkbox"/>	responsable	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
<input type="checkbox"/>	tutor	InnoDB	utf8mb4_general_ci	16 384	0	0		0	
	8 en total	InnoDB	utf8mb4_general_ci	131 072	0	0			

Selected (0)

[Analizar](#) [Optimizar](#) [Comprobar](#) [Reparar](#) [Vaciar](#) [Eliminar](#)

Mover a otra base de datos: gestionPracticas [Mover](#) [Copiar](#) ☐ overwrite

[Crear tabla](#) [Crear vista](#)

Procedimientos

[Crear procedimiento](#) [Crear función](#)

Eventos

[Crear Evento](#)

Se accede desde el puerto 8081 en localhost.