TRABAJO FIN DE GRADO

GS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Sistema para fichajes de empleados mediante Arduino, y aplicación web en Angular + API REST para la gestión y visualización de los mismos.

Alumno: Daniel Jiménez Gutiérrez
Tutor: Rafael Juan Alamañac Garrido

Calahorra, junio, 2022

ÍNDICE

INTRODUCCIÓN

Uno de los software más necesarios y utilizados son los sistemas de fichaje de las empresas. En el mundo en el que vivimos cada vez existen más negocios e industrias, y resulta clave tener una forma de gestionar la actividad de los trabajadores.

Desconocer el periodo de tiempo al que asisten los empleados es un gran problema, pues no tendríamos forma de saber si se cumplen el contrato y horario pactados.

A pesar de tener muchas soluciones en el mercado actualmente, todas ellas cuentan con interfaces poco intuitivas o con pocas funcionalidades, siendo algo repetitivas. Además, requieren instalar el software en cuestión, y no son flexibles en cuanto a sistemas operativos, por lo que no puedes consultar información relativa a los fichajes desde cualquier dispositivo.

La finalidad de este proyecto es resolver este planteamiento mediante una interfaz novedosa y moderna, que se aleja de las interfaces antiguas a las que estamos acostumbrados. Esto será realizado mediante una página web, proporcionando mayor portabilidad, pues no va a estar limitada a ningún sistema operativo, y sin requerir instalación previa.

AGRADECIMIENTOS

No puedo dar por finalizado este proyecto sin agradecer de corazón a todas aquellas personas que se lo merecen.

A mi tutor, Rafael Juan Alamañac Garrido, por haberme apoyado y aconsejado siempre que lo he necesitado, y, sobre todo, por motivarme a dar lo mejor de mí y atreverme a proponerme nuevas metas.

A mi familia, por su apoyo incondicional y comprensión infinita.

A mis amigos, especialmente a Daniel Gil, por haber estado siempre ahí cuando los he necesitado.

OBJETIVOS

En este apartado se definirán los objetivos a cumplir durante el desarrollo de este proyecto. La meta es diseñar y crear un sistema que permita llevar el control del horario de los trabajadores. Para ello, se deberá:

- Montar un sistema que pueda leer tarjetas.
    1. Preparar el NodeMCU V3 (Arduino) y RFID RC522 (lector), para que al pasar una tarjeta pueda leer su información.

- Preparar una base de datos para guardar información de empleados y fichajes.
- Preparar API para poder manipular los datos de la base de datos.

- Registrar horas de inicio y fin de jornada
    1. Guardar ID de cada trabajador en una tarjeta distinta.
    2. Al pasar la tarjeta por el lector, detectar la ID del trabajador y enviarla a la base de datos (mediante API) junto a la fecha en la que ha fichado.

- Preparar página web para mostrar la información de la base de datos
    1. Crear inicio de sesión, validando mediante JWT (diferenciar entre usuario y administrador, ofreciendo diferentes funcionalidades/opciones).
    2. Crear apariencia página web y mostrar inicios de sesión mediante calendario y/u otros métodos.

DEFINICIÓN DEL PROBLEMA

Cualquier empresa, independientemente del número de trabajadores que posea en plantilla, necesita llevar un control del trabajo que realiza cada uno de los miembros que pertenecen a la misma.

El hecho de que los trabajadores fichen, y que esta información quede guardada para poder comprobar el horario que han realizado, cumple este propósito.

CONCRECIÓN DE CIERTOS ASPECTOS ABIERTOS AL PROBLEMA

Realizando un estudio de mercado de los sistemas de fichaje que se utilizan hoy en día podemos descubrir las fortalezas y debilidades que tiene nuestro software respecto a la competencia.

Prácticamente todos los sistemas de fichaje son soluciones de software, lo que significa que no tienen una portabilidad tan alta como una página web, ya que puede ser que no estén operativos o disponibles para algún sistema operativo.

Mediante una página web solucionamos este problema, ya que es accesible desde cualquier navegador y dispositivo.

Un problema con el que nos encontramos podría ser la velocidad. Un programa (software) puede cargar datos rápidamente. En el caso de la página web, vamos a tener que garantizar y conseguir una buena velocidad de carga de datos, ya que en caso contrario la experiencia podría ser más negativa que las alternativas existentes.

Utilizando Angular para diseñar nuestra página web parece una buena solución, puesto que permite crear páginas web con una rápida respuesta, justo lo que necesitamos. Podemos llegar a conseguir bastante fluidez de carga, lo que va a hacer que el usuario sienta más satisfacción al consultar nuestra web.

ESTIMACIÓN DE TIEMPOS INICIAL

- Preparación hardware (planteamiento circuito, montaje con soldadura componentes preparación Arduino y tarjetas (grabar IDs empleados): 5 horas
- Diseño y creación de la base de datos: 2 horas
- Diseño y creación de la API: 9 horas
- Diseño y creación página web hecha con Angular: 16 horas
- Testing/pruebas: 2 horas
- Documentación/memoria: 3 horas
- Preparación presentación (PowerPoint): 2 horas
- Preparación entorno para aplicación: 1 horas

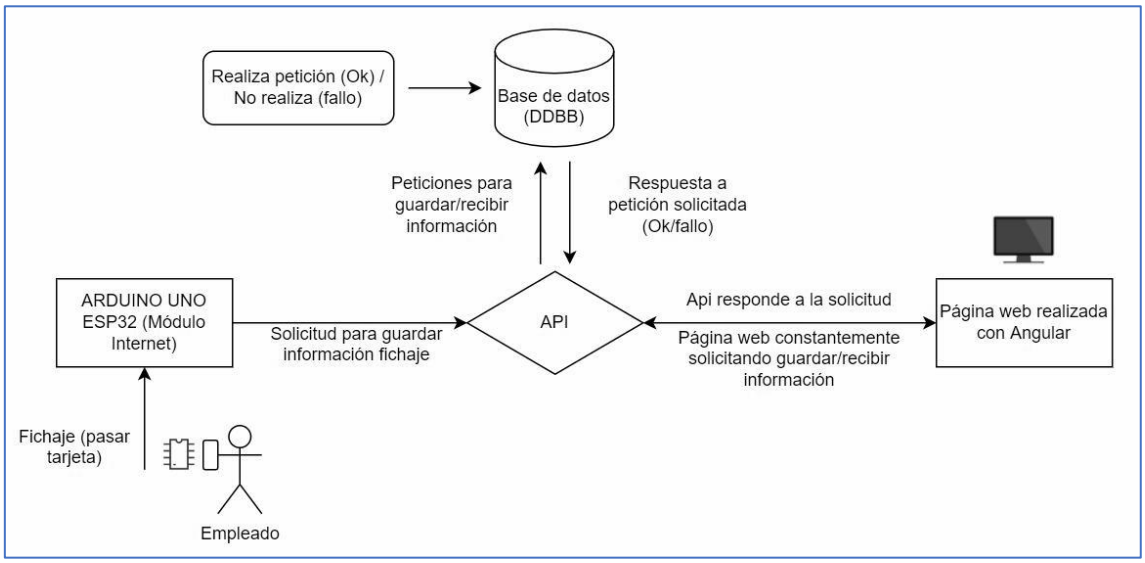
Tiempo total: 40 horas

DISEÑO, COMPONENTES Y TECNOLOGÍAS DE USO

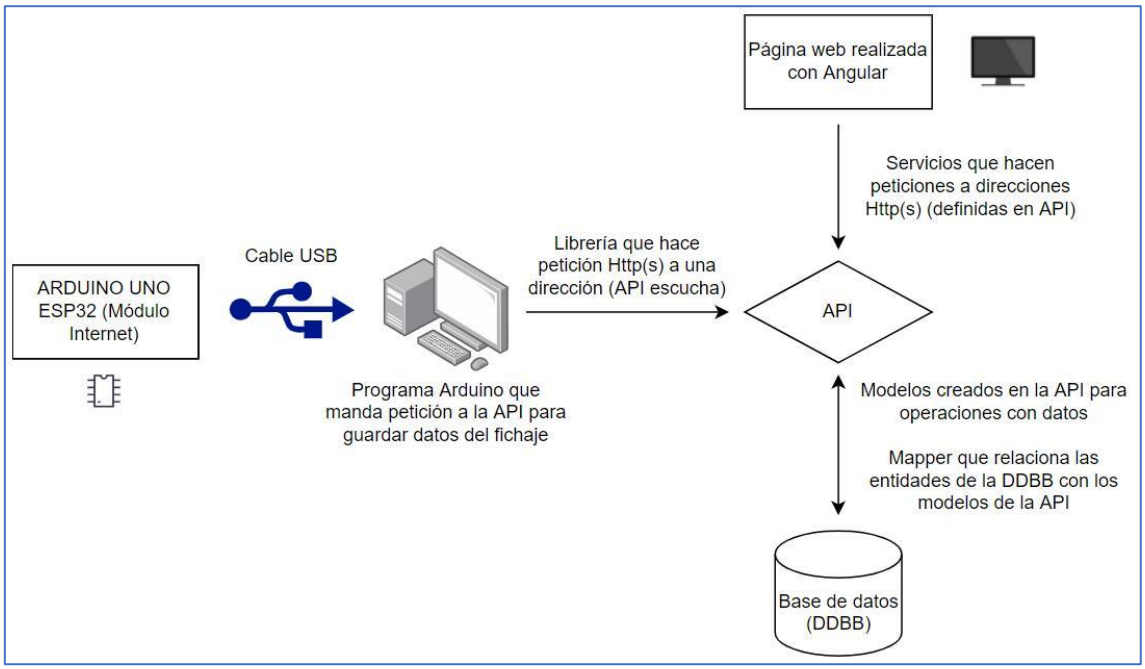
El proyecto es llevado a cabo con el uso de las siguientes tecnologías y componentes:

- Hardware/electrónica:
  - NodeMCU Lolin V3 Módulo ESP8266 ESP-12F (Arduino)
  - Lector de tarjetas RFID RC522 + Tarjetas con IDs grabadas
  - Cables (Jumper Wires)
  - Protoboard (unir elementos y formar esquema)
  - Cable UART-USB para conectar Arduino a PC
- Software utilizado por hardware/Arduino (archivos .ino):
  - Arduino IDE (entorno de programación)
    - Programa para leer información tarjeta
    - Programa para grabar ID de empleados en tarjetas
    - Programa para leer ID de empleado de tarjeta
    - Programa para hacer POST de la ID de la tarjeta (empleado) a la API, utilizando librerías para conexión a Internet
- Base de datos (Imagen de SQL Server dockerizada) v.2019 - latest
  - Entidades para guardar la información relativa a empleados y fichajes
- API (realizada en .NET 6):
  - Validación de inicio de sesión de la página web mediante JWT
  - Documentación mediante Swagger
- Página web (Angular 13.1.4)
  - Inicio de sesión validando mediante Token -> cifrado Jwt
- Git (control de versiones, y backup, en caso de que cometamos algún error que queramos deshacer)

En el siguiente esquema podemos ver la función que desempeña cada uno, y cómo fluye el flujo de trabajo de la aplicación:



Como vemos, la API actúa como núcleo para comunicar al Arduino y página web con la base de datos. El anterior esquema muestra las funciones; si queremos saber cómo se comunican podemos prestar atención a este otro diagrama:



Esto nos lleva a pensar el porqué de usar estas tecnologías y no otras. A la hora de diseñar el sistema me he basado en los siguientes razonamientos:

ESCOGIENDO EL HARDWARE

Lo primero: buscamos simular un sistema de fichaje. Una de las mejores opciones es utilizar un lector RFID RC522, por varios motivos: es potente, lleva usándose mucho tiempo, y ha sido utilizado mucho tiempo, por lo que existen muchas guías para cualquier proyecto.

Este lector debe ir conectado a un Arduino, para poder procesar los datos que se obtienen al leer la tarjeta. Sin embargo, no podemos realizar la petición a la API, ya que para ello necesitamos bien librerías para conectarlo por WIFI o algún módulo adicional. En nuestro caso vamos a proceder con esta última opción, gracias al módulo ESP32.

ESCOGIENDO EL SOFTWARE

Hemos de decidir la base de datos que vamos a utilizar. Ya que tengo más experiencia con bases de datos relacionales voy a escoger SQL Server, que encaja en este perfil, y podemos obtener fácilmente una imagen dockerizada.

En cuanto a la API y página web: ya que he realizado algún proyecto que mezcla página web hecha con Angular y API hecha con .net, tengo claro cómo voy a proceder con esta parte.

A esto sumamos que Angular es una maravilla, ya que permite crear páginas web muy rápidas y de una forma muy estructurada, a la vez que podemos incorporar funcionalidades con TypeScript de una forma muy práctica.

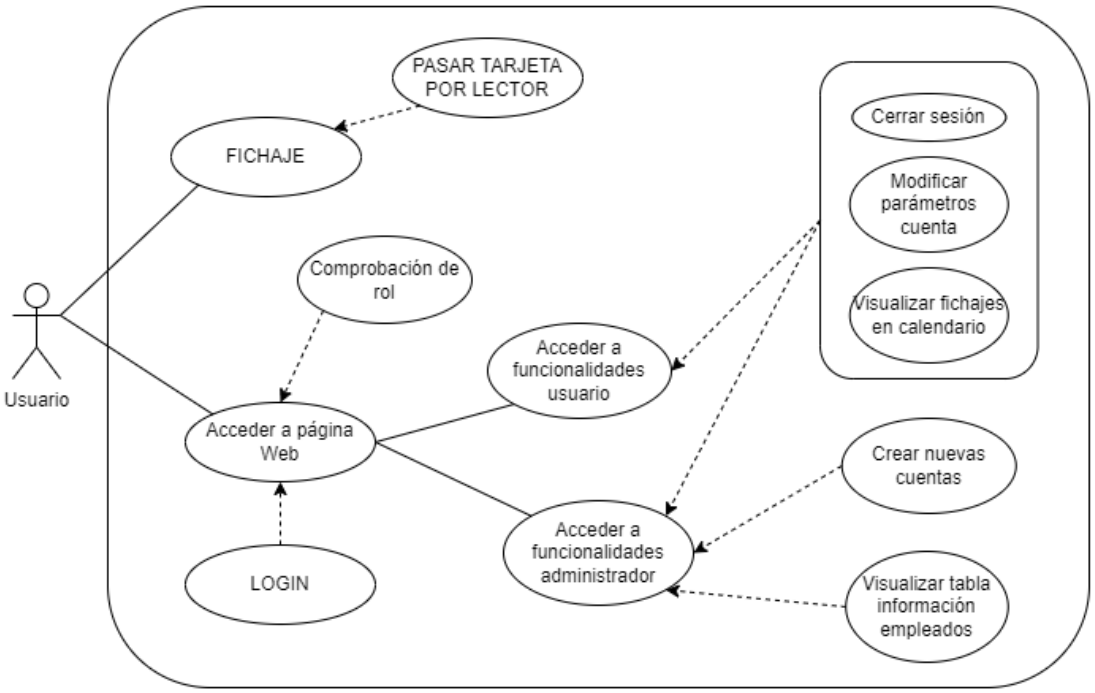
FASES DEL TRABAJO

ANÁLISIS

En este apartado se describe el análisis de los requisitos del proyecto.

1. DIAGRAMA DE CASOS DE USO

Nuestro sistema va a tener dos perfiles o roles: administrador y usuario. A continuación, se muestran las funcionalidades a las que puede acceder cada uno de ellos:



2. ESPECIFICACIÓN DE CASOS DE USO

En este apartado se desarrollarán los casos de uso descritos en el esquema anterior.

Caso de uso	Fichaje
Actor	Usuario, Administrador (Empleados)
Propósito	Registrar el fichaje del empleado en la BBDD
Pre-condiciones	Poseer tarjeta con ID registrada en la base de datos (perteneciente a un empleado de la empresa)
Flujo básico: 1. El actor pasa la tarjeta por delante del lector RFID RC522 2. El lector lee el espacio de memoria de la tarjeta donde está guardada la ID 3. El módulo NodeMCU manda petición POST a la base de datos, guardando el fichaje	
Flujos alternos: 1. El lector no lee ninguna ID: no se realiza ningún POST a la base de datos 2. Fallo del lector de tarjetas (en principio no debería pasar)	

Caso de uso	Login
Actor	Usuario, Administrador (Empleados)
Propósito	Acceder al contenido de la página web
Pre-condiciones	Poseer una cuenta creada (credenciales de inicio de sesión). No tener iniciada sesión (no podríamos acceder al login).
Flujo básico: 1. El actor introduce usuario y contraseña y hace click en iniciar sesión 2. La API manda los datos a la base de datos 3. Se comprueba si las credenciales son correctas 4. Se inicia sesión (crea token) 5. Se redirige al primer componente dentro de la página web	
Flujos alternos: 1. Las credenciales no son correctas (usuario, contraseña o ambos) 2. Se produce un aviso de ello 3. El formulario se resetea (se vacían los campos) 4. No se inicia sesión, se deberá volver a intentar el proceso	

Caso de uso	Cerrar sesión
Actor	Usuario, Administrador (Empleados)
Propósito	Cerrar sesión en la página web
Pre-condiciones	Poseer una cuenta creada (credenciales de inicio de sesión). Haber iniciado sesión (no estaríamos dentro de la página).
Flujo básico: 1. Hacer click en icono de cuenta (parte derecha del encabezado), y de nuevo click en la opción 'cerrar sesión' 2. Se cerrará sesión (se borrará token) y se redireccionará al login.	

Caso de uso	Modificar parámetros cuenta
Actor	Usuario, Administrador (Empleados)
Propósito	Actualizar domicilio (localidad) o puesto de trabajo
Pre-condiciones	Tener sesión iniciada
Flujo básico: 1. Acceder a página 'cuenta', haciendo click en icono de cuenta (parte derecha del encabezado), y de nuevo click en la opción 'cuenta' 2. El actor hace click en los iconos de los inputs 'domicilio' o 'puesto de trabajo' 3. Los campos se vuelven editables 4. El actor realiza las modificaciones deseadas 5. El actor hace click en el botón de confirmar modificaciones 6. Se hace una petición PATCH a la base de datos, y se actualiza la información de la cuenta	
Flujos alternos: 1. El actor hace click en el botón de confirmar modificaciones sin haber realizado cambios 2. Se detecta que no se han realizado cambios y se avisará de ello 3. No se realiza ninguna petición a la API para que actualice los datos	

Caso de uso	Visualizar los fichajes en el calendario
Actor	Usuario, Administrador (Empleados)
Propósito	Consultar los fichajes realizados (Usuario verá propios, y Administrador los de toda la plantilla)
Pre-condiciones	Tener sesión iniciada

Flujo básico:

1. Acceder a página 'calendario', haciendo click en dicha opción, en el menú superior central
2. Se mostrará un calendario, con los fichajes realizados por el empleado (en caso de ser usuario) o con los fichajes de todos los empleados (siendo administrador)
    - El actor podrá cambiar la visualización haciendo click en las opciones de 'mensual'/'semanal'/'diario'
    - El actor podrá moverse a través de los meses, semanas o días (dependiendo de la opción actual) haciendo click en las opciones 'siguiente'/'actual'/'anterior'
    - El actor podrá hacer click en un día con fichajes para que se muestre un desplegable con la hora de los mismos, y el empleado que ha fichado
    - El actor podrá pasar el ratón por encima de los fichajes para mostrar información relativa a los mismos

Caso de uso	Crear nuevas cuentas
Actor	Administrador (Empleado)
Propósito	Habilitar cuenta y credenciales para que nuevo empleado pueda acceder a la página web
Pre-condiciones	Tener sesión iniciada y ser administrador

Flujo básico:

1. Acceder a página 'nueva cuenta', haciendo click en icono de cuenta (parte derecha del encabezado), y de nuevo click en la opción 'nueva cuenta'
2. El actor introduce todos los datos en los inputs del formulario
3. El actor hace click en el botón de 'crear nueva cuenta'
4. Se realiza una petición POST para guardar la cuenta en la base de datos
5. Se muestra un aviso de que se ha creado correctamente, junto al botón

Flujos alternos:

1. El campo de contraseña y repetir contraseña no coinciden, y se pulsa el botón de crear (no se realiza POST y se avisa del error)
2. El email ya está siendo utilizado por otro usuario (asignado a otra cuenta): no se realiza el POST y se avisa del error
3. El DNI ya pertenece a otro empleado/cuenta: no se realiza el POST y se avisará del error
4. El rol introducido no es 'Admin' ni 'User': no se realiza el POST y se avisará del error

Caso de uso	Visualizar tabla empleados
Actor	Administrador (Empleado)
Propósito	Consultar información de todos los empleados de la plantilla de la empresa
Pre-condiciones	Tener sesión iniciada y ser administrador

Flujo básico:

1. Acceder a página 'empleados, haciendo click en dicha opción, en el menú superior central
2. Se mostrará una tabla, con la información de todos los empleados de la empresa
    - El actor podrá hacer click en el botón superior 'Información' para que se muestre un desplegable con una guía de las funcionalidades de la tabla
    - El actor podrá escribir en el buscador para buscar por valores específicos en la tabla, tanto numéricos como cadena (se filtrarán los registros que tengan cualquier atributo conteniendo los valores buscados)
    - El actor podrá elegir el número de registros a mostrar, en múltiplos de 5, y el número de página actual, haciendo click en los botones situados justo encima del encabezado de la tabla

DISEÑO DE LA APLICACIÓN E IMPLEMENTACIÓN

En este apartado se expondrá el diseño de todos los componentes por los que está formada la aplicación, detallando cómo funciona cada uno de ellos y su estructura (archivos que las componen y funcionalidades que aportan).

Vamos a comenzar nombrando las funcionalidades que queremos que implemente nuestra página web, para posteriormente dar paso a la introducción de cada uno de los componentes.

1. PLANTEAMIENTO DE FUNCIONALIDADES

Nuestra página web está dedicada al fichaje de los empleados de las empresas. Los usuarios deberán iniciar sesión, ya que la finalidad principal es poder consultar tus propios fichajes y cuenta personal. No tiene sentido poder ver ningún tipo de información sin haber pasado por este proceso.

Para este inicio de sesión se definirán dos tipos de perfiles: usuarios y administradores. Vamos a tener diferentes funcionalidades dependiendo del rol actual que tengamos:

- Comunes a ambos roles (Admin y User)
    - Inicio y cierre de sesión
    - Gestión de cuenta (visualizar y cambiar datos de propia cuenta)
    - Calendario (ver los fichajes que se han realizado). El administrador podrá ver los fichajes relativos a todos los usuarios. Cada empleado (user) estará limitado a ver sus propios fichajes, por privacidad.
    - Overview (vista general de los fichajes con diferentes gráficas y funcionalidades). Al igual que el anterior, el Admin verá todos, cada usuario sus propios parámetros.
- Exclusivos de Admin
    - Crear nueva cuenta (registrar a un nuevo empleado para que pueda acceder a la página web)
    - Consultar información de todos los empleados de la plantilla, con las siguientes funcionalidades:

- Buscador por cualquier campo (numérico o texto)
- Ordenación ascendente y descendente por cualquier campo
- Diferentes opciones para mostrar los datos (x números por página)

Una vez tenemos claros los requisitos que debe cumplir, podemos pasar a diseñar los componentes. Comenzaremos diseñando la base de datos.

2. BASE DE DATOS

Vamos a tener que guardar información tanto de los empleados como de los fichajes. Analizando parámetros que considero importantes para mostrar en la página web, he llegado a la conclusión de que necesitaría almacenar lo siguiente:

- Empleados: Id del empleado, nombre, edad, dirección, puesto, dni, correo, contraseña cifrada y rol.
- Fichajes: Id del fichaje, Id del empleado que realiza el fichaje y fecha y hora en la que se realiza.

La contraseña hasheada (a partir de ella no es posible obtener la contraseña original) es debido a que necesitamos guardar las credenciales de inicio de sesión (usuario y contraseña), pero no es seguro guardar las mismas directamente en la base de datos. Es por esto que guardaremos la contraseña hasheada, para una mayor seguridad.

En caso de que alguien en algún caso pudiera llegar a visualizar las contraseñas, las vería cifradas, lo que haría más difícil el hecho de robar o acceder a cuentas ajenas.

Para guardar esta información, vamos a tener dos entidades (empleado y fichaje), con los mismos campos que los atributos nombrados anteriormente, de los cuales queremos guardar información.

Respecto a la estructura de base de datos (utilizando SQL Server):

La relación será de 1 a muchos (un empleado tiene muchos fichajes, y cada fichaje pertenece a un solo empleado). Por ser una relación 1:N, no se creará una tabla adicional. En la entidad fichaje tendremos un campo que haga referencia a la Id de empleado.

- Entidad Empleado:

```
CREATE TABLE Empleado(
Id INTEGER IDENTITY(1,1),
Nombre VARCHAR(200),
Edad INTEGER,
Direccion VARCHAR(150),
Puesto VARCHAR(50),
Dni VARCHAR(9) UNIQUE,
Correo VARCHAR(100) UNIQUE,
HashPassword VARCHAR(MAX),
Rol BIT,
CONSTRAINT Pk_Empleado PRIMARY KEY(Id)
);
```

Id será la clave primaria. La base de datos se encargará de aumentar la ID en una unidad cada vez que creamos un nuevo registro. Los demás campos son cadena (texto) o numéricos, excepto el rol, que será bit (valor 1 para Administrador / valor 0 para Usuario).

El DNI y correo no van a poder repetirse, ya que no vamos a tener dos empleados con el mismo DNI, ni dos cuentas con el mismo correo asociado.

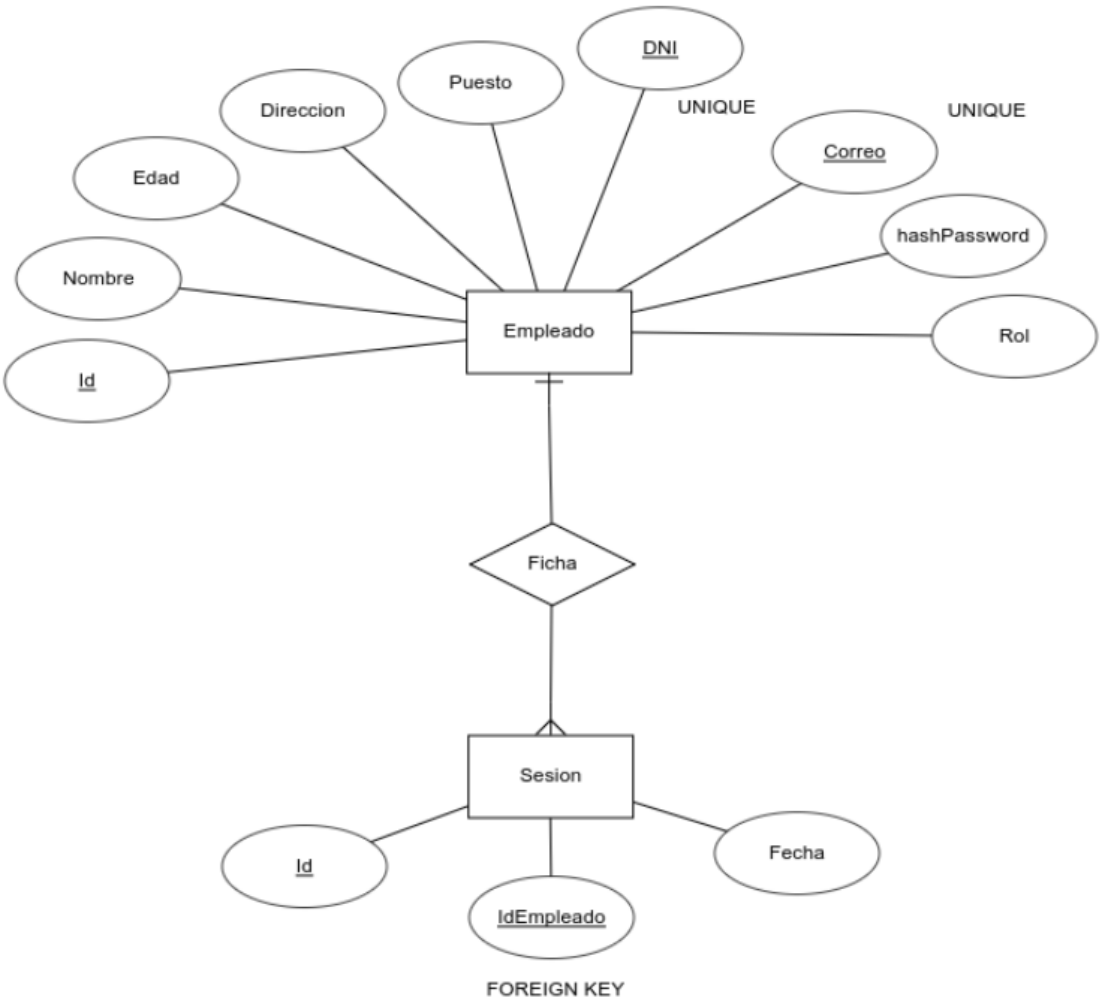
- Entidad Fichaje:

```
CREATE TABLE Sesion(
Id INTEGER IDENTITY(1,1),
IdEmpleado INTEGER,
Fecha DATETIME,
CONSTRAINT Pk_Sesion PRIMARY KEY(Id),
CONSTRAINT Fk_Empleado_Sesion FOREIGN KEY (IdEmpleado) REFERENCES Empleado(Id)
ON UPDATE CASCADE ON DELETE CASCADE
);
```

En este caso es prácticamente lo mismo que para la entidad anterior. Lo único nuevo es que utilizamos un campo DATETIME para Fecha (lo que nos permite almacenar tanto fecha como hora al mismo tiempo), y que especificamos que la IdEmpleado de esta tabla hace referencia a la Id de la tabla anterior.

Esto es porque cada fichaje (sesión) va a ser realizada por un empleado ya existente. En caso de que se borre el empleado se borrarán sus fichajes (ON DELETE CASCADE), y en caso que se actualizara su Id (en teoría no va a pasar, pero por si acaso), se actualizará también en cada uno de sus fichajes.

Esquema general:



3. API

La función principal de este componente es actuar como intermediario, para permitir a la web recibir y enviar datos a la base de datos, y también al Arduino enviar la información y guardarla en la DDBB cuando se realiza un fichaje.

La API es RESTful y está realizada mediante .Net6.

3.1) MODELOS

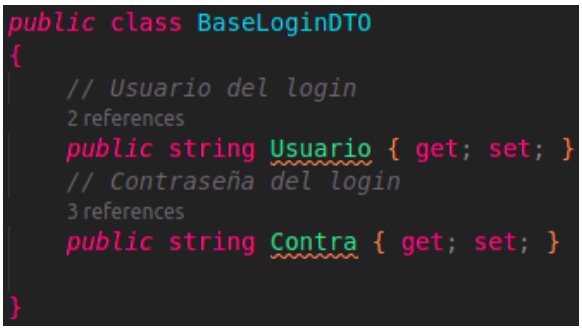
Los modelos son objetos que nos sirven para tratar con la información una vez se han modificado en la base de datos.

Digamos que tanto la API como la WEB tienen modelos de datos. La API relacionará las entidades de la base de datos con sus modelos. La web enlazará estos últimos con sus propios modelos. De esta forma se va formando una cadena de objetos para mostrar su información (atributos) donde queramos.

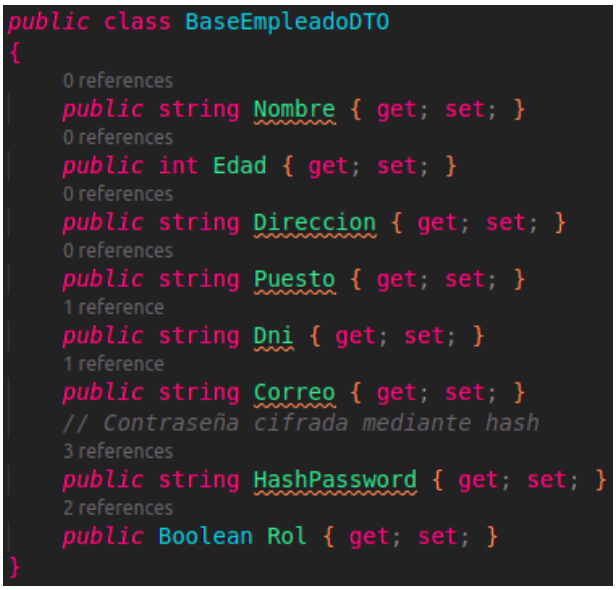
Por ejemplo, tenemos un EmpleadoDTO (modelo), para que, al hacer una petición de recibir un empleado en la página web, pueda recibir dicho objeto y mostrar su nombre, id…etc.

Los modelos de la API que vamos a tener son los siguientes:

- LoginDTO (para tratar con la información introducida en el Login/inicio de sesión)

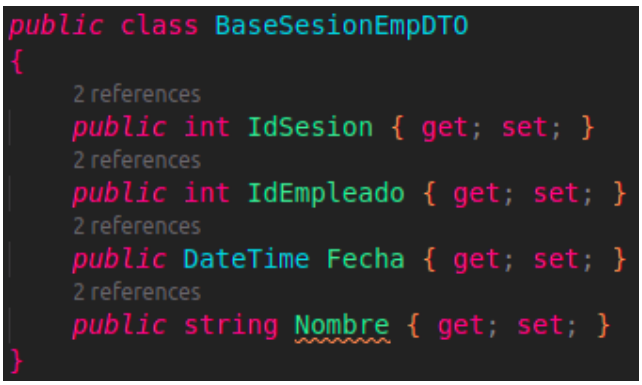


- EmpleadoDTO (para tratar con la información de los empleados)



- SesionEmpDTO (muestra información tanto de fichajes: IdSesion, IdEmpleado, Fecha, como de empleados: Nombre).

  Este modelo existe porque en la página web no voy a mostrar solamente información de fichajes, si no que vamos a mostrar cada fichaje junto a información del empleado que la ha realizado.



3.2) SERVICIOS

Los servicios de la API estarán configurados para poder añadir, consultar o modificar datos de la DDBB. Tendremos diferentes operaciones para cada uno de los modelos de datos que utilicemos.

Los servicios serán los siguientes:

- EmpleadoService:

```
public interface IEmpleadoService
{
    // Obtener todos los empleados
    1 reference
    public IEnumerable<EmpleadoDTO> GetAll();

    // Obtener empleado con una Id específica
    1 reference
    public EmpleadoDTO GetByID(int guid);

    // Obtener empleado con un correo específico
    3 references
    public EmpleadoDTO GetByUser(string email);

    // Comprueba si el email ya está siendo utilizado
    1 reference
    public Boolean EmailExists(string email);

    // Comprueba si el dni pertenece a alguien ya registrado
    1 reference
    public Boolean DniExists(string dni);

    // Añade un empleado a la base de datos
    1 reference
    public EmpleadoDTO Add(BaseEmpleadoDTO guid);

    // Modificar los datos de un empleado
    1 reference
    public EmpleadoDTO Modify(BaseEmpleadoDTO Empleado, int guid);
}
```

- SesionEmpService:

```
public interface ISesionEmpService
{
    // Devuelve todos los fichajes del empleado con esa Id
    1 reference
    public IEnumerable<SesionEmpDTO> GetByIdSesion(int id);

    // Devuelve todos los fichajes
    1 reference
    public IEnumerable<SesionEmpDTO> GetAllSesionEmp();
}
```

En cada método se muestra comentada la función que desempeña.

Tenemos estos métodos, con los que la página puede manipular la información de la base de datos a través de la API, tenemos que preparar una forma de que la web sea capaz de llamarlos.

Establecemos los endpoints en los controladores.

3.3) CONTROLADORES

Vamos a tener tres controladores. En la siguiente tabla podemos ver los endpoints de cada uno de ellos y métodos a los que llamamos al hacer peticiones a los mismos.

EmpleadoController (https://localhost:7199/empleados/)

endpoint	tipo request	método ejecutado	comprobaciones
	GET	GetAll	Solicitante es Admin
GetByEmail/{Email}	GET	GetByUser	Si solicitante es usuario, no solicitar id ajena
	POST	Add	Email y DNI no utilizados por otro empleado Solicitante es Admin
update/{Id}	PATCH	Modify	Si solicitante es usuario, no actualizar id ajena

SesionController (https://localhost:7199/sesiones/)

endpoint	tipo request	método ejecutado	comprobaciones
sesionemp	GET	GetAllSesionEmp	Solicitante es Admin
sesionEmp/{IdEmp}	GET	GetByIdSesion	Si solicitante es usuario, no solicitar id ajena

LoginController (https://localhost:7199/api/)

endpoint	método ejecutado
login	generar y devolver token

En este último caso (LoginController), al llamar al método login y pasarle usuario y contraseña (tipo modelo LoginDTO), si las credenciales son correctas se creará un Token, utilizado posteriormente por la autenticación (siguiente apartado). Si las credenciales no son correctas el token no se generará, y se avisará de credenciales no válidas.

Este token contendrá en su interior el usuario que acaba de iniciar sesión, cifrado (email), y estará preparado para estar activo durante 24h (tras las cuales se cerrará sesión).

Los métodos con fondo marcado en las tablas anteriores requieren que exista un token para mostrar resultados (es decir, sesión iniciada). En caso de que no exista el token, no van a proporcionar la información o llevar a cabo la petición. Estos métodos se encuentran marcados mediante el atributo [Authorize], en los controladores.

3.4) SEGURIDAD Y AUTENTICACIÓN

La seguridad es uno de los aspectos más importantes en cualquier aplicación. Si van a existir cuentas de usuario hay que garantizar alguna forma para que un usuario no sea capaz de acceder a los datos de otro, pudiendo ver su información privada o incluso modificarla.

Cualquier persona con ciertos conocimientos de informática tendría varias formas de violar la seguridad de nuestra página. Hay que tratar varios aspectos. Ahora vamos a comenzar con la autenticación.

Vamos a preparar una serie de servicios/embudos que nos permitan verificar que el usuario ha iniciado sesión correctamente, y, en caso de que no haya sesión iniciada, (con credenciales correctas) no se permitirá acceder a la información.

3.4.1) VALIDACIÓN MEDIANTE JWT

- JwtMiddleware: es un servicio que actúa como embudo. Todas las peticiones que se hagan a la API van a pasar por él. Este archivo comprueba si hay token presente, y en ese caso recoge la información que contenía (email) e intenta serializarlo en un objeto.
- AuthorizeAttribute: servicio por el que pasan las peticiones marcadas con [Authorize] (después del MiddleWare). Intenta deserializar el objeto nombrado en el servicio anterior.
    - Si consigue deserializarlo y obtenerlo significa que el token existe, y por lo tanto se ha iniciado sesión correctamente. En este caso, dicha petición marcada continúa (autorizada).
    - Si no consigue deserializarlo, la petición no continuará, y se muestra un mensaje ‘no autorizado’.

3.4.2) COMPROBACIONES ADICIONALES

- EmpleadoCheck: vamos a tener dos filtros
    - isAdmin: comprueba que el usuario que hace la petición es administrador
    - isSameUser: comprueba que el usuario que hace la petición no intenta manipular información de otra id (otro usuario), a no ser que sea administrador (en tal caso puede acceder a la información de todos).

Estos son los dos aspectos relevantes de seguridad en el lado de la API. Con esto conseguimos que solo se pueda acceder a la información si se inicia sesión y que solo puedas ver lo que tengas permitido, no datos de otras personas.

En la web, no obstante, también trataremos varios aspectos de seguridad, de los que hablaremos a su debido momento.

4. PÁGINA WEB

Ha llegado el momento de hablar de la interfaz que va a ver el usuario, la parte visible de nuestra aplicación.

Esta página web ha sido realizada utilizando Angular 13.1.4, y para poder entender cómo está estructurada y las opciones que aporta al usuario, considero oportuno comenzar hablando por el back. Una vez comprendamos los procesos que tienen lugar, podremos entender con más idea la apariencia de la misma.

4.1) SERVICIOS

4.1.1) SERVICIOS PETICIONES API

Prácticamente toda la información que se va a mostrar en nuestra página va a venir de la base de datos.

En el apartado anterior hemos visto los endpoints que tenemos en nuestra API, por lo que necesitamos preparar servicios en la web que hagan peticiones a dichos endpoints para tratar con la información de la base de datos. De esta forma podremos mostrar la información que recibamos de las peticiones a la API, mandar la que necesitemos o manipularla.

- Empleado.Service:

```
getEmpleadoData(): Observable<any> {
  return this.http.get(environment.API_URL + 'empleados', {});
}

getUserEmpleadoData(email: string): Observable<Empleado> {
  return this.http.get<Empleado>(
    environment.API_URL + 'empleados/GetByEmail/' + email
  );
}

modifyEmpleadoData(datosFormAccount: any, id: number): Observable<Empleado> {
  return this.http.patch<Empleado>(
    environment.API_URL + 'empleados/update/' + id,
    datosFormAccount
  );
}

// TODO
postEmpleadoData<T>(body: any): Observable<HttpResponse<T>> {
  let bodyData = new Empleado();
  bodyData.nombre = body.Nombre;
  bodyData.edad = body.Edad;
  bodyData.direccion = body.Direccion;
  bodyData.puesto = body.Puesto;
  bodyData.dni = body.Dni;
  bodyData.correo = body.Email;
  // Pasamos la contraseña y la ciframos en el controller
  bodyData.hashPassword = body.Pass;
  if (body.Rol == 'Admin') {
    bodyData.rol = true;
  } else if (body.Rol == 'User') {
    bodyData.rol = false;
  }

  return this.http.post<T>(environment.API_URL + 'empleados', bodyData, {
    observe: 'response',
  });
}
}
```

Tal y como podemos ver, los métodos en verde en cada return (get,patch,post) hacen referencia a las operaciones descritas en los endpoints anteriormente nombrados, y en amarillo tenemos las rutas http a las que vamos a realizar las peticiones (es decir, las definidas en los controladores).

Si desde nuestra web llamamos a cualquiera de estos métodos, hará una petición a la API, que a su vez hará de intermediario con la base de datos solicitando obtener/enviar o modificar datos, y devolviendo a la web la respuesta de la petición (correcta/fallida…etc.).

- Sesion.Service:

```
export class SesionService {
  constructor(private http: HttpClient) {}

  // SesionEmp of specific User (used for calendar User)
  getSesionsOfEmp(idEmp: number): Observable<SesionEmp[]> {
    return this.http.get<SesionEmp[]>(
      environment.API_URL + 'sesiones/sesionEmp/' + idEmp
    );
  }

  // All SesionEmp (used for calendar Admin)
  getSesionsEmp(): Observable<SesionEmp[]> {
    return this.http.get<SesionEmp[]>(
      environment.API_URL + 'sesiones/sesionEmp'
    );
  }
}
```

- Login.Service:

```
postLoginData<T>(login: any): Observable<HttpResponse<T>> {
  const httpHeaders: HttpHeaders = this.getHeaders();
  return this.http.post<T>(environment.API_URL + 'api/login', login, {
    headers: httpHeaders,
    observe: 'response',
  });
}

public getHeaders(): HttpHeaders {
  let httpHeaders: HttpHeaders = new HttpHeaders();
  return httpHeaders;
}
```

Si nos fijamos, todos estos métodos coinciden con los definidos en los controladores de la API. Todos ellos hacen referencia a las direcciones https que ya teníamos establecidas.

4.1.2) SERVICIOS PARA GESTIONAR TOKEN / SESIÓN INICIADA

Anteriormente se ha explicado que, al hacer login correctamente, se generará un token. Este token se guardará en las cookies de nuestra página web, y se mandará en cualquier petición futura, para que la API permita que se lleve a cabo, autenticando el proceso.

Si nos fijamos en el método 'postLoginData' de la última imagen podemos ver que espera una respuesta (observe response). Una vez se realice recibiremos el token, creado en la API, el cual manipularemos mediante los siguientes métodos (definidos en un servicio):

```
constructor(private cookie: CookieService) {}

// HANDLE COOKIES
// 1. Create cookie
public setCookie(token: string) {
  this.cookie.set('X-Token', token);
}

// 2. Get cookie
public getCookie() {
  return this.cookie.get('X-Token');
}

// 3. Delete cookie
public closeToken() {
  // Cerramos sesión
  this.cookie.delete('X-Token');
}
```

Se guardará el token gracias a setCookie. También podemos obtenerlo (para enviarlo a la API) y borrarlo, en caso de que tengamos que cerrar sesión.

También tenemos algunas funciones definidas para comprobar qué usuario ha iniciado sesión, ya que hay que recordar que en el token hemos incluído el email introducido en el login. A través de este, podremos saber qué usuario está actualmente visitando nuestra página.

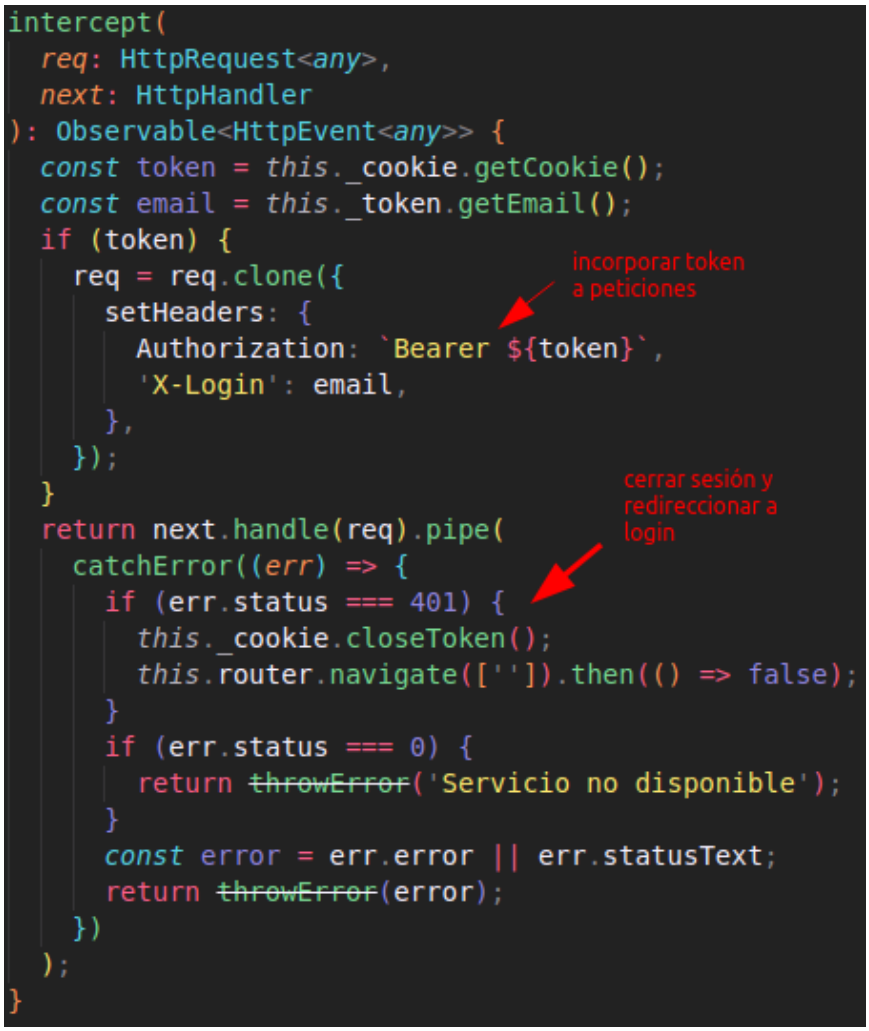
```
// 1. Get token content
getDecodedAccessToken(): any {
  try {
    return jwt_decode(this._cookieHandler.getCookie());
  } catch (Error) {
    return null;
  }
}

// 2. Get user
getEmail(): string {
  try {
    return this.getDecodedAccessToken()['user'];
  } catch (Error) {
    return '';
  }
}

// 3. Get Empleado
getEmpleado(): Observable<Empleado> {
  return this._empleado.getUserEmpleadoData(this.getEmail());
}
```

El primer método nos devuelve el token una vez se ha decodificado, con el segundo obtenemos el email del usuario que ha iniciado sesión y con el último directamente el empleado.

4.1.3) INTERCEPTOR

Se trata de un servicio cuya función principal es interceptar todas las peticiones que van hacia la API, e incorporar el token a las mismas. De esta forma una vez lleguen a la API van a tener el token para poder ser autenticadas.

Si no encuentran el token guardado en las cookies (si han pasado sus 24h de duración o se ha modificado a mano, por ejemplo) se cerrará sesión, y se redireccionará al login.




```
intercept(
  req: HttpRequest<any>,
  next: HttpHandler
): Observable<HttpEvent<any>> {
  const token = this._cookie.getCookie();
  const email = this._token.getEmail();
  if (token) {
    req = req.clone({
      setHeaders: {
        Authorization: `Bearer ${token}`,
        'X-Login': email,
      },
    });
  }
  return next.handle(req).pipe(
    catchError((err) => {
      if (err.status === 401) {
        this._cookie.closeToken();
        this.router.navigate(['']).then(() => false);
      }
      if (err.status === 0) {
        return throwError('Servicio no disponible');
      }
      const error = err.error || err.statusText;
      return throwError(error);
    })
  );
}
```

4.1.4) GUARDS

Se encargarán de realizar ciertas comprobaciones, y en caso de que no se cumplan nos redireccionarán a la ruta que establezcamos.

Se han creado 3 guards:

- Login.guard: si tenemos iniciada sesión (hay token válido presente en cookies) no nos dejará acceder a la página de login. Si la sesión está iniciada no tiene sentido ver el login antes de que se cierre.

- Token.guard: el efecto contrario al anterior. Si no hay sesión iniciada, no se podrá acceder a ninguna página de la web, exceptuando login, claro. De otro modo, cualquier persona podría cambiar la url de la página y acceder, saltándose el login.
- Requisito-admin.guard: no permitirá acceder a la página en cuestión si el usuario no es administrador. Utilizado para las funcionalidades solo accesibles por admins.

Podemos ver que en nuestra página tenemos muchas comprobaciones de seguridad. En primer lugar, no se mostrarían datos, pues las peticiones no estarían autorizadas. Además de eso, ni siquiera se podrá acceder a la página en primer lugar.

4.2) COMPONENTES

La página web está compuesta por los siguientes componentes, siendo los marcados en azul los accesibles únicamente por las cuentas con perfil de administrador:

- app-routing-module (no visible, actúa como servicio): se encarga de mostrar los componentes correspondientes dependiendo de la url actual de la página
- not-found: página que se muestra por defecto cuando la url no coincide con ninguna de las establecidas
- login: formulario en el que introducimos usuario y contraseña para iniciar sesión
- header: encabezado de nuestra página web, con menú de navegación, opciones relacionadas con cuenta y cerrar sesión
- footer: pie de página de nuestra web, con links de contacto, redes sociales y copyright
- account: formulario en el que mostramos los parámetros de la cuenta actual, y permite modificar algunos de ellos
- nuevo-empleado: formulario mediante el cual se puede crear una nueva cuenta. Comprueba que los parámetros introducidos tienen el formato adecuado, y que los campos únicos no existen ya para otros empleados (dni y correo).
- empleados: tabla que muestra todos los empleados. Tiene varias funcionalidades:
    - Se puede ordenar en sentido ascendente y descendente por cualquier parámetro (excepto id).
    - Tiene un buscador para buscar por cualquier valor en la tabla (cadena o numérico), filtrándose los resultados mostrados en la misma.
    - Permite seleccionar el número de registros a mostrar (en múltiplos de 5), y cambiar entre las páginas de resultados mostrados.
- calendario: calendario que muestra los fichajes (hora y nombre del empleado que ha fichado). Permite moverte por los meses, e incluso cambiar a formato semana o día, visualizando los datos en diferentes formatos.

5. HARDWARE (LECTOR DE TARJETAS + ARDUINO)

Llegados a este punto conocemos cómo se muestra la información de la base de datos, dónde y en qué formato, pero todavía no hemos tratado de dónde proceden los datos.

Una aplicación de fichajes requiere que se defina una forma para que los empleados puedan realizar los fichajes. El método más común (y efectivo) es utilizar tarjetas con información

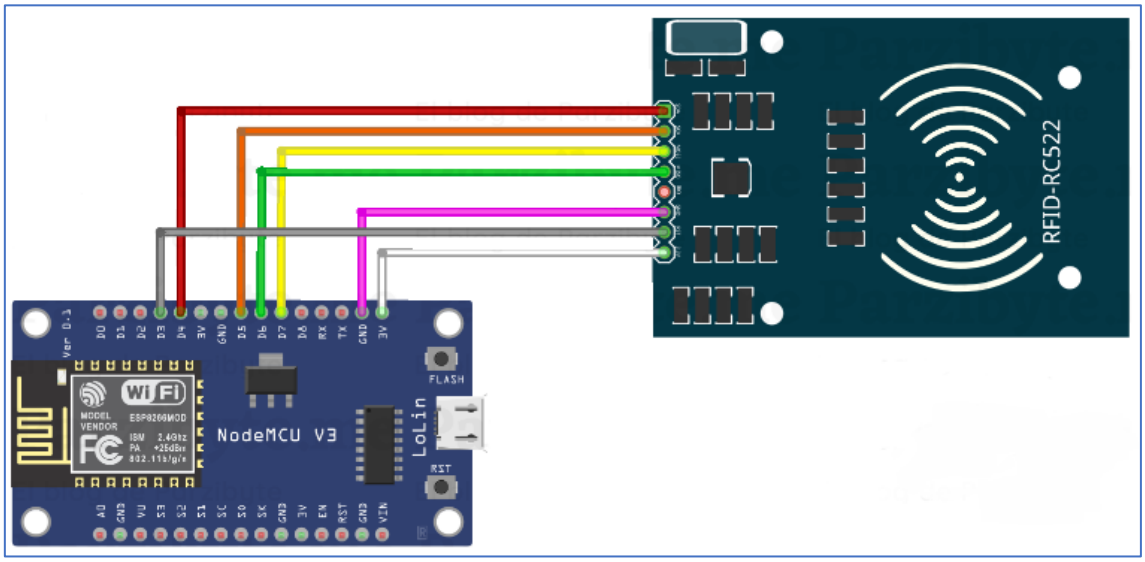
grabada. Cada empleado tendrá su propia tarjeta, que contiene su Id dentro del sistema de la empresa, por lo que, al pasarla por un lector, los datos serán enviados a la base de datos.

Como lector de tarjetas, tal y como se ha explicado anteriormente, se utilizará el RFID RC522, por su potencia y facilidad de uso. Para enviar los datos que este último lee, necesitaremos un componente de Arduino, que conectaremos al ordenador.

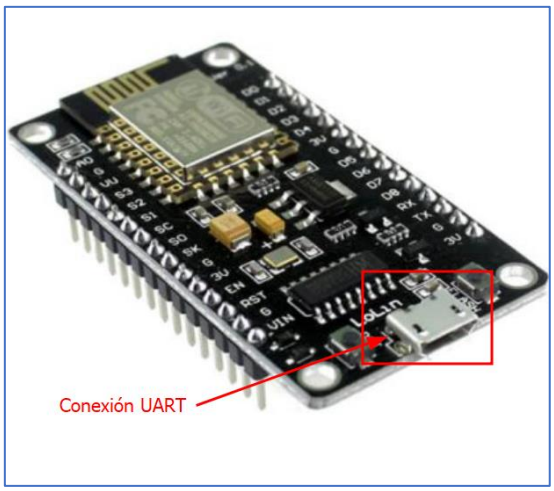
La pregunta es la siguiente: ¿cuál utilizar? La respuesta es simple: elegir un módulo de Arduino preparado para poder conectarse a la red. De esta forma, podremos enviar los datos a la API por una petición HTTP POST. Un NodeMCU V3 es perfecto para esta aplicación:

- Es potente (puede tratar una buena transferencia de datos).
- Es compatible con lenguajes Arduino IDE (plataforma utilizada para el desarrollo del código, a lo largo de este proyecto).
- Mayor facilidad para tratar con conexiones a la red.

Por lo tanto, ya sabemos el lector y módulo que vamos a utilizar. Debemos conectar ambos entre sí, y al ordenador mediante un cable UART-USB. El esquema de la conexión será el siguiente:



Tal y como podemos ver, a la izquierda tenemos el NodeMCU y a la derecha el lector RC522. Simplemente los conectamos con Jumper Wires tal y cómo indica la imagen, y posteriormente el NodeMCU V3 al ordenador, a través de su puerto UART (mediante un cable UART-USB).



Una vez montado y conectado al ordenador, simplemente tendremos que utilizar los programas correspondientes para:

- Grabar cada ID en una tarjeta distinta
- Leer las ID + enviarlas mediante Http

Al ejecutar el segundo programa el Arduino se quedará permanentemente a la espera para leer tarjetas. No realizará ninguna acción mientras no tenga una tarjeta delante. Cuando se acerque una, leerá su información y la mandará por Http.

Para visualizar el código de los programas utilizados léase el apartado del anexo denominado 'ACLARACIONES'.

PRUEBAS

Prueba página web https://youtu.be/1PuK0nwLsP0

Prueba fichaje administrador (hardware + web) https://youtube.com/shorts/BMaYeAmd1bY

Prueba fichaje usuario (hardware + web) https://youtu.be/pUMfT1tV0YE

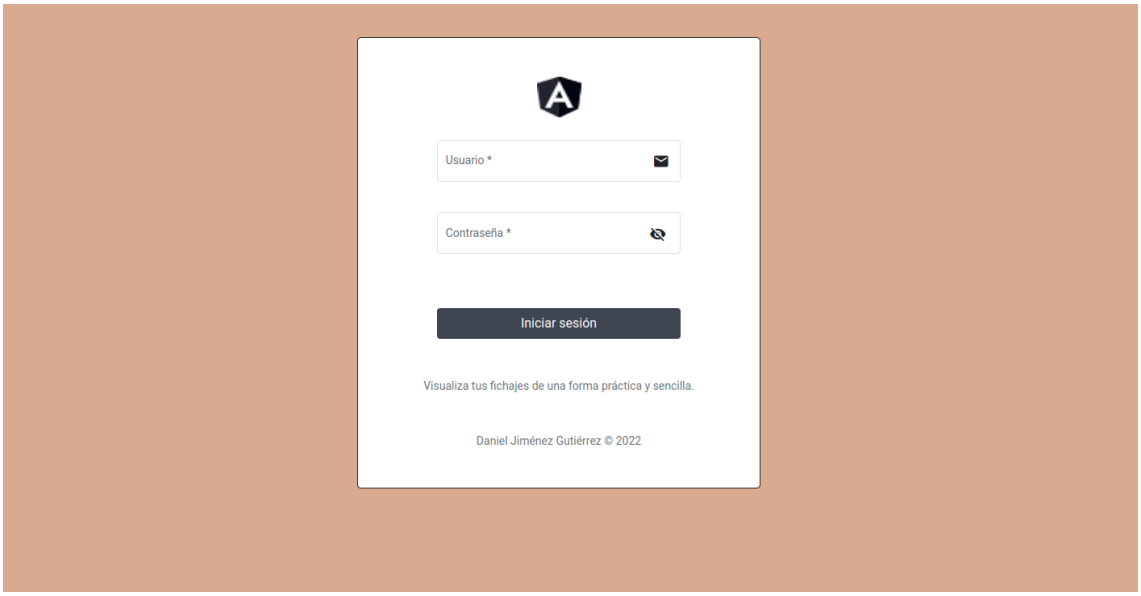
AMPLIACIÓN Y MEJORAS

- Historial/registro de fichajes mensuales, enviado por correo.
- Filtros en el calendario: por persona, intervalos de edad y otros parámetros.
- Sustituir icono de cuenta por imagen (permitir subir imagen o asignar con las iniciales de nombre y apellidos).
- Implementar gráficas mostrando diferentes parámetros a lo largo de semanas o mensual (fichaje, horas trabajadas…etc.).
- Incorporar LEDS al Arduino para que indique cuando se ha pasado la tarjeta y la ha detectado correctamente (indicación verde) o de forma errónea (indicación roja).

MANUAL DEL USUARIO

En este apartado se describen las acciones que el usuario puede realizar en la web, y el resultado esperado de cada una de ellas. Todos los datos personales utilizados para la demostración son inventados.

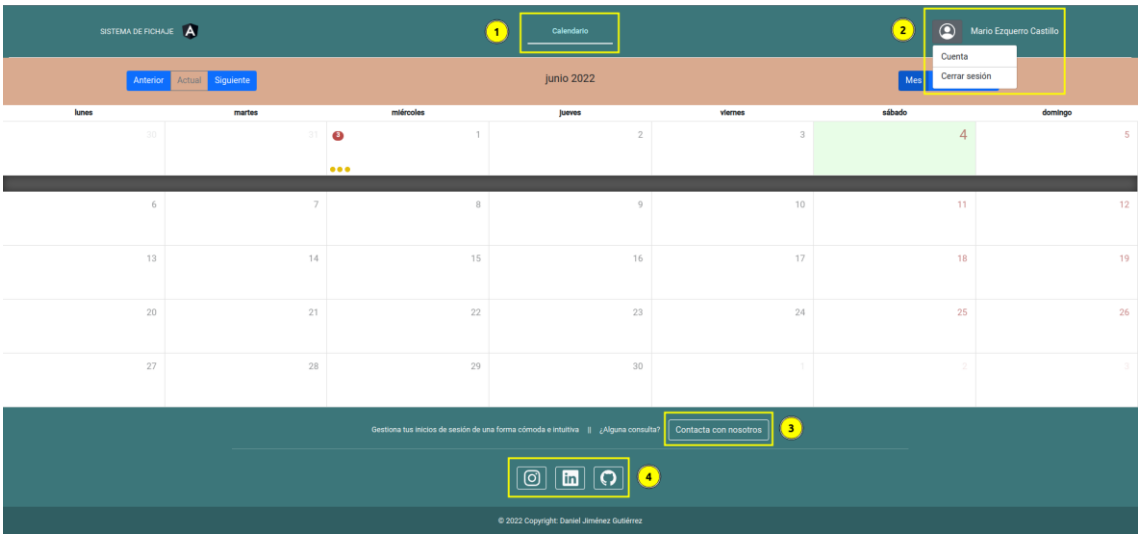
Cuando accedemos a la página nos encontramos con el Login.



Esta pantalla sirve para iniciar sesión. El usuario introducirá su usuario y contraseña. En caso de que la combinación de credenciales sea errónea, se avisará de ello y los inputs se vaciarán.

Cuando se introduzcan ambos correctamente, se redireccionará a la primera página de la web, el calendario. Recordemos que, dependiendo del rol del perfil que ha iniciado sesión (administrador o usuario), se mostrarán unas opciones u otras. Comenzaremos hablando de las opciones que tienen los usuarios.

1. USUARIOS



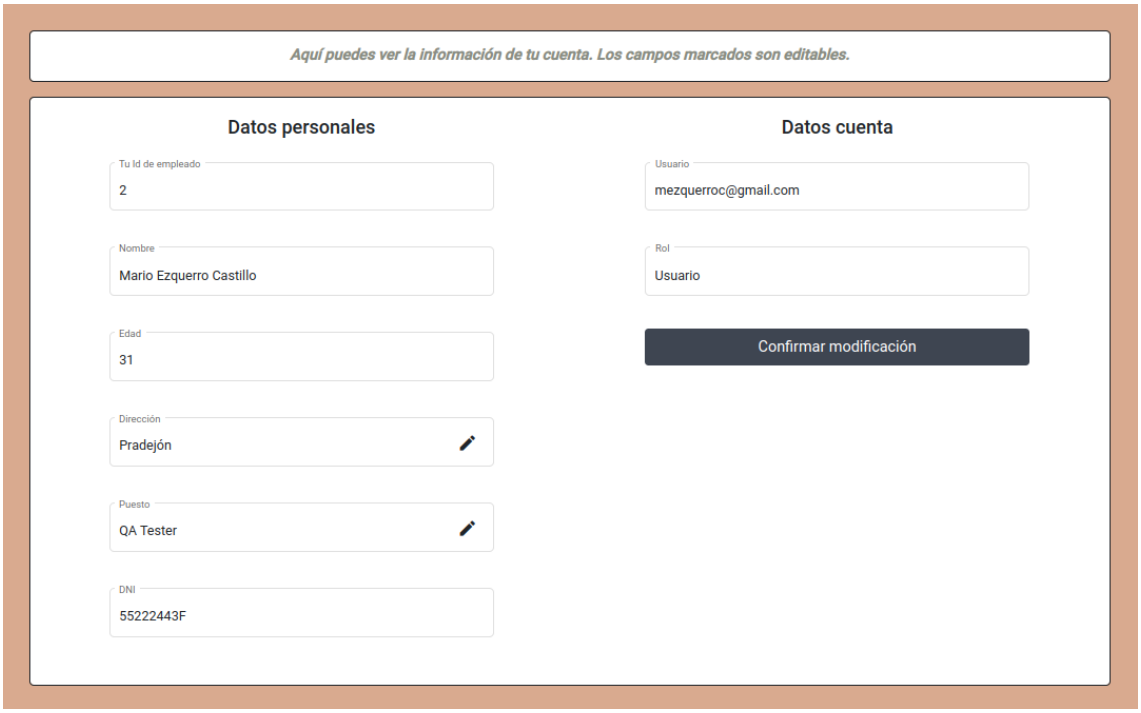
El header (encabezado) y footer (pie de página) son comunes en toda la página web. A continuación, se muestran las interacciones disponibles:

- Header
    1. En la parte superior central tenemos el menú de navegación, con el que podemos movernos a través de las distintas páginas (en caso de ser usuario, solamente tenemos la opción actual - calendario).
    2. En la parte superior derecha tenemos el icono de cuenta, junto a nuestro nombre. Si hacemos click en él, se mostrará un desplegable, en el que tenemos dos opciones: cuenta y cerrar sesión. La segunda opción borra el token de las

cookies (cierra sesión) y nos devuelve al login. La primera nos lleva a 'CUENTA' (visualizar siguiente apartado).

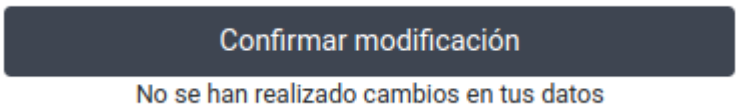
- Footer
  - En la sección superior del pie de página tenemos un botón que nos permitirá mandar un email a una dirección de correo establecida, simulando información de contacto con el soporte de la página web. Abrirá nuestro cliente de correo con la dirección establecida como destinatario.
  - Los botones inferiores del pie de página abren las redes sociales del desarrollador de la página web.

1.1) CUENTA



Se trata de un formulario en el que se muestra la información del empleado actual. Se pueden modificar los campos marcados con el lápiz (Dirección y puesto). Haciendo click en el mismo, los campos pasarán a ser editables y se podrán realizar cambios.

El botón de confirmar modificación aplicará los cambios y los actualizará en la base de datos (petición PATCH). En caso de que pulsemos el botón sin haber realizado cambios en ninguno de los dos campos, se avisará de ello y no se llevará a cabo ninguna petición.



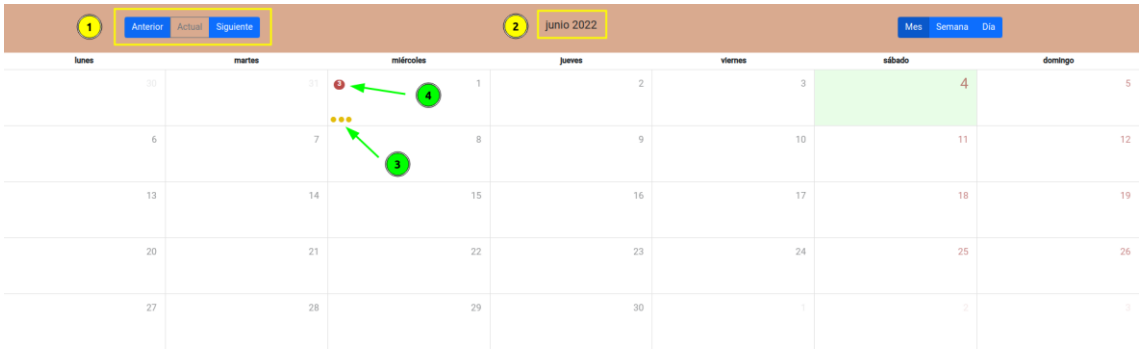
Si queremos regresar al calendario pulsaremos sobre el botón de navegación nombrado anteriormente (parte superior central).

1.2) CALENDARIO

En este calendario se mostrarán todos los fichajes que ha realizado el empleado actual. Se mostrarán en los días correspondientes, indicando la hora del fichaje.

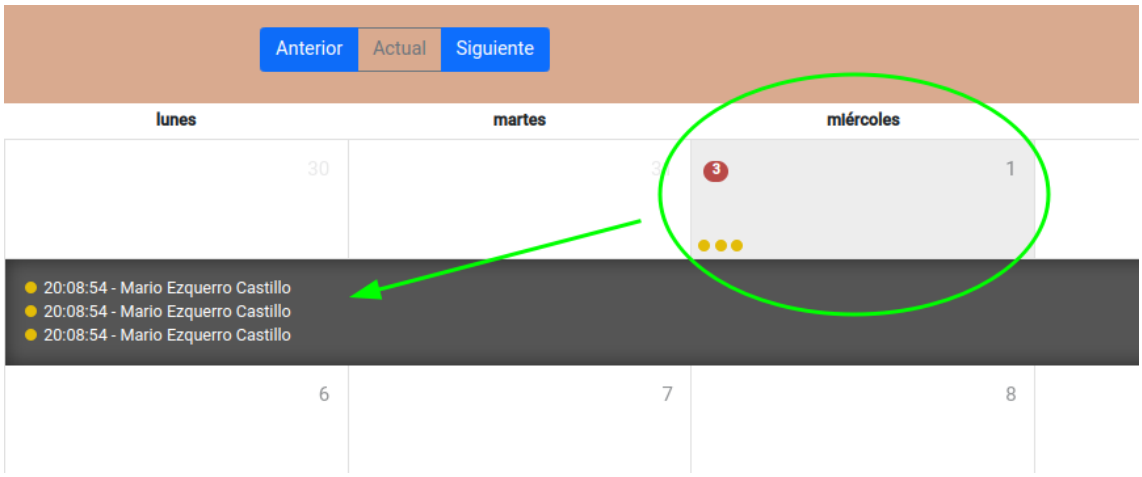
Este componente tiene tres vistas: podemos visualizar por mes/semana y día. Para cambiar entre vistas pulsaremos sobre los botones de la parte superior derecha, marcados en la siguiente imagen.

1.2.1) VISTA MENSUAL (Por defecto cuando carga el componente de calendario)

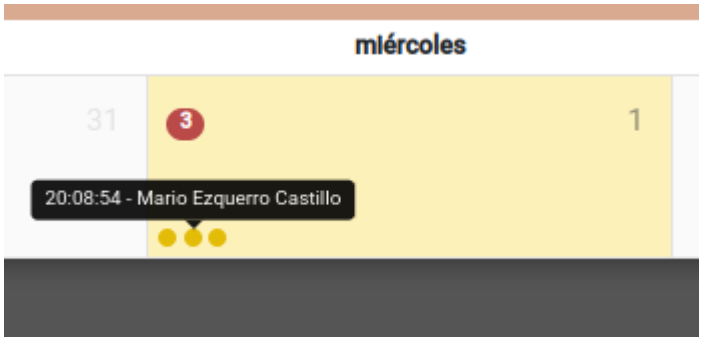


1. En la parte superior izquierda podemos movernos por los meses (anterior, siguiente y actual). Esta última opción nos llevará al mes en el que nos encontremos en este momento.
2. La parte central mostrará en todo momento el mes que estamos visualizando, conforme vayamos cambiándolo con las opciones anteriores.
3. Los puntitos en cada día son los fichajes de los empleados.
4. El número dentro del círculo representa los fichajes totales de ese día (suma de los fichajes anteriores).

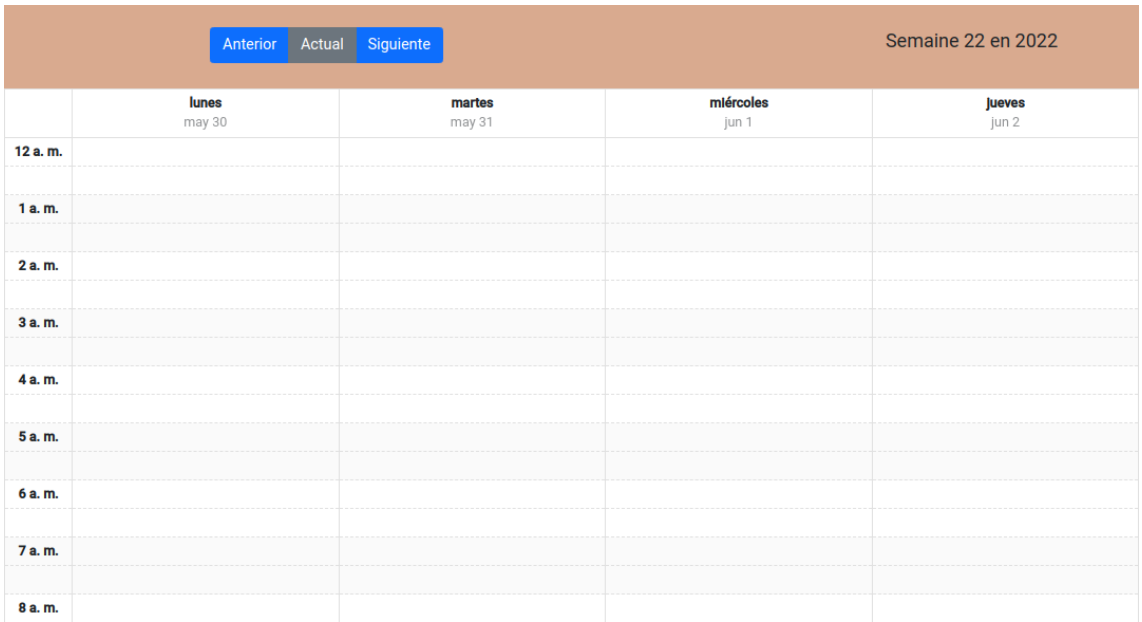
Podemos hacer click en un día que contenga fichajes para que se muestre una lista con todos ellos:



También podemos pasar el ratón por encima de los puntitos (fichajes) para que se muestre su información. En la siguiente foto se mantiene el ratón encima del punto central.



1.2.2) VISTA SEMANAL



(La imagen ha sido recortada para que se aprecie mejor la vista).

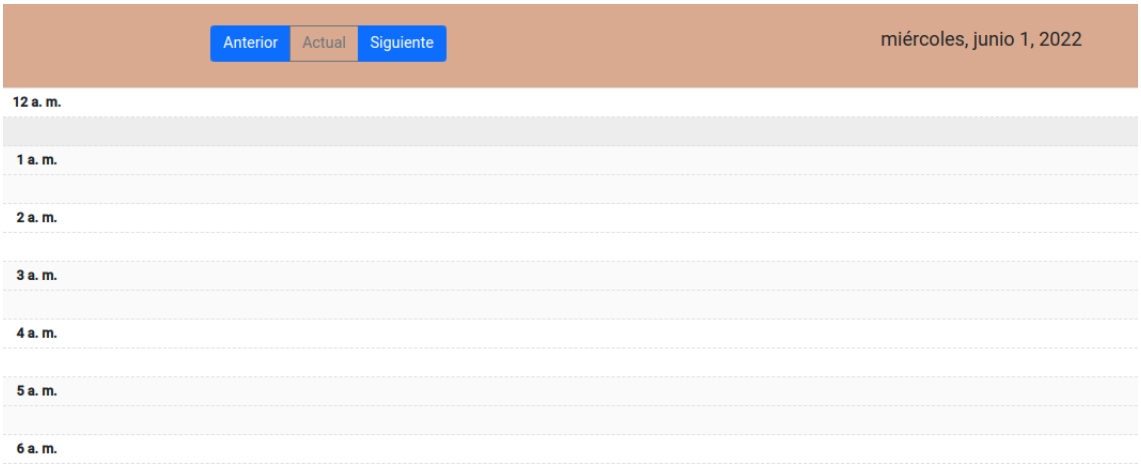
En este caso se mostrará por semanas. En el eje ‘X’ podemos ver los días de la semana, junto al número de día. En el eje ‘Y’ veremos las horas.

Tenemos las mismas opciones que en la vista mensual: mediante los botones de la parte superior izquierda podemos movernos por semanas, y la parte superior central mostrará la semana que estamos visualizando en cada momento.

Los fichajes se mostrarán en su periodo de tiempo correspondiente. Al igual que en la vista anterior, podemos pasar el ratón por encima para ver la información de los mismos.

1.2.3) VISTA DIARIA

Exactamente igual a la vista anterior, solo que se mostrará solamente un día.



Los menús superiores ya los conocemos: navegar entre días e indicador de día que estamos viendo en cada momento. Esta vista nos permite visualizar mejor los fichajes, ya que se muestran más espaciados.

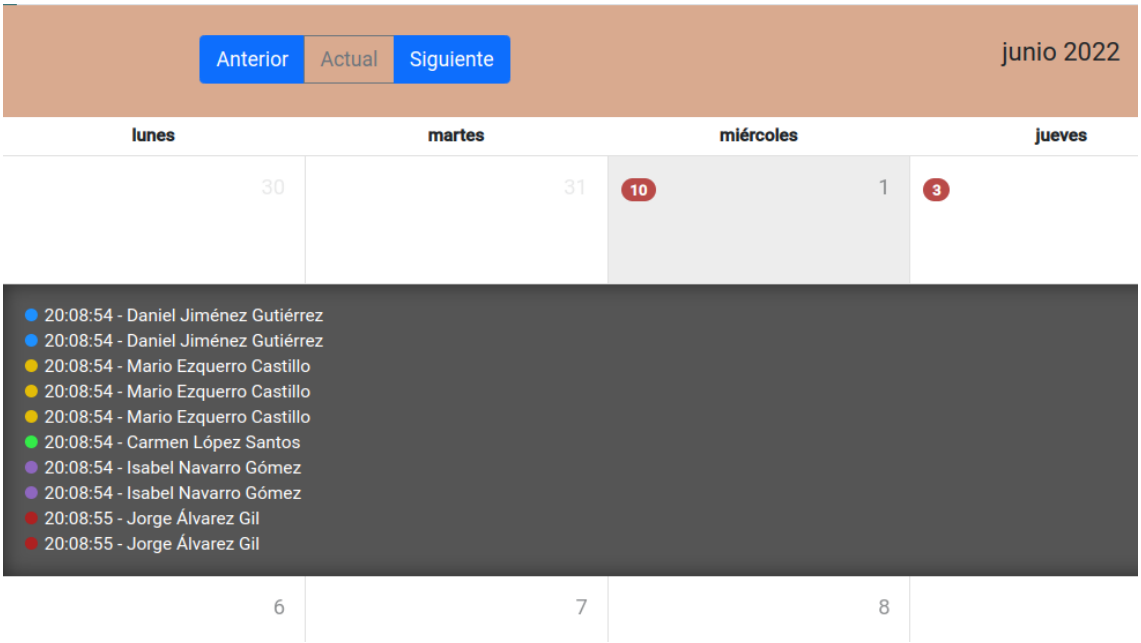
Al pasar el ratón por encima se muestra la información del fichaje, al igual que con las dos vistas anteriores.

2. ADMINISTRADORES

Los administradores pueden realizar todas las acciones que se acaban de explicar para los usuarios, además de un par de nuevas opciones. Antes de explicarlas, se hace una aclaración respecto al calendario:

Los usuarios verán solamente sus propios fichajes, mientras que a los administradores se le mostrarán los de toda la plantilla. Esto es por privacidad, para que un empleado no pueda ver información relativa a otro.

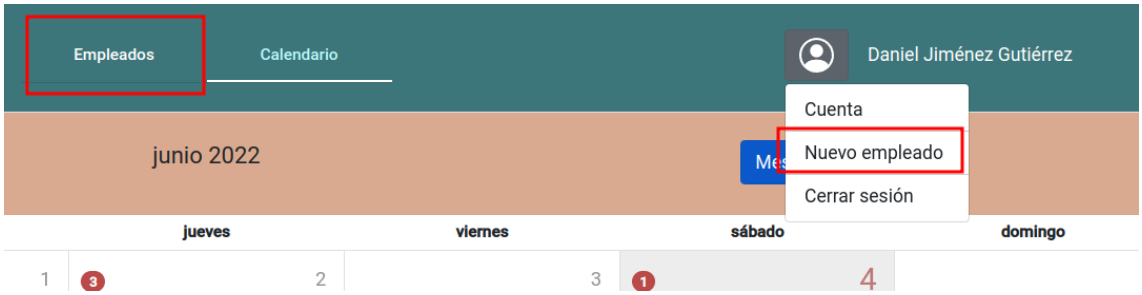
A continuación, se muestra cómo se visualizan los fichajes de todos los empleados en el calendario de un administrador:



Se mostrarán en 5 colores distintos, dependiendo de la ID del trabajador, para una menor confusión.

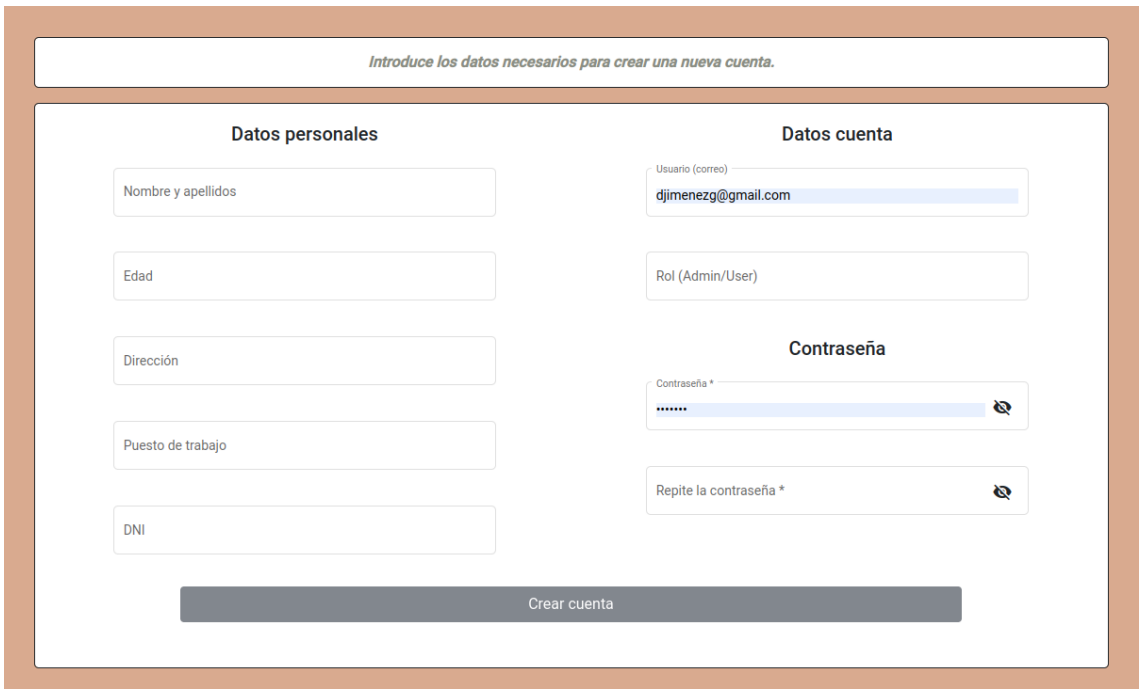
También podemos notar que no se muestran los puntitos en cada día. Esto es por no sobrecargar los días, ya que una gran cantidad de fichajes provocaría que veamos cada día lleno de puntitos (eventos).

Las opciones nuevas por ser administrador son: empleados y nueva cuenta. Estas opciones siendo usuario directamente no aparecían.



2.1) NUEVO EMPLEADO

Nos mostrará otro componente en el que podemos crear nuevas cuentas, para iniciar sesión en esta página web. Esto es algo necesario si sucede la incorporación de un nuevo empleado a la empresa.



Nos encontramos con un formulario, en el que tenemos que introducir todos los campos para que se desbloquee el botón inferior. Al pulsarlo, con todos los campos introducidos (con ciertas comprobaciones nombradas a continuación) se creará una nueva cuenta.

Comprobaciones que se realizan:

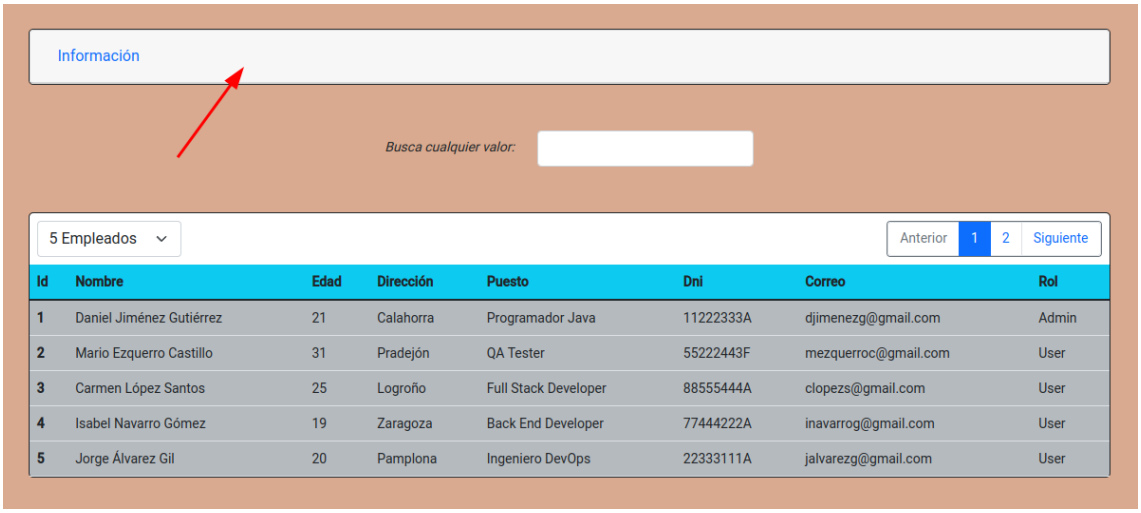
- El valor introducido en contraseña debe coincidir con repetir contraseña.
- El Rol es 'Admin' o 'User'.
- No existe ningún empleado ya registrado con el mismo DNI con el que estamos intentando crear la cuenta (DNI no repetido).
- El correo (email) no está siendo utilizado por ninguna otra cuenta.

Al pulsar el botón de crear cuenta se mostrará junto a él un mensaje de aviso, en caso de que no se cumpla alguna de estas condiciones. En dicho caso no se realizará ninguna petición a la API para que cree la cuenta y guarde en la base de datos.

Cuando se pulse el botón con todos los campos verificados y comprobados, se creará nueva cuenta y se avisará de ello, de nuevo, junto al botón inferior.

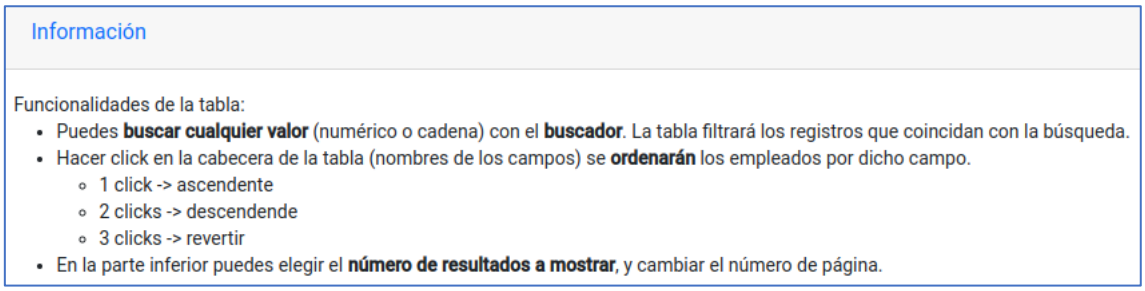
2.2) EMPLEADOS

Nos encontramos con un botón de información, un buscador, y una tabla que muestra todos los empleados de la plantilla de la empresa.



Id	Nombre	Edad	Dirección	Puesto	Dni	Correo	Rol
1	Daniel Jiménez Gutiérrez	21	Calahorra	Programador Java	11222333A	djimenezg@gmail.com	Admin
2	Mario Ezquerro Castillo	31	Pradejón	QA Tester	55222443F	mezquerroc@gmail.com	User
3	Carmen López Santos	25	Logroño	Full Stack Developer	88555444A	clopezs@gmail.com	User
4	Isabel Navarro Gómez	19	Zaragoza	Back End Developer	77444222A	inavarrog@gmail.com	User
5	Jorge Álvarez Gil	20	Pamplona	Ingeniero DevOps	22333111A	jalvarezg@gmail.com	User

Si pulsamos en el botón de información se desplegará una guía de las funcionalidades de la tabla:



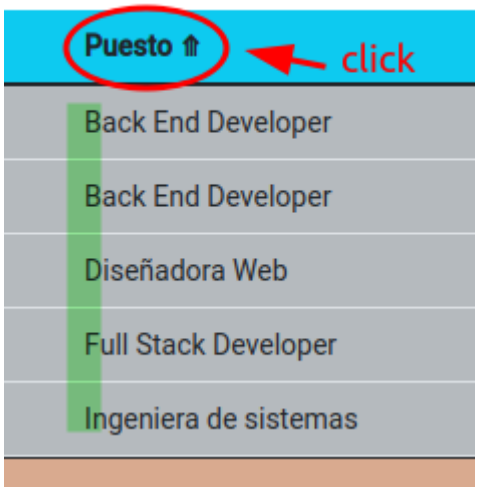
Tal y como podemos ver, la tabla tiene un buscador. Escribiendo en él cualquier valor se filtrarán los empleados que contienen dicho valor, sea numérico o texto. Por ejemplo: si buscamos '**or**' se mostrará J**or**ge Álvarez, o '**244**' se mostrará Mario Ezquerro, cuyo DNI es 5522**244**3F.

En la propia tabla podemos seleccionar el número de registros a mostrar (x empleados): en función del número de empleados que se muestren en cada momento tendremos un número determinado de páginas. Para movernos por ellas utilizamos los botones de navegación.

Id	Nombre	Edad	Dirección	Puesto	Dni	Correo	Rol
1	Daniel Jiménez Gutiérrez	21	Calahorra	Programador Java	11222333A	djimenezg@gmail.com	Admin

Si pulsamos en cualquier campo de la tabla (excepto Id) se ordenarán los registros por dicho campo:

- 1 click -> ascendente (⤊)
- 2 clicks -> descendente (⤋)
- 3 clicks -> vuelve a neutral (por Id ascendente)


Puesto
click
Back End Developer
Back End Developer
Diseñadora Web
Full Stack Developer
Ingeniera de sistemas

MANUAL DE INSTALACIÓN

Se necesitará instalar:

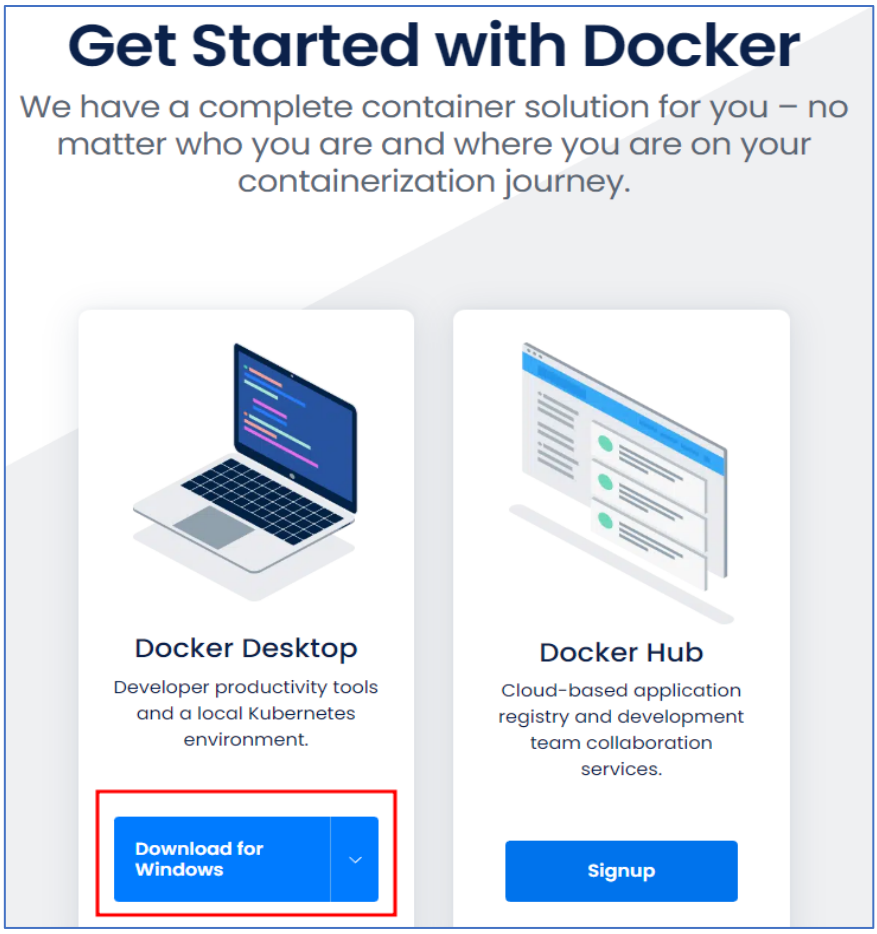
- Base de datos
  - SQL Server contenedor Docker
- API
  - .net6
- Relacionado con página web
  - Node.js
  - Angular

1. INSTALACIÓN CONTENEDOR DOCKER SQL Server

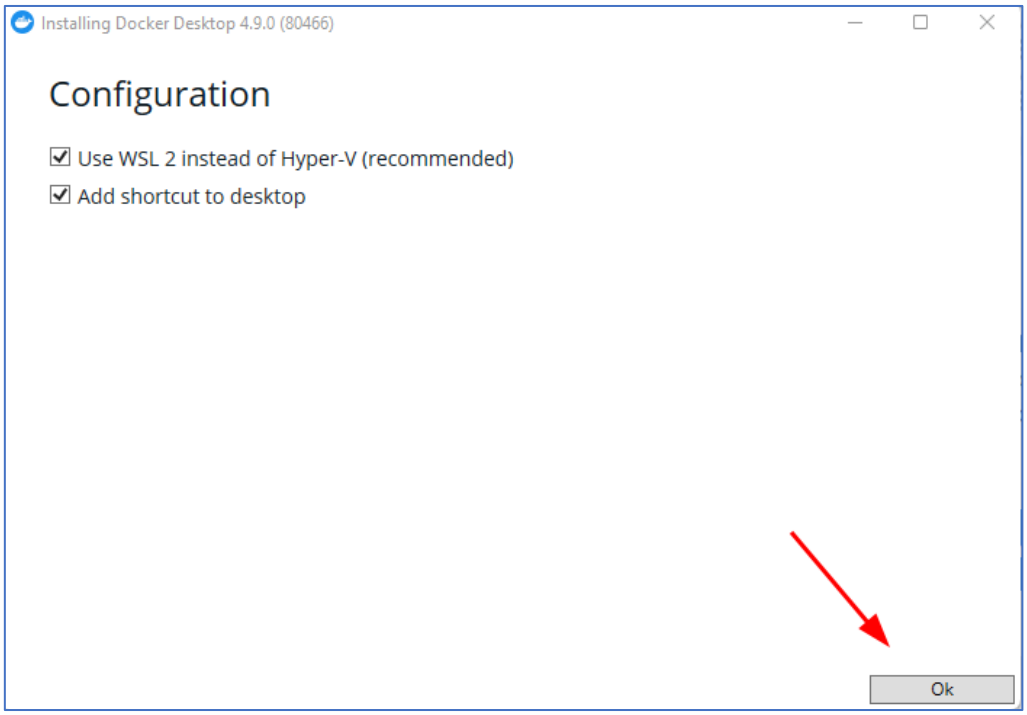
Docker nos permite tener una especie de contenedor que contiene la base de datos. Simplemente lanzándolo cada vez que queramos utilizarla vamos a poder acceder a ella.

Nos dirigimos a la siguiente página -> https://www.docker.com/get-started/

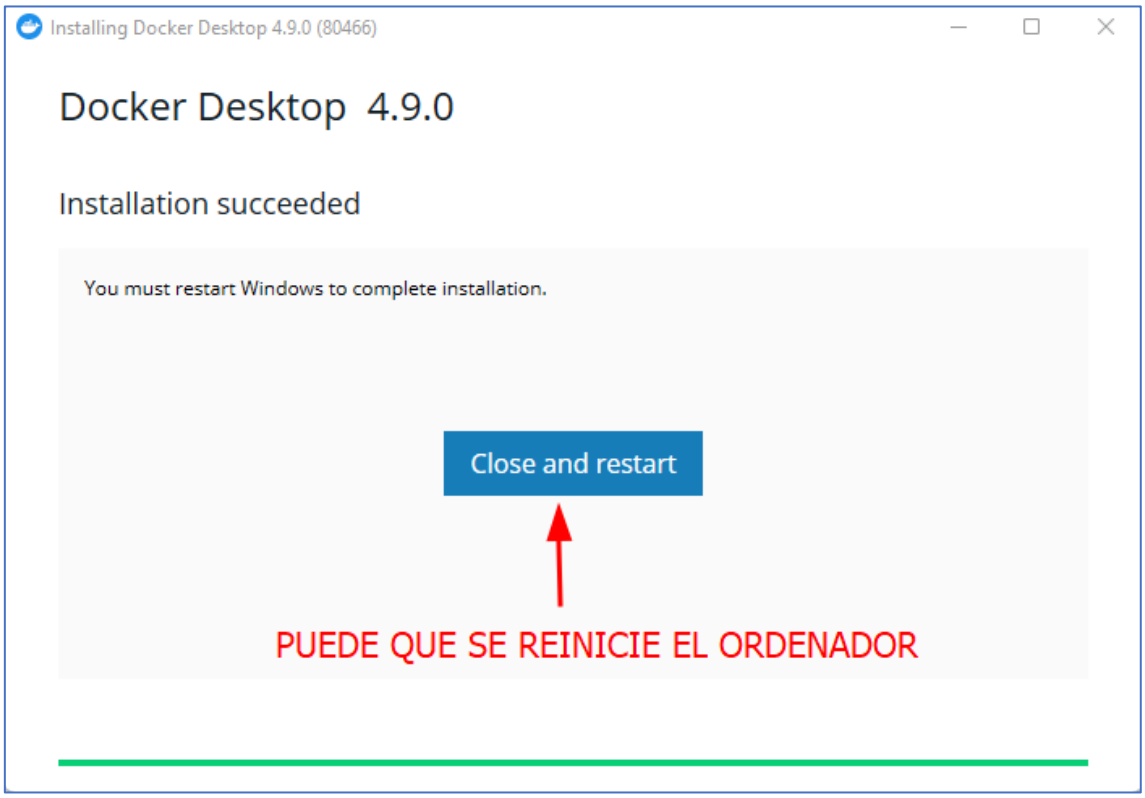
Pulsamos sobre 'Descargar para Windows'



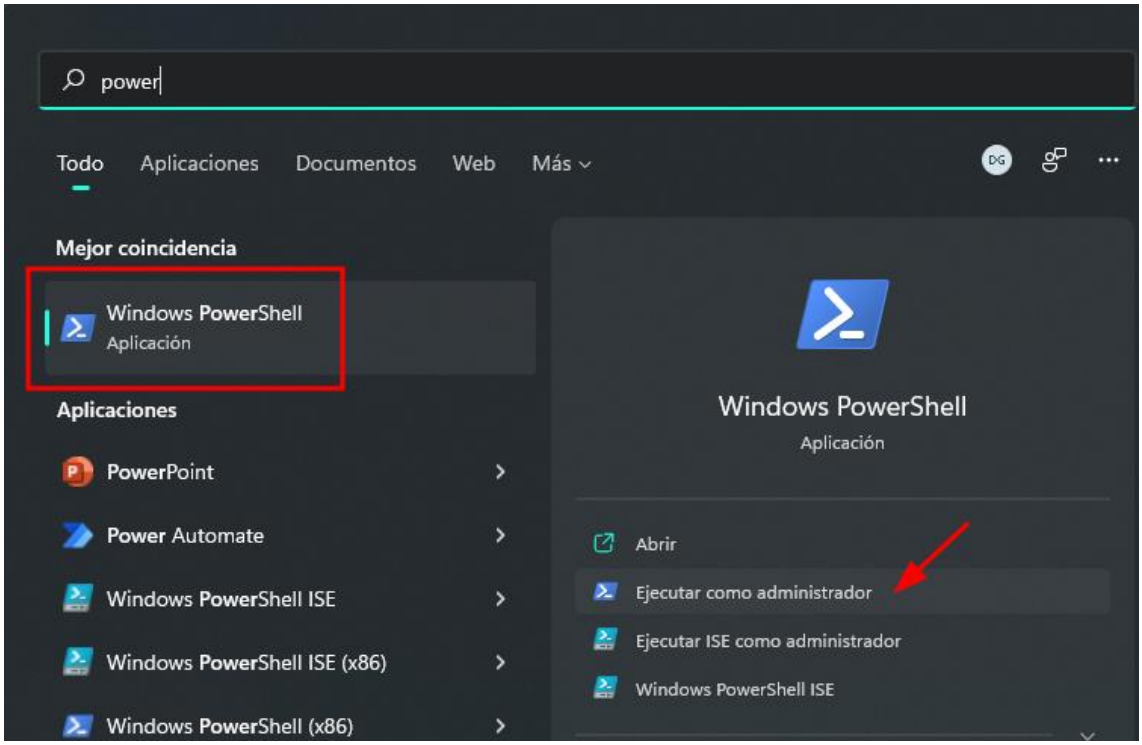
Esperamos a que se descargue el setup y lo ejecutamos.



Dejamos la opción recomendada (primer check). La segunda a elegir (creación de atajo en el escritorio). Pulsamos en Ok. Esperamos a que finalice la instalación y pulsamos en el botón (AVISO: probablemente se reinicie el ordenador).

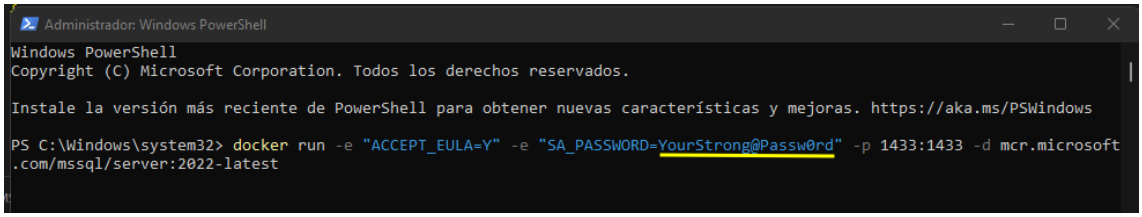


Una vez se ha reiniciado vamos a PowerShell en modo administrador. Ya tenemos Docker, por lo que tendremos que preparar el contenedor con la imagen de SQL Server, para tener nuestra base de datos.



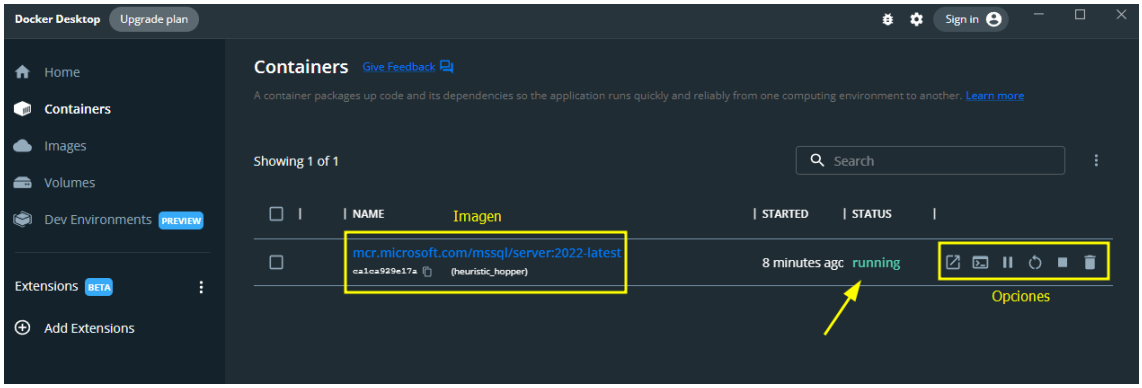
Escribimos el siguiente comando para preparar el contenedor. En la zona marcada debemos poner nuestra contraseña de la base de datos:

docker run -e "ACCEPT_EULA=Y" -e "SA_PASSWORD=YourStrong@Passw0rd" -p 1433:1433 -d mcr.microsoft.com/mssql/server:2022-latest



Pulsamos enter (ejecutamos). Esperamos un poquito a que finalice.

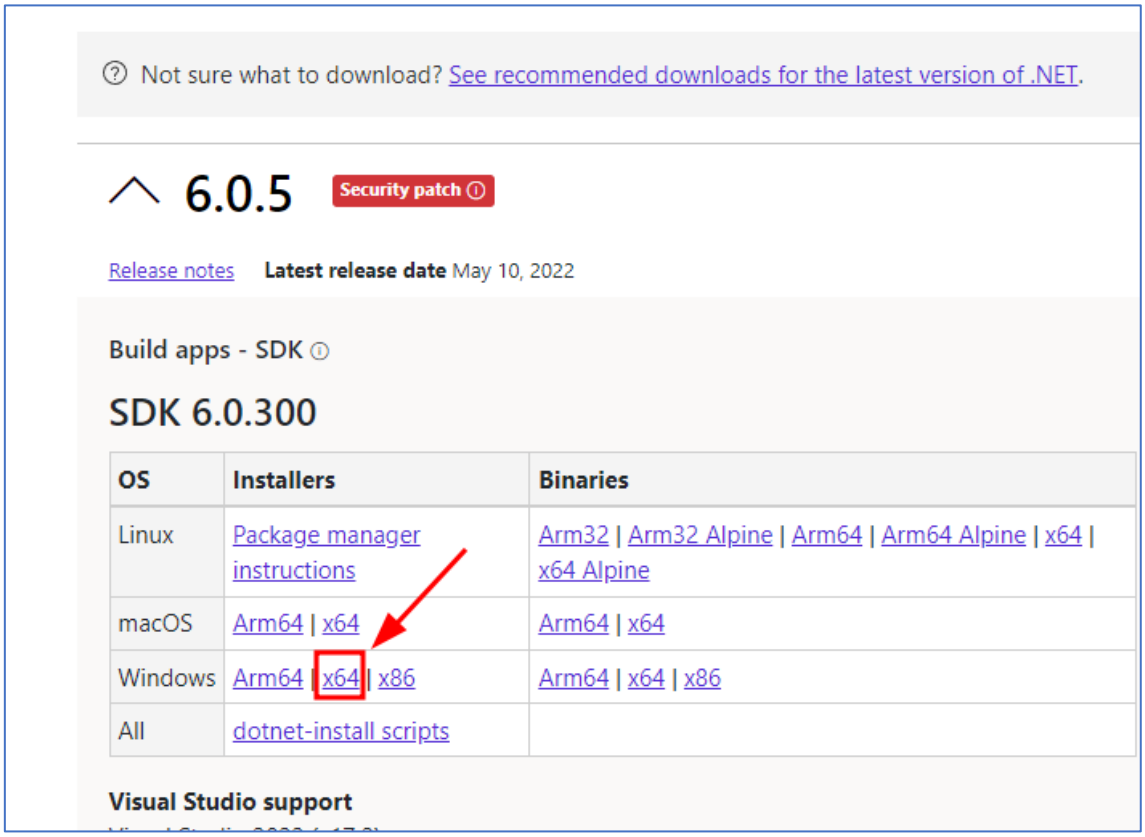
Ahora sí, abrimos Docker y vamos a 'Containers'. Veremos nuestro contenedor con la base de datos. Esta vez está ejecutado, por ser la primera vez, pero cada vez que queramos que la base de datos esté operativa nos dirigiremos a la página y lo lanzaremos.



Ya tenemos nuestra base de datos ejecutándose y preparada.

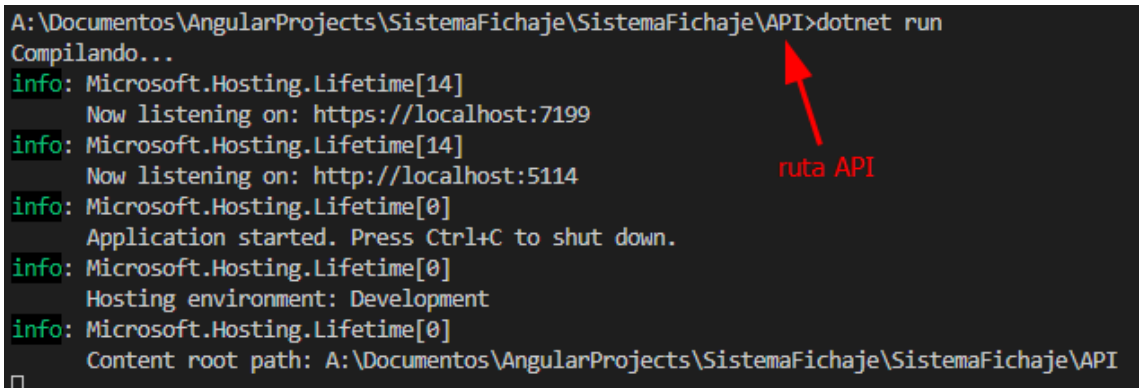
2. INSTALACIÓN .net6

Para que nuestra API funcione vamos a necesitar instalar dotnet 6. Para ello nos dirigimos a su página oficial: https://dotnet.microsoft.com/en-us/download/dotnet/6.0



Descargamos la versión x64 y ejecutamos el setup. Se instalará directamente. Ya podemos cerrarlo.

Ya tenemos instalado dotnet 6. Para ejecutar la API vamos a la ruta donde tenemos la carpeta correspondiente y ejecutamos el comando 'dotnet run'.



Ya tenemos la API preparada y lista.

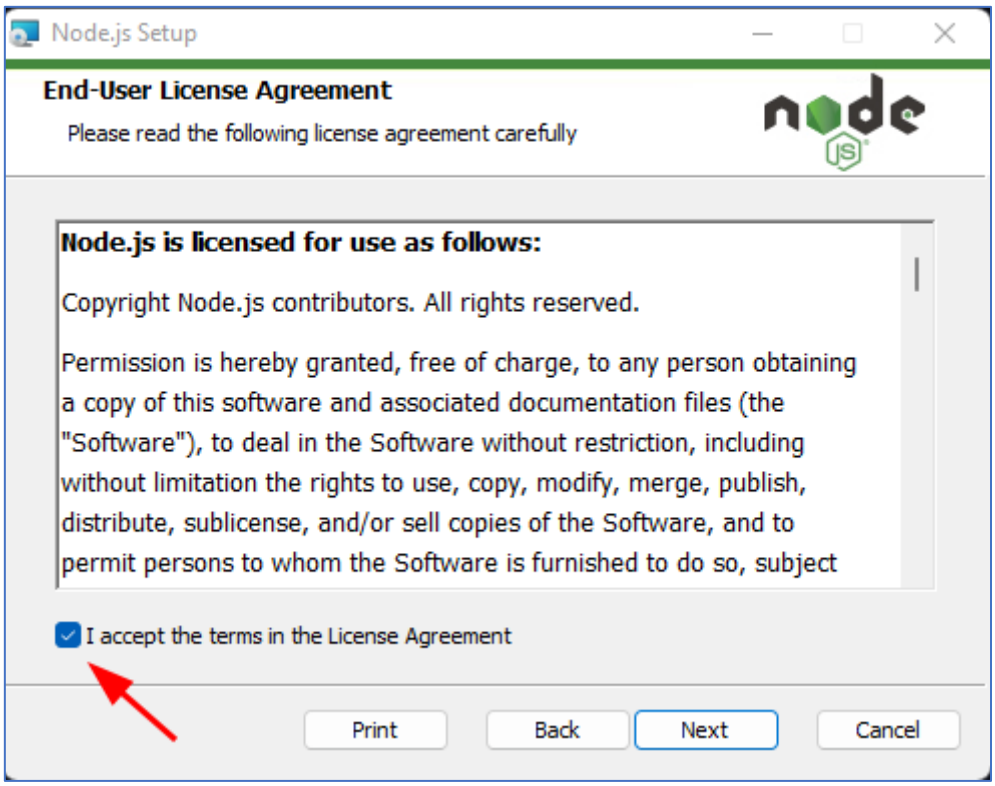
3. INSTALACIÓN Node.js - Angular

Comenzamos instalando Node.js.

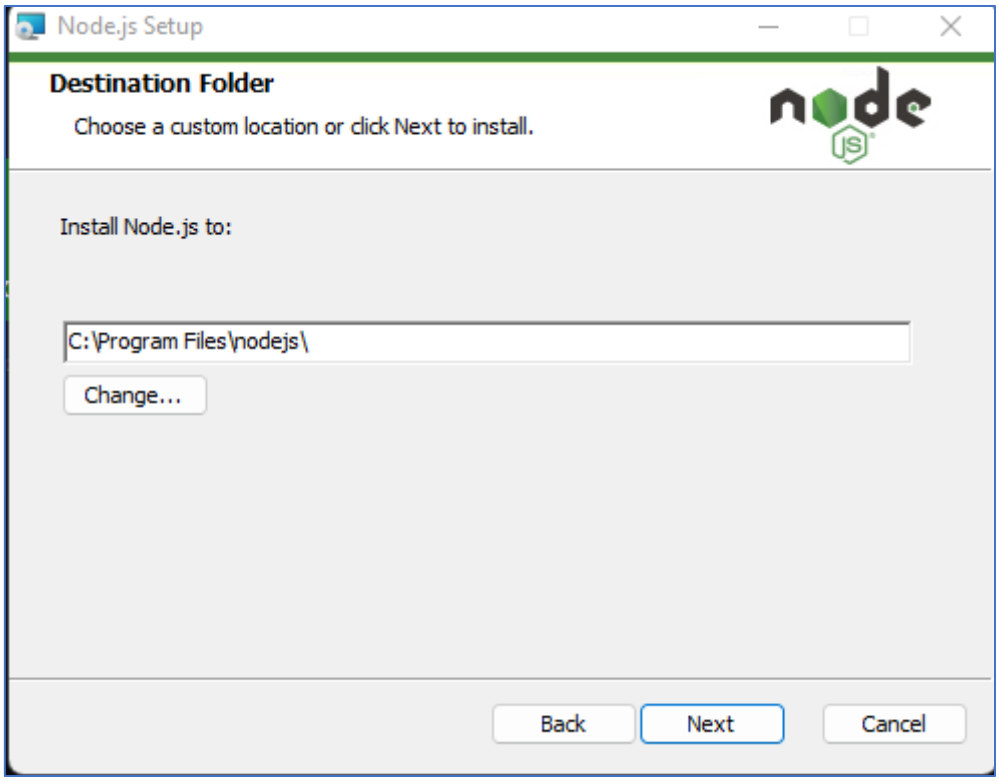
Para ello nos dirigimos a su página oficial: https://nodejs.org/es/

Descargamos la versión que nos recomiendan (LTS) marcada en la siguiente imagen:

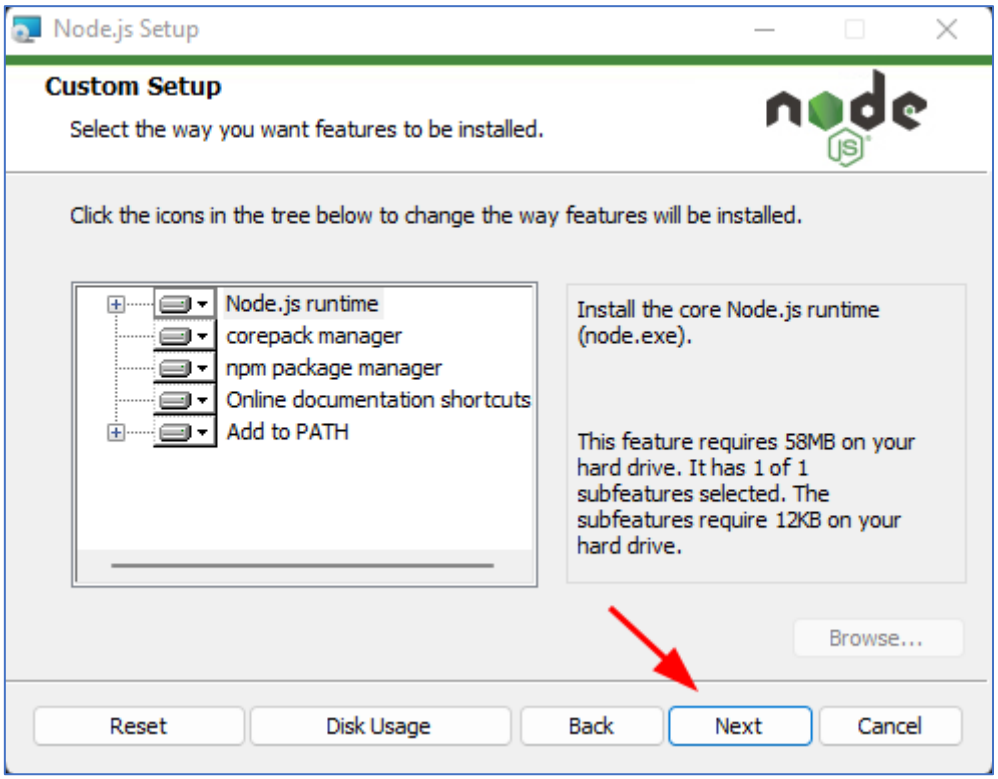
Ejecutamos el setup, aceptamos los términos de licencia.



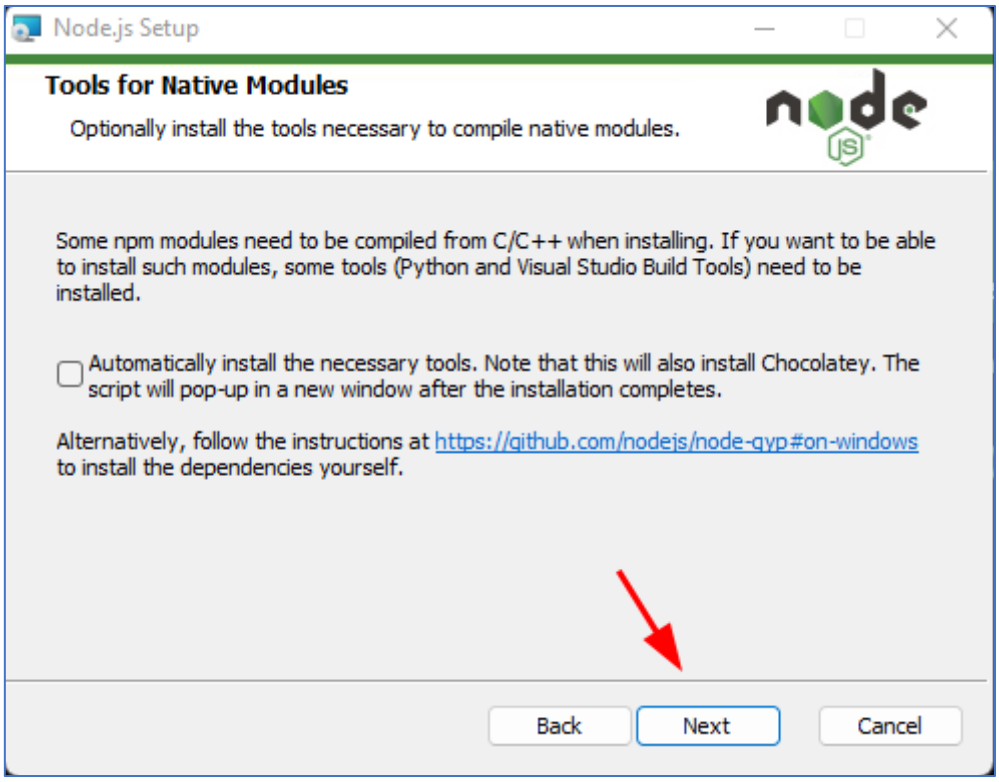
Elegimos nuestra ruta de instalación.



Seleccionamos las características a instalar, en mi caso le doy click a siguiente.



En la siguiente ventana NO le damos click al check, no nos van a hacer falta las herramientas que van a instalarse si lo marcamos.



Hacemos click en instalar y esperamos a que finalice. Con esto ya tenemos Node.js instalado. Ahora debemos abrir el proyecto (ruta de la WEB) y hacer npm install, para que se instalen todas las dependencias necesarias para la web.

```
AngularProjects\SistemaFichaje\SistemaFichaje\WEB> npm install
```

Esperamos a que se instalen todas (esperamos un rato ya que tardará lo suyo). Una vez finalice ya lo tenemos todo preparado. Para lanzar Angular tendremos que ejecutar el siguiente comando:

```
PS A:\Documentos\AngularProjects\SistemaFichaje\SistemaFichaje\WEB> npm start
vendor.js             | vendor        |   7.42 MB |
styles.css, styles.js | styles        | 426.40 kB |
polyfills.js          | polyfills     | 308.49 kB |
main.js               | main          | 261.47 kB |
scripts.js            | scripts       | 164.58 kB |
runtime.js            | runtime       |   6.52 kB |

                      | Initial Total |   8.56 MB

Build at: 2022-06-04T12:25:23.846Z - Hash: 631fe0d44b8ab958 - Time: 8776ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

Ya tenemos nuestra página web preparada.

CÁLCULO DE TIEMPOS FINAL

- Preparación hardware (planteamiento circuito, montaje con soldadura componentes, preparación Arduino y tarjetas (grabar IDs empleados): 10 horas
- Diseño y creación de la base de datos: 1.5 horas
- Diseño y creación de la API: 50 horas
- Diseño y creación página web hecha con Angular: 60 horas
- Testing/pruebas: 2 hora
- Documentación/memoria: 10 horas
- Preparación presentación (PowerPoint): 4 horas
- Preparación entorno para aplicación: 0.5 horas

Tiempo total: 138 horas

CONCLUSIÓN

Se trata de un proyecto con el que he disfrutado mucho, ya que he aprendido bastante. Considero que ha merecido la pena haber escogido esta opción.

Incorpora muchas tecnologías, tanto back como front, y las funcionalidades que aporta realmente son útiles para el mercado laboral. Es por esto que creo que podría llegar a usarse de forma real, con algunos retoques o mejoras.

Me ha servido como toma de contacto con nuevas tecnologías, con las que no había trabajado, como Jwt (codificación/cifrado en general) y Arduino (tenía conocimientos previos y algo de práctica, pero nunca lo había llevado a ningún proyecto real, o a una escala más amplia).

También he aprendido muchas funcionalidades de tecnologías con las que ya había trabajado:

- Realizar peticiones PATCH en API
- Interceptors, guards (en cuanto a tipos de servicios de Angular)
- Módulos como Material Angular o Angular Calendar, para crear y diseñar componentes y estilos para la web
- Usar directiva de ordenación para la tabla de empleados, y el formateador de fechas para la tabla (cambiar de idioma)

Sin embargo, lo que más saco en claro de todo esto es que he ganado bastante soltura a la hora de buscar documentación y soluciones. A lo largo de la realización de este proyecto me he topado muchas veces con cosas que me han resultado realmente difíciles o no sabía por dónde seguir, pero me las he apañado para encontrar alguna solución y no dejarlas de lado, incluso en algún caso habiendo descubierto soluciones por mí mismo.

He de añadir que Angular me parece increíble, y me encanta hacer APIs en .NET. Espero que se le encuentre el potencial que yo soy capaz de ver, ya que estoy bastante contento con el resultado obtenido.

En general, una experiencia muy positiva. Me ha gustado mucho tener la oportunidad (de verdad) de poder trabajar en un proyecto y participar tanto en el front como en el back, ya que me encantaría poder trabajar en ambas partes en futuros proyectos, por lo que no está nada mal como toma de contacto.

El código está disponible en el siguiente enlace: https://github.com/Danijg2k/TFG_SistemaFichaje

BIBLIOGRAFÍA

Material al que se ha recurrido para buscar información durante la creación de la aplicación.

Instalar Angular (con Node.js previo) https://ccbill.com/kb/install-angular-on-windows

Login (JWT en Cookies) https://codingpotions.com/angular-login-sesion

Autorización, interceptor, guard (para ver estructura general): https://github.com/programando-ideas/cursoangular/tree/master/Capitulo2

Autenticación API: https://github.com/cornflourblue/dotnet-5-jwt-authentication-api

Hash string: https://stackoverflow.com/questions/12416249/hashing-a-string-with-sha256

Guards en Angular: https://codingpotions.com/angular-seguridad

Interceptors en Angular: https://angular.io/api/common/http/HttpInterceptor

Conectar RFID RC522 con Arduino (primer prototipo): https://programarfacil.com/blog/arduino-blog/lector-rfid-rc522-con-arduino/

Esquema de montaje NodeMCU V3 (versión final):

https://parzibyte.me/blog/2020/11/23/leer-rfid-nodemcu-esp8266-rc522/

ANEXOS

TERMINOLOGÍA RELACIONADA CON HARDWARE

NodeMCU: Pequeña placa Wifi compatible con Arduino. Está montada alrededor del ESP8266. Ofrece más ventajas, como la incorporación de un regulador de tensión, y es conectable mediante un puerto UART.

ESP8266: Se trata de un chip integrado con conexión Wifi y compatible con el protocolo TCP/IP. El objetivo principal es dar acceso a cualquier microcontrolador a una red.

RFID RC522: El RFID RC522 (Identificador por radiofrecuencia) está diseñado para leer etiquetas (tags) a corta distancia, de forma inalámbrica. Posee comunicación SPI lo que facilita su uso con la mayoría de microcontroladores.

TERMINOLOGÍA RELACIONADA CON SOFTWARE

API: Conjunto de funciones y procedimientos que permite integrar sistemas, permitiendo que sus funcionalidades puedan ser reutilizadas por otras aplicaciones o software.

Rest: Estilo de arquitectura que hace énfasis en que las interacciones entre los clientes y los servicios se mejoran al tener un número limitado de operaciones (verbos).

API RESTful: aquella que ha sido diseñada usando la arquitectura REST.

Operaciones fundamentales de la API (CRUD), utilizadas en este proyecto:

OPERACIÓN	FUNCIÓN
GET	Obtener datos
POST	Enviar datos
DELETE	Borrar datos
PUT	Actualización
PATCH	Actualización parcial (ciertos atributos)

JWT: estándar abierto basado en JSON, para la creación de tokens de acceso que permiten la propagación de identidad y privilegios (claims en inglés).

Docker: plataforma de software abierta diseñada para crear y ejecutar aplicaciones de un modo ágil y versátil. Automatiza el despliegue de aplicaciones dentro de contenedores de software, permitiendo la virtualización automatizada de aplicaciones en múltiples sistemas operativos.

Swagger: conjunto de herramientas de software de código abierto para diseñar, construir, documentar, y utilizar servicios web RESTful.

Angular: framework para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página.

Framework: marco o esquema de trabajo generalmente utilizado por programadores para realizar el desarrollo de software.

Cookies del navegador: pequeños fragmentos de texto que los sitios web que visitas envían al navegador. Permiten que los sitios web recuerden información sobre tu visita, lo que puede hacer que sea más fácil volver a visitar los sitios y hacer que estos te resulten más útiles.

.net 6/dotnet 6: es un framework informático administrado, gratuito y de código abierto para los sistemas operativos Windows, Linux y macOS. Proporciona diversos servicios a las aplicaciones en ejecución.

Tipos de relaciones en bases de datos relacionales (en este proyecto se utiliza 1:N):

Notación	Significado	Ejemplo
1:1	1 entidad se relaciona con 1	1 persona tiene 1 DNI 1 DNI pertenece solo a 1 persona
1:N	1 entidad se relaciona con varias	1 persona tiene varios cuadernos 1 cuaderno pertenece a solo 1 persona
M:N	Varias entidades se relacionan con varias	1 persona tiene muchos hobbies 1 hobbie es practicado por muchas personas

VERSIONES TECNOLOGÍAS UTILIZADAS

Elemento	Versión
Imagen Docker mssql server	2022-latest
API	.NET 6
Angular	13.1.4
Arduino IDE (entorno)	1.8.19
Bootstrap (estilos)	5.1.3
Ng bootstrap (estilos)	11.0.0
Material angular (estilos)	13.3.7
Angular calendar	0.29.0
Flatpickr (necesario para calendar)	4.6.13
JQuery (para filtrado tabla)	3.6.0

ACLARACIONES

El código de los archivos utilizados por el NodeMCU (Arduino) se encuentra en el repositorio de Github del proyecto (link al final de la conclusión). Por motivos de complicaciones a la hora de mostrar el mismo (son programas largos que no se pueden mostrar aquí debidamente) se ha decidido encomendar la tarea de visitarlo.

Podemos encontrarlos en la carpeta denominada 'ARDUINO'. En ella tenemos dos carpetas: Arduino_Uno_R3 (prototipo inicial del proyecto), y NODEMCU (versión final). Se incluyen ambos para su visita y en caso de que se requiera volver a realizar alguna prueba con el componente inicial.