**ÍNDICE DE LA MEMORIA DEL PROYECTO**

1. [Introducción](#introduccion)
2. [Objetivos del proyecto](#objetivosdelproyecto)
3. [Justificación del sistema de fichajes](#justificacionsistemaficajes)
4. [Tecnologías utilizadas](#tecnologiasusadas)
5. Análisis del sistema  
   5.1. Requisitos funcionales  
   5.2. Requisitos no funcionales  
   5.3. Modelo entidad-relación (E/R)  
   5.4. Diagramas UML
6. Diseño de la solución  
   6.1. Arquitectura del sistema  
   6.2. Base de datos  
   6.3. Seguridad y autenticación
7. Desarrollo e implementación
8. Pruebas realizadas
9. Conclusiones y mejoras futuras
10. Anexos

**1.** **Introducción**

En la actualidad, la correcta gestión del tiempo laboral se ha convertido en un aspecto fundamental para las empresas. Tanto por motivos de organización interna como por cumplimiento de la normativa legal, llevar un registro de las horas de trabajo de los empleados es obligatorio en España desde el Real Decreto-ley 8/2019.

Este proyecto tiene como objetivo el desarrollo de un sistema de fichajes que permita a una empresa gestionar de forma sencilla y eficiente los horarios de entrada y salida de sus trabajadores. Además, el sistema contempla funcionalidades como el control de horas extra, la validación por parte de responsables, la auditoría de actividad, y la autenticación segura mediante verificación en dos pasos (2FA).

La aplicación está pensada para ser utilizada tanto por empleados como por el equipo de dirección y administración de la empresa. Por ello, se ha desarrollado una estructura basada en roles que permite adaptar las funcionalidades según el tipo de usuario.

Este proyecto se ha desarrollado como parte del Trabajo de Fin de Grado del ciclo formativo de **Desarrollo de Aplicaciones Multiplataforma (DAM)**, utilizando tecnologías modernas y seguras, y con el objetivo de simular una aplicación real que podría ser implementada en un entorno empresarial.

**2.** **Objetivos del proyecto**

**2.1 Objetivo general**

Desarrollar un sistema de fichajes que permita a las empresas registrar y gestionar el horario laboral de sus empleados, cumpliendo con la legislación vigente y mejorando la eficiencia y transparencia en la gestión del tiempo de trabajo.

**2.2 Objetivos específicos**

* Implementar un sistema de registro de entrada y salida de empleados.
* Permitir la gestión de solicitudes de horas extra y su posterior validación por parte del personal de dirección.
* Incorporar un sistema de roles para diferenciar permisos y accesos de usuarios (empleado, dirección, administrador).
* Garantizar la seguridad del sistema mediante autenticación de dos factores (2FA).
* Crear un historial de actividad que permita auditar las acciones realizadas por los usuarios.
* Facilitar la gestión de usuarios, empresas y roles desde un panel de administración.
* Desarrollar una base de datos relacional que almacene de forma estructurada y coherente toda la información del sistema.
* Diseñar una arquitectura backend/frontend modular basada en tecnologías modernas (Spring Boot + Angular/React).
* Realizar pruebas funcionales que aseguren el correcto funcionamiento del sistema.

**3.** **Justificación del sistema de fichajes**

Desde la entrada en vigor del **Real Decreto-ley 8/2019**, todas las empresas en España están obligadas a registrar diariamente la jornada laboral de sus empleados, incluyendo las horas de entrada y salida, así como las horas extraordinarias. Esta medida busca evitar el fraude laboral y garantizar el cumplimiento de los derechos de los trabajadores.

Sin embargo, muchas empresas siguen utilizando métodos poco eficientes como hojas de cálculo o registros en papel, lo cual puede ocasionar errores, pérdidas de datos o problemas legales en caso de inspección.

Este proyecto surge como respuesta a esa necesidad, ofreciendo una **solución digital, segura y sencilla** que permite automatizar y mejorar el proceso de control horario. Además, el sistema no solo cumple con la normativa legal, sino que añade funcionalidades que aportan valor añadido a las empresas, como:

* Gestión centralizada del personal.
* Control y validación de horas extra.
* Auditoría de acciones para mayor transparencia.
* Mejora de la seguridad gracias al uso de autenticación en dos pasos.

En definitiva, este sistema de fichajes busca facilitar a las empresas una herramienta útil y moderna para el cumplimiento de la ley y una gestión más eficiente de sus recursos humanos.

**4.** **Tecnologías utilizadas**

Para el desarrollo de este proyecto se han utilizado tecnologías actuales y ampliamente utilizadas en el desarrollo de aplicaciones empresariales. Se ha optado por una arquitectura **cliente-servidor**, separando el frontend del backend, lo que permite mayor escalabilidad y mantenimiento.

**4.1 Backend – Spring Boot**

* **Lenguaje**: Java
* **Framework**: Spring Boot
* **ORM**: Hibernate (JPA)
* **Seguridad**: Spring Security
* **Gestión de dependencias**: Maven
* **Base de datos**: MySQL
* **Gestor de base de datos**: DBeaver

Spring Boot facilita la creación de aplicaciones Java robustas con una configuración mínima. Junto con Hibernate, se ha implementado una gestión eficiente de la base de datos y las relaciones entre entidades.

**4.2 Frontend – Angular**

* **Lenguaje**: TypeScript (Angular)
* **Framework**: Angular
* **Comunicación con backend**: Servicios REST mediante HTTP
* **Diseño responsivo**: Tailwind

El frontend permite una experiencia de usuario fluida e interactiva, permitiendo que los usuarios realicen sus fichajes y gestiones de manera intuitiva.

**4.3 Otras herramientas**

* **GitHub**: Control de versiones y trabajo colaborativo.
* **Postman o Thunder Client**: Pruebas de endpoints de la API.
* **DBeaver**: Gestión de base de datos y pruebas SQL.
* **VS Code / IntelliJ IDEA**: Entornos de desarrollo para frontend y backend.
* **Google Authenticator**: Autenticación de dos factores (2FA).
* **Docker Desktop**: Inicio automático contenedor MySQL.

**5.** **Análisis del sistema**

**5.1** **Requisitos funcionales**

Los requisitos funcionales describen lo que el sistema **debe hacer**. A continuación, se detallan las funcionalidades principales del sistema de fichajes:

1. **Inicio de sesión de usuarios** mediante credenciales y autenticación de dos factores (2FA).
2. **Registro de entrada y salida** por parte de los empleados.
3. **Consulta del historial de fichajes** por parte del usuario.
4. **Solicitud de horas extra** por parte del empleado.
5. **Aprobación o rechazo de horas extra** por parte del usuario con rol de dirección.
6. **Creación, edición y eliminación de usuarios** (solo por administradores).
7. **Asignación de roles** a usuarios (administrador, dirección, empleado).
8. **Gestión de empresas** (solo por administradores).
9. **Auditoría del sistema**: el sistema debe guardar un historial de las acciones relevantes.
10. **Gestión del cambio de contraseñas** y su registro histórico.
11. **Activación/desactivación de autenticación 2FA** por parte del usuario.

**5.2** **Requisitos no funcionales**

Los requisitos no funcionales describen **cómo** debe comportarse el sistema. A continuación se detallan los principales aspectos:

1. **Seguridad**
   * Las contraseñas deben almacenarse de forma segura mediante hashing.
   * Se implementará autenticación en dos pasos (2FA) mediante Google Authenticator.
   * El acceso a funciones sensibles estará restringido por roles.
2. **Usabilidad**
   * La interfaz debe ser clara, intuitiva y accesible para todo tipo de usuarios.
   * El sistema debe estar disponible desde distintos dispositivos (responsive).
3. **Rendimiento**
   * Las operaciones del sistema deben ser rápidas y no bloquear la experiencia del usuario.
   * La base de datos debe estar bien optimizada para manejar múltiples registros diarios.
4. **Mantenibilidad**
   * El código debe estar estructurado y documentado para facilitar futuras mejoras.
   * Se utilizará control de versiones con Git para gestionar los cambios.
5. **Escalabilidad**
   * La arquitectura cliente-servidor permitirá que el sistema crezca fácilmente en número de usuarios y funcionalidades.
6. **Compatibilidad**
   * El sistema debe funcionar correctamente en los navegadores más utilizados (Chrome, Firefox, Edge).
7. **Disponibilidad**
   * El sistema debe estar disponible para su uso continuo durante la jornada laboral de la empresa.

**5.3** **Modelo entidad-relación (E/R)**

El modelo entidad-relación (E/R) representa de forma gráfica la estructura lógica de la base de datos. Permite visualizar las entidades principales del sistema, sus atributos y cómo se relacionan entre sí.

**Entidades principales del sistema:**

1. **Usuarios**  
   Contiene los datos personales y credenciales de acceso de cada usuario. Cada usuario pertenece a una empresa y puede tener uno o varios roles.
2. **Empresas**  
   Representa a las diferentes empresas que utilizan el sistema. Cada empresa puede tener múltiples usuarios.
3. **Roles**  
   Define el tipo de permisos que tiene cada usuario (Administrador, Dirección, Empleado). Un usuario puede tener varios roles.
4. **Fichajes**  
   Registra las entradas y salidas de los empleados. Cada fichaje está relacionado con un usuario.
5. **Horas extras**  
   Gestiona las solicitudes de horas extra. Cada solicitud está asociada a un usuario y puede ser aprobada por otro usuario con rol de dirección.
6. **Registro de cambios de contraseña**  
   Guarda un historial de los cambios de contraseña realizados por los usuarios.
7. **Configuración de autenticación**  
   Almacena el código secreto y el estado del 2FA para cada usuario.
8. **Historial de actividad**  
   Registra las acciones importantes realizadas en el sistema, como aprobaciones, cambios o fichajes.
9. **Usuarios\_Roles** (tabla intermedia)  
   Permite la relación de muchos a muchos entre usuarios y roles.

**Relaciones clave:**

* Una **empresa** tiene muchos **usuarios** (1:N).
* Un **usuario** puede tener muchos **roles**, y un **rol** puede estar asignado a muchos **usuarios** (N:M).
* Un **usuario** puede tener muchos **fichajes** y **horas extras** (1:N).
* Un **usuario** (de dirección) puede aprobar muchas **horas extras**.
* Un **usuario** tiene una única configuración de **autenticación 2FA** (1:1).
* Un **usuario** tiene muchos registros de **cambios de contraseña** y **actividades**.

Este modelo garantiza una estructura clara, escalable y adecuada para representar todas las funcionalidades del sistema de fichajes.

**5.4** **Diagramas UML**

Los diagramas UML (Lenguaje Unificado de Modelado) son herramientas visuales que permiten representar gráficamente distintos aspectos del sistema. En este proyecto se han utilizado los siguientes tipos:

**1. Diagrama de clases**

El diagrama de clases representa la estructura del sistema a nivel de programación. Muestra las clases, sus atributos, métodos y las relaciones entre ellas.

En este proyecto, las clases más importantes son:

* Usuario: contiene atributos como nombre, email, password, etc.
* Empresa: contiene la información de la empresa.
* Rol: define el tipo de acceso del usuario.
* Fichaje: representa una entrada o salida laboral.
* HoraExtra: almacena solicitudes de horas extra.
* RegistroCambioContrasenia, ConfiguracionAutenticacion y HistorialActividad representan componentes auxiliares del sistema.

Relaciones principales:

* Asociación entre Usuario y Empresa (muchos usuarios por empresa).
* Asociación Usuario ↔ Rol (N:M) mediante una clase intermedia.
* Herencia o asociación entre clases dependiendo del diseño detallado.

**2. Diagrama de casos de uso**

Este diagrama muestra las funcionalidades principales del sistema desde la perspectiva de los distintos tipos de usuario (actores).

**Actores del sistema:**

* Empleado
* Dirección
* Administrador

**Casos de uso principales:**

* Iniciar sesión (con 2FA)
* Fichar entrada/salida
* Solicitar horas extra
* Aprobar horas extra
* Consultar historial
* Gestionar usuarios, empresas y roles
* Consultar auditorías

Este diagrama es útil para entender qué puede hacer cada tipo de usuario y cómo interactúa con el sistema.

**6.** **Diseño de la solución**

El sistema de fichajes ha sido diseñado con una arquitectura moderna, separando claramente el **frontend** (interfaz de usuario) del **backend** (lógica de negocio y acceso a datos), lo que mejora la escalabilidad, el mantenimiento y la seguridad del proyecto.

**6.1** **Arquitectura del sistema**

Se ha seguido una arquitectura **cliente-servidor**, donde:

* El **cliente (frontend)** se encarga de mostrar la interfaz al usuario y enviar peticiones al servidor.
* El **servidor (backend)** gestiona la lógica del sistema, las reglas de negocio y el acceso a la base de datos.

La comunicación entre ambos se realiza mediante **peticiones HTTP REST** con datos en formato **JSON**.

**Resumen de capas:**

* **Frontend**: desarrollado con Angular (o React), ofrece una interfaz clara y accesible.
* **Backend**: desarrollado con Spring Boot, expone una API REST y controla la seguridad, lógica y validaciones.
* **Base de datos**: MySQL almacena de forma estructurada toda la información del sistema.

**6.2** **Base de datos**

La base de datos ha sido diseñada respetando las reglas de normalización, garantizando integridad referencial y rendimiento.

* Contiene entidades como: usuarios, empresas, roles, fichajes, horas\_extras, entre otras.
* Se utilizan claves foráneas para mantener relaciones entre tablas.
* Existen índices únicos en campos sensibles como email o username para evitar duplicados.
* Se han creado tablas intermedias (como usuarios\_roles) para manejar relaciones de muchos a muchos.

El modelo se diseñó inicialmente en papel (modelo E/R) y luego fue implementado directamente en MySQL usando DBeaver.

**6.3** **Seguridad y autenticación**

Para garantizar que el sistema sea seguro, se han aplicado las siguientes medidas:

* **Hash de contraseñas**: Las contraseñas no se almacenan en texto plano. Se utiliza un algoritmo de hashing como BCrypt.
* **Autenticación en dos pasos (2FA)**: Cada usuario puede activar el segundo paso de verificación con **Google Authenticator**. Esto agrega una capa extra de protección al inicio de sesión.
* **Control de roles y permisos**: Solo los usuarios con determinados roles pueden acceder a ciertas funciones, por ejemplo, solo dirección puede aprobar horas extra.
* **Auditoría de actividad**: Se registra toda acción importante del sistema, lo que permite revisar qué usuarios han realizado qué acciones y cuándo.
* **Restricción de accesos**: Cada endpoint del backend está protegido según el rol del usuario autenticado.

**7.** **Desarrollo e implementación**

El desarrollo del sistema de fichajes se ha realizado siguiendo una metodología **ágil**, dividiendo el trabajo en tareas pequeñas y manejables. Esto ha permitido avanzar de forma organizada, probando y validando cada parte del sistema a medida que se desarrollaba.

**7.1 Fases de desarrollo**

**1. Diseño inicial y planificación**

Se definieron los requisitos del sistema, tanto funcionales como técnicos. Se diseñaron los modelos de base de datos y se planificó la arquitectura general (cliente-servidor).

**2. Implementación del backend**

* Se creó el proyecto Spring Boot con las dependencias necesarias (JPA, Security, Web, etc.).
* Se desarrollaron las entidades, repositorios, servicios y controladores REST.
* Se implementó la autenticación con seguridad básica y más adelante se añadió la integración con Google Authenticator para 2FA.
* Se configuraron los roles y permisos usando Spring Security.

**3. Implementación del frontend**

* Se inició el proyecto Angular (o React).
* Se crearon las vistas principales para fichajes, login, gestión de usuarios y horas extra.
* Se integraron los servicios REST del backend usando HTTP y JSON.
* Se mejoró la experiencia de usuario con validaciones y diseño responsive.

**4. Integración y pruebas**

* Se conectó frontend y backend en local para verificar el funcionamiento completo del sistema.
* Se probaron todos los casos de uso: login, fichajes, creación de usuarios, 2FA, etc.
* Se validó la integridad de los datos en la base de datos mediante DBeaver.

**7.2 Herramientas utilizadas durante el desarrollo**

* **Visual Studio Code / IntelliJ IDEA**: editores principales de código.
* **Postman y Thunder Client**: para probar los endpoints REST.
* **DBeaver**: para gestionar y verificar la base de datos.
* **GitHub**: control de versiones.
* **Google Authenticator**: para probar el 2FA en dispositivos reales.

**8. Pruebas realizadas**

Para garantizar el correcto funcionamiento del sistema, se realizaron diferentes tipos de pruebas durante y después del desarrollo. Estas pruebas permitieron detectar errores, validar requisitos y asegurar que todas las funcionalidades cumplieran con lo esperado.

**8.1 Pruebas funcionales**

Las pruebas funcionales se centraron en verificar que cada funcionalidad del sistema funcionara correctamente de principio a fin.

**Ejemplos de pruebas:**

* ✅ **Login de usuarios**
  + Se probó el inicio de sesión con credenciales válidas e inválidas.
  + Se verificó el funcionamiento del segundo paso de autenticación con Google Authenticator.
* ✅ **Registro de fichajes**
  + Un empleado puede fichar la entrada y la salida correctamente.
  + No se permite realizar dos fichajes de entrada seguidos sin cerrar el anterior.
* ✅ **Gestión de horas extra**
  + Un usuario puede solicitar horas extra.
  + Dirección puede aprobar o rechazar la solicitud.
  + El estado cambia correctamente y se guarda el aprobador.
* ✅ **Gestión de usuarios y roles**
  + El administrador puede crear, editar y desactivar usuarios.
  + Se puede asignar uno o varios roles a cada usuario.
* ✅ **Historial de actividad**
  + Se registran automáticamente acciones importantes (fichajes, aprobaciones, etc.).

**8.2 Pruebas de integración**

Se verificó la correcta comunicación entre las distintas partes del sistema:

* Frontend y backend se comunican a través de peticiones HTTP usando JSON.
* Los datos introducidos desde la interfaz se almacenan correctamente en la base de datos.
* Las respuestas del backend se muestran de forma clara en el frontend.

**8.3 Pruebas de seguridad**

* Se validó que las contraseñas se almacenan en formato hash.
* Se comprobó que un usuario sin rol adecuado no puede acceder a funciones restringidas.
* Se probó la activación y uso del 2FA con Google Authenticator.

**8.4 Pruebas de base de datos**

* Se verificó que las relaciones entre tablas funcionan correctamente (por ejemplo, la relación entre usuarios y empresas).
* Se probaron restricciones como claves únicas y claves foráneas.
* Se realizaron consultas de ejemplo para validar la estructura y consistencia de los datos.

**9. Conclusiones y mejoras futuras**

**9.1 Conclusiones**

El desarrollo de este sistema de fichajes ha permitido poner en práctica gran parte de los conocimientos adquiridos durante el ciclo de **Desarrollo de Aplicaciones Multiplataforma (DAM)**, tanto en programación backend como frontend, diseño de bases de datos y seguridad.

El sistema cumple con los objetivos planteados, ofreciendo una solución funcional, segura y adaptable que permite a una empresa registrar el horario laboral de sus empleados, gestionar solicitudes de horas extra, controlar el acceso mediante roles y mantener un historial de auditoría.

Además, se ha tenido en cuenta la normativa vigente, permitiendo que el sistema pueda ser utilizado en entornos reales donde el cumplimiento legal es esencial.

También se ha trabajado con herramientas profesionales como Spring Boot, Angular, DBeaver y Git, lo que contribuye a que el proyecto se acerque al mundo real del desarrollo software.

**9.2 Mejoras futuras**

Aunque el sistema es completamente funcional, existen algunas mejoras que podrían implementarse en el futuro para ampliar sus capacidades:

* Implementar un sistema de **notificaciones** por correo electrónico ante eventos importantes (aprobación de horas extra, cambios de contraseña, etc.).
* Crear un módulo de **informes y estadísticas** con gráficos sobre el tiempo trabajado por empleado o equipo.
* Añadir control de geolocalización del fichaje para evitar suplantaciones (por ejemplo, fichar desde casa sin permiso).
* Mejorar el sistema de permisos con una **gestión más granular**.
* Internacionalizar la aplicación para permitir su uso en distintos idiomas.
* Desarrollar una **versión móvil** (app nativa o PWA) para facilitar el fichaje desde el móvil.

**10. Anexos**

En esta sección se incluyen documentos, diagramas y otros elementos de apoyo que complementan la información desarrollada en la memoria. Estos anexos ayudan a visualizar mejor la estructura del sistema y permiten una comprensión más técnica y detallada del proyecto.

**10.1 Scripts SQL**

Se adjuntan los siguientes archivos:

* **modelo\_datos\_fichajes.sql**: contiene la estructura de la base de datos con todas las tablas necesarias.
* **datos\_ejemplo\_fichajes.sql**: incluye datos de ejemplo para poblar la base de datos y facilitar las pruebas del sistema.

**10.2 Modelo entidad-relación (E/R)**

Se presenta el diagrama E/R del sistema, en el que se muestran las entidades principales, sus atributos y las relaciones entre ellas. Este diagrama fue diseñado como base para la creación de la base de datos relacional.

📝 *(Aquí puedes incluir una imagen del modelo E/R exportada desde DBeaver o cualquier herramienta de modelado que hayas usado.)*

**10.3 Diagramas UML**

* **Diagrama de clases**: muestra la estructura del sistema desde el punto de vista del desarrollo, con las clases y relaciones.
* **Diagrama de casos de uso**: representa las interacciones de los distintos tipos de usuarios con el sistema.

📝 *(También puedes añadir aquí capturas o imágenes de estos diagramas generados con herramientas como Lucidchart, Draw.io, StarUML, etc.)*

**10.4 Capturas de pantalla**

Se incluyen capturas del funcionamiento del sistema:

* Pantalla de inicio de sesión con autenticación 2FA.
* Panel de usuario para registrar fichajes.
* Vista de administración de usuarios y roles.
* Formulario de solicitud y validación de horas extra.
* Historial de actividad y auditoría.

📝 *(Puedes tomar capturas de tu aplicación y pegarlas aquí con un pie de foto explicativo.)*