

# ÍNDICE

<b>INTRODUCCIÓN.....</b>	<b>3</b>
<b>APP WEB.....</b>	<b>4</b>
INTERFACES.....	4
PÁGINA PRINCIPAL.....	4
INICIAR SESIÓN/LOGIN.....	6
PERFIL.....	7
CAMBIAR CONTRASEÑA.....	8
GESTOR DE TAREAS.....	11
EDITAR TAREA.....	13
ELIMINAR TAREA.....	14
SUPERVISOR.....	15
DENEGADO.....	17
CREAR NUEVA TAREA.....	18
CREAR NUEVO PROYECTO.....	19
<b>BACK-END.....</b>	<b>22</b>
DEPENDENCIAS DEL PROYECTO.....	22
APPLICATION.PROPERTIES.....	22
CONTROLADOR.....	23
JPA.....	32
REPOSITORIOS.....	35
SEGURIDAD.....	36
<b>FRONT-END.....</b>	<b>38</b>
<b>CONCLUSIÓN.....</b>	<b>41</b>
<b>BIBLIOGRAFÍA.....</b>	<b>42</b>

# INTRODUCCIÓN



MagicWater es una empresa líder en la evaluación de la calidad del agua en áreas específicas. Reconocida por su compromiso con la excelencia en la investigación y el análisis, la compañía se ha destacado en el campo de la ingeniería ambiental y la gestión de recursos hídricos. Consciente de la necesidad de optimizar sus procesos internos, el director de MagicWater ha propuesto el desarrollo de una aplicación de gestión de tareas adaptada a las demandas particulares de la empresa.

Con el objetivo de mejorar la eficiencia y la organización de sus actividades, la aplicación web diseñada para MagicWater debe satisfacer una serie de requisitos específicos. En un entorno donde la mayoría de los empleados son ingenieros encargados de proyectos de análisis de la calidad del agua, la aplicación debe permitir la asignación y el seguimiento de tareas de manera eficaz. Desde la recolección de muestras hasta la elaboración de informes, cada etapa del proceso debe integrarse de manera coherente y automatizada.

La aplicación debe ofrecer una estructura flexible que diferencie entre roles de supervisor y trabajador. Mientras los trabajadores pueden gestionar sus propias tareas, los supervisores tendrán acceso completo a todas las tareas asignadas. Además, se espera que la aplicación genere informes detallados sobre el progreso de los proyectos, facilitando así la toma de decisiones informadas y la optimización continua de los recursos de MagicWater.

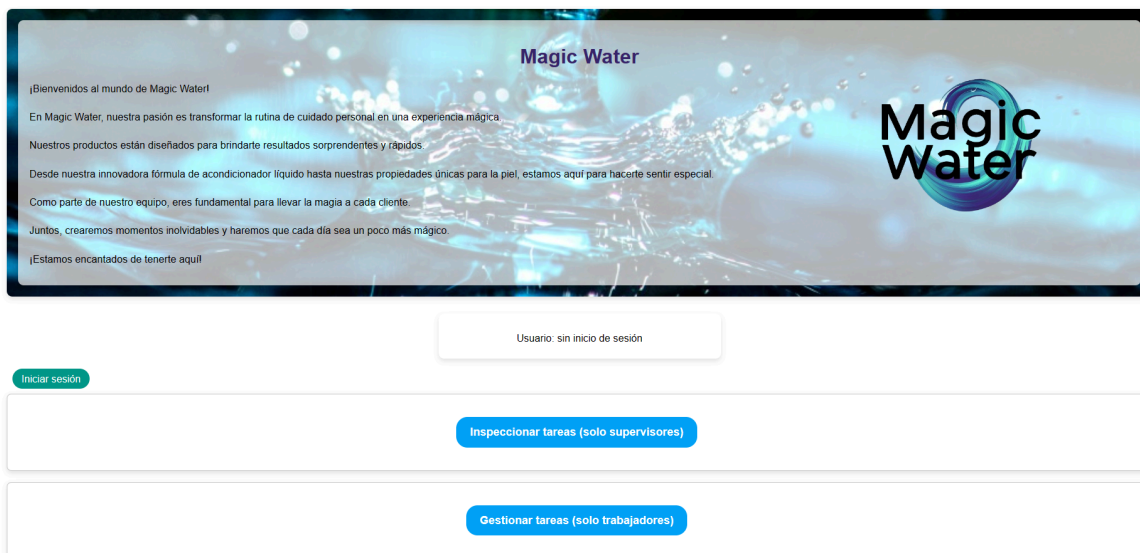
# APP WEB

## INTERFACES

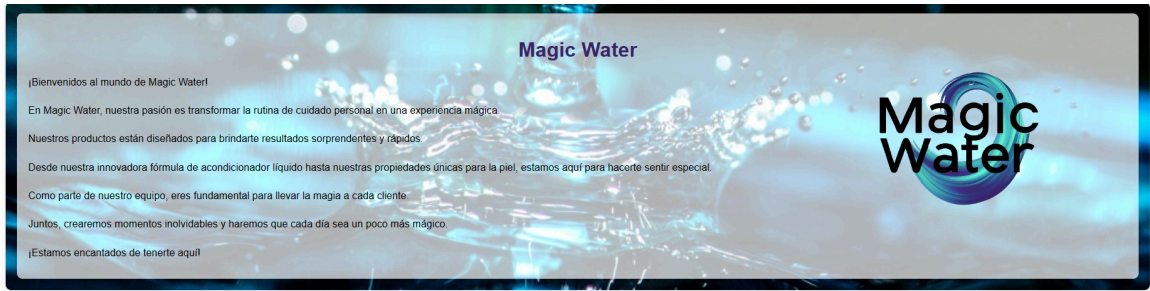
### PÁGINA PRINCIPAL

DESPUÉS DE LA MODIFICACIÓN

AL ABRIR LA PÁGINA:



UNA VEZ INICIADO SESIÓN:



Usuario: 67890123F  
Isabel Garcia  
Ingeniero

[Cerrar sesión](#) [Perfil](#)

[Inspeccionar tareas \(solo supervisores\)](#)

[Gestionar tareas \(solo trabajadores\)](#)

## INICIAR SESIÓN/LOGIN



### Inicie sesión

Ir a la tu cuenta

67890123F

...

[Iniciar sesión](#)

## PERFIL



#### Edita tu perfil

Nombre:

Isabel

Apellidos:

Garcia

Categoría:

Ingeniero

Dirección de email:

isabel.garcia@email.com

Teléfono:

543210987

Guardar cambios

Cambiar  
contraseña

## CAMBIAR CONTRASEÑA

**AL QUERER CAMBIAR LA CONTRASEÑA:**



Establece la nueva contraseña

Nueva contraseña:

Confirmar contraseña:

Guardar nueva contraseña

**AL NO COINCIDIR LAS CONTRASEÑAS:**



**Las contraseñas no coinciden**

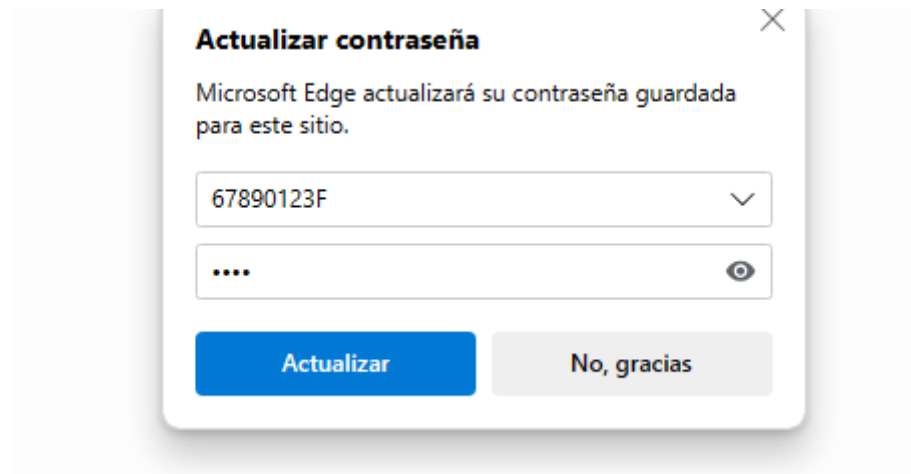
Establece la nueva contraseña

Nueva contraseña:

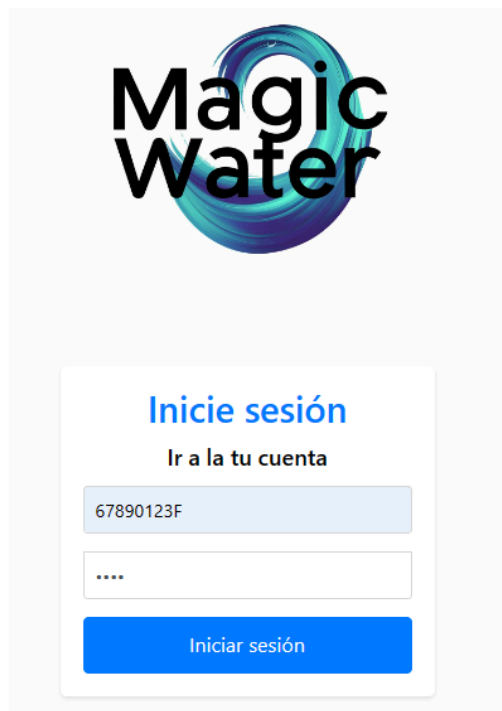
Confirmar contraseña:

Guardar nueva contraseña

## AL CAMBIAR LA CONTRASEÑA:



## INICIAR SESIÓN CON NUEVA CONTRASEÑA:



Ahora debemos añadir la nueva contraseña = 1234

**GESTOR DE TAREAS**

# GESTOR DE TAREAS

Crear nueva tarea





Crear

Crear nuevo proyecto

Crear



## Proyecto: Investigación de la Contaminación Acuática en Getafe

Estudio de la contaminación acuática en diferentes cuerpos de agua en Getafe.

Título	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Recolección de muestras	2024-08-03	2024-08-03	Pendiente	Isabel Garcia	 
Análisis de laboratorio	2024-08-08	2024-08-08	Pendiente	Isabel Garcia	 
Interpretación de resultados	2024-08-13	2024-08-13	Pendiente	Isabel Garcia	 
Elaboración del informe	2024-08-18	2024-08-18	Pendiente	Isabel Garcia	 



## Proyecto: Recolección de muestras

h ilhjihiyhj

Título	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Recolección de muestras	2333-03-03	2333-04-02	Pendiente	Isabel Garcia	 









## Proyecto: Recolección de muestras

wwwwwwwwwww

Título	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Recolección de muestras	3333-03-03	3333-04-02	Pendiente	Isabel Garcia	 

## Proyecto: Come verga

Comer muchas vergas para comer, mmmm que rico

Título	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Marcos			En curso	Carlos Lopez	 
Marcoss			En curso	Carlos Lopez	 
Marcosss			En curso	Carlos Lopez	 
Marcos	2025-10-06	2025-04-04	En curso	Isabel Garcia	 

Cerrar sesión



## EDITAR TAREA

EDITAR LA TAREA

Inspeccionar tareas  
(solo supervisores)

Gestionar tareas  
(solo trabajadores)

Título:

Marcos

Descripción:

Marcos jodar gomez Yimenez Moratilla

Estado:

Pendiente

Inicio previsto:

dd/mm/aaaa

Fin previsto:

dd/mm/aaaa

Inicio real:

dd/mm/aaaa

Fin real:

dd/mm/aaaa

Guardar cambios

## ELIMINAR TAREA

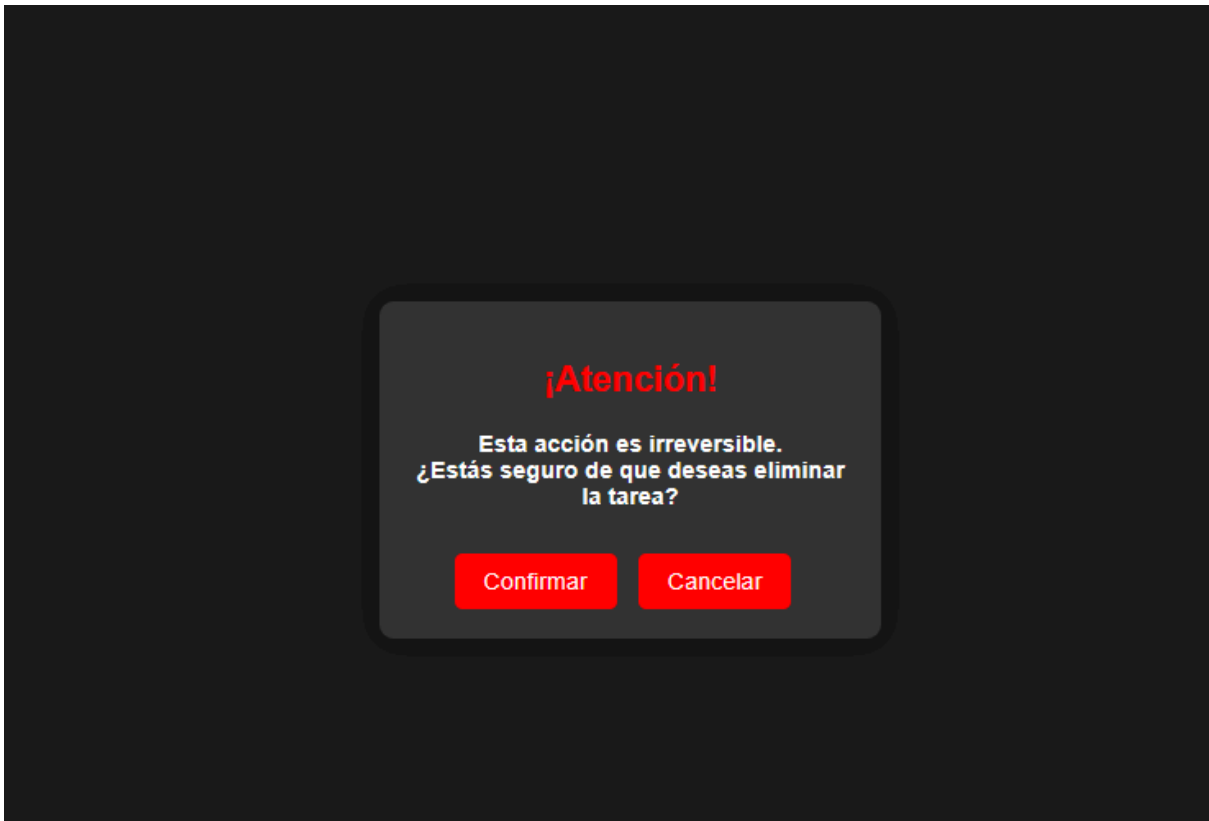
SELECCIONAMOS UNA TAREA Y LE DAMOS A ELIMINAR:

### Proyecto: Come verga

Comer muchas vergas para comer, mmmm que rico

Titulo	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Marcos			En curso	Carlos Lopez	 
Marcoss			En curso	Carlos Lopez	 
Marcos	2025-10-06	2025-04-04	En curso	Isabel Garcia	 





CONFIRMACIÓN:



REVISIÓN:

Proyecto: Come verga

Comer muchas vergas para comer, mmmm que rico

Título	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Marcos			En curso	Carlos Lopez	 
Marcos	2025-10-06	2025-04-04	En curso	Isabel Garcia	 

SE HA ELIMINADO CORRECTAMENTE

SUPERVISOR

## Supervisor

### Proyecto: Come verga

Comer muchas vergas para comer, mmmm que rico

Titulo	Inicio previsto	Fin previsto	Estado	A cargo de
Marcos			En curso	Carlos Lopez
Marcos	2025-10-06	2025-04-04	En curso	Isabel Garcia
Recolección de muestras	2222-02-22	2222-02-22	Pendiente	Isabel Garcia

### Proyecto: Análisis de Calidad del Agua en Barrio Salamanca, Madrid

Estudio detallado de la calidad del agua en el Barrio Salamanca de Madrid.

Titulo	Inicio previsto	Fin previsto	Estado	A cargo de
Recolección de muestras	2024-03-15	2024-03-15	Pendiente	Juan Gomez
Análisis de laboratorio	2024-03-20	2024-03-20	Pendiente	Juan Gomez
Interpretación de resultados	2024-03-25	2024-03-25	Pendiente	Juan Gomez
Elaboración del informe	2024-03-30	2024-03-30	Pendiente	Juan Gomez

### Proyecto: Evaluación de la Contaminación del Agua en Leganés

Investigación sobre la contaminación del agua en diferentes zonas de Leganés.

Titulo	Inicio previsto	Fin previsto	Estado	A cargo de
Recolección de muestras	2024-04-02	2024-04-02	Pendiente	Ana Martinez
Análisis de laboratorio	2024-04-07	2024-04-07	Pendiente	Ana Martinez
Interpretación de resultados	2024-04-12	2024-04-12	Pendiente	Ana Martinez
Elaboración del informe	2024-04-17	2024-04-17	Pendiente	Ana Martinez

### Proyecto: Estudio de Microorganismos en Aguas del Barrio de Chamartín

Estudio de la presencia y actividad de microorganismos en las aguas del Barrio de Chamartín.

### Proyecto: HITO NUMERO 2

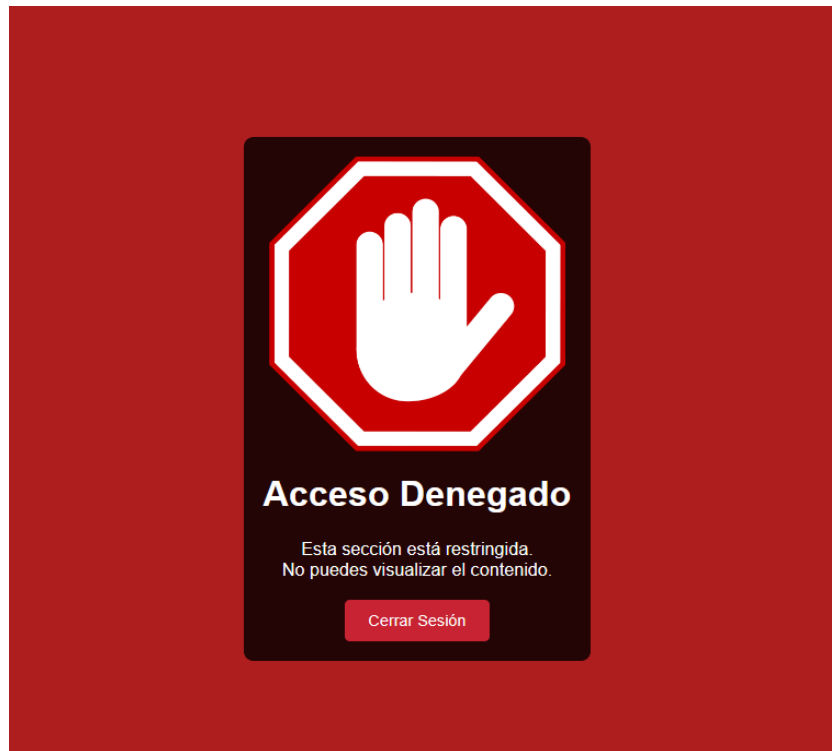
Este es un hito de gran importancia, valiendo mucho en la nota final

Titulo	Inicio previsto	Fin previsto	Estado	A cargo de
Recolección de muestras	2024-03-07	2024-04-06	Pendiente	Isabel Garcia
Análisis de laboratorio	2024-03-07	2024-05-06	Pendiente	Isabel Garcia
Interpretación de resultados	2024-03-07	2024-06-05	Pendiente	Isabel Garcia
Elaboración del informe	2024-03-07	2024-07-05	Pendiente	Isabel Garcia

Cerrar sesión


## DENEGADO

AL IR A GESTOR DE TAREAS O SUPERVISORES CON UN USUARIO ERRÓNEO:



## CREAR NUEVA TAREA

CREAMOS LA TAREA NUEVA:



La nueva tarea será añadida al proyecto Tareas Sueltas

Añade la nueva tarea

Título:

Descripción:

Estado:

Pendiente ▾

Inicio previsto:

22 / 02 / 2222

Fin previsto:

22 / 02 / 2222

Inicio real:

22 / 02 / 2222











Fin real:

22 / 02 / 2222

Guardar cambios


SE AÑADE AL PROYECTO:

Comer muchas vergas para comer, mmmm que rico

Titulo	Inicio previsto	Fin previsto	Estado	A cargo de	Acciones
Marcos			En curso	Carlos Lopez	 
Marcoss			En curso	Carlos Lopez	 
Marcosss			En curso	Carlos Lopez	 
Marcos	2025-10-06	2025-04-04	En curso	Isabel Garcia	 
Recolección de muestras	2222-02-22	2222-02-22	Pendiente	Isabel Garcia	 


CREAR NUEVO PROYECTO


CREAR NUEVO PROYECTO:





Creación de un nuevo proyecto

Se añadirán de manera predeterminada las siguientes tareas:

☒  Recolección de muestras

☒  Análisis de laboratorio

☒  Interpretación de resultados

☒  Elaboración del informe

Nombre del Proyecto:

HITO NUMERO 2

Descripción:

Este es un hito de gran importancia, valiendolo mucho en la nota final

Zona:

Madrid

Fecha:

07/03/2024

Guardar

SI NO RELLENAS LOS CAMPOS OBLIGATORIOS:



1. **Spring data jpa:** Proporciona las dependencias necesarias para trabajar con JPA (Java Persistence API), que es un estándar de la plataforma Java para el mapeo objeto-relacional y la gestión de entidades en una base de datos relacional.
2. **Spring security:** Proporciona las dependencias necesarias para la seguridad de la aplicación, incluida la autenticación y la autorización.
3. **Thymeleaf:** Proporciona las dependencias necesarias para integrar Thymeleaf, un motor de plantillas HTML para aplicaciones web en Spring Boot.
4. **Spring web:** Proporciona las dependencias necesarias para desarrollar aplicaciones web utilizando Spring MVC (Model-View-Controller).

## APPLICATION.PROPERTIES

---

```
server.port=8085
spring.datasource.url=jdbc:mysql://localhost:3306/magic_water
spring.datasource.username=root
spring.datasource.password=clase
spring.datasource.driver=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL8Dialect
spring.jpa.show-sql=true
```

APPLICATION.PROPERTIES contiene la configuración de propiedades para una aplicación Spring Boot. Estas propiedades son:

**server.port=8085:** Puerto en el que el servidor de la aplicación Spring Boot escuchará las solicitudes HTTP. Estando disponible en el puerto 8085.

**spring.datasource.url=jdbc:mysql://localhost:3306/magic\_water:** Define la URL de conexión a la bbdd MySQL. Establece la dirección del servidor de la bbdd (localhost), el puerto (3306) y el nombre de la base de datos (magic\_water).

**spring.datasource.username=root:** Establece el nombre de usuario de la bbdd para conectarse a ella, siendo **root**.

**spring.datasource.password=clase:** Establece la contraseña para autenticarse en la bbdd MySQL, siendo **clase**.

**spring.datasource.driver=com.mysql.cj.jdbc.Driver:** Dicta el nombre de la clase del controlador JDBC que se utilizará para la conexión con la base de datos MySQL.

**spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect:** Esta propiedad especifica el dialecto de Hibernate para interactuar con la base de datos MySQL.

**spring.jpa.show-sql=true:** Habilita la impresión de consultas SQL generadas por Hibernate en la consola de registro. Cuando está configurado en true, se mostrarán las consultas SQL en la consola de registro de la aplicación.

Básicamente, estas configuraciones definen la información de conexión a la base de datos MySQL y otras configuraciones de JPA e Hibernate para una aplicación Spring Boot.

## CONTROLADOR

---

```
package es.magicwater.controladores;

import es.magicwater.jpa.Proyecto;
import es.magicwater.jpa.Tarea;
import es.magicwater.jpa.Usuario;
import es.magicwater.repositorios.ProyectoRepositorio;
import es.magicwater.repositorios.TareaRepositorio;
import es.magicwater.repositorios.UsuarioRepositorio;
import jakarta.servlet.http.HttpServletRequest;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.ModelAndView;

import java.sql.Date;
import java.time.LocalDate;
import java.time.ZoneId;
import java.util.List;
import java.util.Optional;

@Controller
public class Controlador {
    @Autowired
    UsuarioRepositorio reUsuario;
    @Autowired
    TareaRepositorio reTarea;
    @Autowired
    ProyectoRepositorio reProyecto;
    @Autowired
    PasswordEncoder encoder;

    private Usuario devolverLogin(Authentication aut) {
        Usuario login;
        if (aut==null) { // No se ha iniciado sesión.
            login = null;
        }
        else { // Se ha iniciado sesión.
            Optional<Usuario> userOpt =
reUsuario.findById(aut.getName());
            login = userOpt.get();
        }
    }
}
```



```

    }
    return login;
}

@RequestMapping("/")
public ModelAndView petitionRaiz(Authentication aut) {
    ModelAndView mv = new ModelAndView();
    Usuario login = devolverLogin(aut);
    mv.addObject("login", login);
    mv.setViewName("index");
    return mv;
}

@RequestMapping("/login")
public ModelAndView petitionSesion() {
    ModelAndView mv = new ModelAndView();
    mv.setViewName("login");
    return mv;
}

@RequestMapping("/supervisor")
public ModelAndView petitionSupervisor() {
    ModelAndView mv = new ModelAndView();

    List<Proyecto> proyectos = reProyecto.findAll();
    mv.addObject("proyectos", proyectos);

    mv.setViewName("miron");
    return mv;
}

@RequestMapping("/trabajador")
public ModelAndView petitionTrabajador(Authentication aut)
{
    ModelAndView mv = new ModelAndView();
    Usuario login = devolverLogin(aut);

    List<Proyecto> proyectos =
reProyecto.proyectosUsuario(login.getNif());
    mv.addObject("proyectos", proyectos);

    mv.setViewName("gestortareas");
    return mv;
}

@RequestMapping("/denegado")
public ModelAndView petitionDenegado() {
    ModelAndView mv = new ModelAndView();
    mv.setViewName("denegado");
    return mv;
}

```

```

    }

    @RequestMapping("/perfil")
    public ModelAndView petitionPerfil(Authentication aut) {
        ModelAndView mv = new ModelAndView();
        Usuario login = devolverLogin(aut);
        mv.addObject("user", login);
        mv.setViewName("perfil");
        return mv;
    }

    @RequestMapping("/guardarperfil")
    public String petitionGuardarPerfil(Usuario u) {
        reUsuario.save(u);
        return "redirect:/";
    }

    @RequestMapping("/contra")
    public ModelAndView petitionCambiarContraseña() {
        ModelAndView mv = new ModelAndView();
        mv.setViewName("contra");
        return mv;
    }

    @RequestMapping("/guardarcontra")
    public String petitionGuardarContraseña(HttpServletRequest request, Authentication aut) {
        String contra = request.getParameter("pw1");
        Usuario login = devolverLogin(aut);
        login.setPassword(encoder.encode(contra));
        reUsuario.save(login);
        return "redirect:/perfil";
    }

    @RequestMapping("/trabajador/tarea/editar")
    public ModelAndView petitionEditarTarea(HttpServletRequest request) {
        ModelAndView mv = new ModelAndView();
        int id = Integer.parseInt(request.getParameter("id"));
        Optional<Tarea> t = reTarea.findById(id);
        Tarea tarea = t.orElse(null);
        // Retorna la tarea si está presente, sino null.
        mv.addObject("tarea", tarea);
        mv.setViewName("editartarea");
        return mv;
    }

    @PostMapping("/trabajador/tarea/editar/guardar")
    public String
    petitionGuardarTareaEditada(HttpServletRequest request) {
        // Por el momento nos limitamos a recoger los

```

```

parámetros del formulario.
    String id = request.getParameter("id");
    String titulo = request.getParameter("titulo");
    String descripcion =
request.getParameter("descripcion");
    String estado = request.getParameter("estado");
    String inicioprevisto =
request.getParameter("inicioprevisto");
    String finprevisto =
request.getParameter("finprevisto");
    String inicioreal =
request.getParameter("inicioreal");
    String finreal = request.getParameter("finreal");

    System.out.println("Id Tarea: " + id);
    System.out.println("Título: " + titulo);
    System.out.println("Descripción: " + descripcion);
    System.out.println("Estado: " + estado);
    System.out.println("Inicio previsto: " +
inicioprevisto);
    System.out.println("Fin previsto: " + finprevisto);
    System.out.println("Inicio real: " + inicioreal);
    System.out.println("Fin real: " + finreal);

    int idTarea = Integer.parseInt(id);
    Tarea tarea = reTarea.findById(idTarea).get();
    tarea.setTitulo(titulo);
    tarea.setDescripcion(descripcion);
    tarea.setEstado(estado);

    // Convierte las cadenas de fecha en objetos Date
    Date inicioPrevistoDate = (inicioprevisto.isEmpty()) ?
null : Date.valueOf(inicioprevisto);
    Date finPrevistoDate = (finprevisto.isEmpty()) ? null
: Date.valueOf(finprevisto);
    Date inicioRealDate = (inicioreal.isEmpty()) ? null :
Date.valueOf(inicioreal);
    Date finRealDate = (finreal.isEmpty()) ? null :
Date.valueOf(finreal);

    tarea.setInicioprevisto(inicioPrevistoDate);
    tarea.setFinprevisto(finPrevistoDate);
    tarea.setInicioreal(inicioRealDate);
    tarea.setFinreal(finRealDate);

    // Las fechas deberían ser coherentes entre ellas y
con el estado.
    // Esto te toca arreglarlo a tí.

    // Guardamos los cambios.

```

```

        reTarea.save(tarea);

        return "redirect:/trabajador";
    }
    @GetMapping("/trabajador/tarea/eliminar/{id}")
    public ModelAndView confirmarEliminarTarea(@PathVariable
int id) {
        ModelAndView modelAndView = new
ModelAndView("eliminar");
        modelAndView.addObject("id", id);
        return modelAndView;
    }

    // Método para eliminar la tarea
    @GetMapping("/trabajador/tarea/eliminar/confirmar/{id}")
    public String eliminarTarea(@PathVariable int id) {
        Optional<Tarea> tareaOptional = reTarea.findById(id);

        if (tareaOptional.isPresent()) {
            Tarea tarea = tareaOptional.get();
            reTarea.delete(tarea);
        }
        return "redirect:/trabajador";
    }

    @GetMapping("/trabajador/tarea/nueva")
    public ModelAndView peticionNuevaTarea(HttpServletRequest
request) {
        ModelAndView mv = new ModelAndView();
        Tarea t = new Tarea();
        mv.addObject("tarea", t);
        mv.setViewName("nuevatarea");
        return mv;
    }
    @PostMapping("/trabajador/tarea/nueva/guardar")
    public String peticionGuardarTareaNueva(HttpServletRequest
request, Authentication aut) {
        String titulo = request.getParameter("titulo");
        String descripcion =
request.getParameter("descripcion");
        String estado = request.getParameter("estado");
        String inicioprevisto =
request.getParameter("inicioprevisto");
        String finprevisto =
request.getParameter("finprevisto");
        String inicioreal =
request.getParameter("inicioreal");
        String finreal = request.getParameter("finreal");
    }

```

```

        Usuario login = devolverLogin(aut);
        Tarea tarea = new Tarea();
        tarea.setIdtarea(reTarea.newIdTarea());
        tarea.setUsuario(login);
        tarea.setTitulo(titulo);
        tarea.setDescripcion(descripcion);
        tarea.setEstado(estado);

        // Es una tarea suelta que será asignada al proyecto 0
        (tareas sueltas).
        Proyecto p = reProyecto.findById(0).get();
        tarea.setProyecto(p);

        Date inicioPrevisto = (inicioprevisto.isEmpty()) ?
null : Date.valueOf(inicioprevisto);
        Date finPrevisto = (finprevisto.isEmpty()) ? null :
Date.valueOf(finprevisto);
        tarea.setFinprevisto(inicioPrevisto);
        tarea.setInicioprevisto(finPrevisto);

        Date inicioReal = (inicioreal.isEmpty()) ? null :
Date.valueOf(inicioreal);
        Date finReal = (finreal.isEmpty()) ? null :
Date.valueOf(finreal);
        tarea.setInicioreal(inicioReal);
        tarea.setFinreal(finReal);

        // Las fechas deberían ser coherentes entre ellas y
        con el estado.
        // Esto te toca arreglarlo a tí.
        reTarea.save(tarea);

        return "redirect:/trabajador";
    }

    @GetMapping("/trabajador/proyecto/nuevo")
    public ModelAndView
peticionNuevoProyecto(HttpServletRequest request) {
        ModelAndView mv = new ModelAndView();
        Proyecto p = new Proyecto();
        mv.addObject("proyecto", p);
        mv.setViewName("nuevoproyecto");
        return mv;
    }

    @RequestMapping(value =
"/trabajador/proyecto/nuevo/guardar", method =
RequestMethod.POST)
    public String
peticionGuardarProyectoNuevo(HttpServletRequest request,
Authentication aut) {

```

```

        String nombre = request.getParameter("nombre");
        String descripcion =
request.getParameter("descripcion");
        String zona = request.getParameter("zona");
        String fecha = request.getParameter("fecha");

        Proyecto proyecto = new Proyecto();
        proyecto.setIdproyecto(reProyecto.newIdProyecto());
        proyecto.setNombre(nombre);
        proyecto.setDescripcion(descripcion);
        proyecto.setZona(zona);

        Date fechaDate = (fecha.isEmpty()) ? null :
Date.valueOf(fecha);
        proyecto.setFecha(fechaDate);
        reProyecto.save(proyecto);

        Usuario login = devolverLogin(aut);

        LocalDate fechaLd =
LocalDate.from(fechaDate.toLocalDate().atStartOfDay());

        // Creación de la tarea predeterminada Recolección de
muestras
        Tarea tarea1 = new Tarea();
        tarea1.setIdtarea(reTarea.newIdTarea());
        tarea1.setTitulo("Recolección de muestras");
        tarea1.setDescripcion("Recolección de muestras en " +
zona + " para proyecto " + fecha);
        tarea1.setEstado("Pendiente");
        tarea1.setUsuario(login);
        tarea1.setProyecto(proyecto);
        tarea1.setInicioprevisto(fechaDate);

tarea1.setFinprevisto(Date.valueOf(fechaLd.plusDays(30)));
        reTarea.save(tarea1);

        // Creación de la tarea predeterminada Análisis de
laboratorio
        Tarea tarea2 = new Tarea();
        tarea2.setIdtarea(reTarea.newIdTarea());
        tarea2.setTitulo("Análisis de laboratorio");
        tarea2.setDescripcion("Análisis de laboratorio en " +
zona + " para proyecto " + fecha);
        tarea2.setEstado("Pendiente");
        tarea2.setUsuario(login);
        tarea2.setProyecto(proyecto);
        tarea2.setInicioprevisto(fechaDate);

tarea2.setFinprevisto(Date.valueOf(fechaLd.plusDays(60)));

```

```

        reTarea.save(tarea2);

        // Creación de la tarea predeterminada Interpretación
de resultados
        Tarea tarea3 = new Tarea();
        tarea3.setIdtarea(reTarea.newIdTarea());
        tarea3.setTitulo("Interpretación de resultados");
        tarea3.setDescripcion("Interpretación de resultados en
" + zona + " para proyecto " + fecha);
        tarea3.setEstado("Pendiente");
        tarea3.setUsuario(login);
        tarea3.setProyecto(proyecto);
        tarea3.setInicioprevisto(fechaDate);

tarea3.setFinprevisto(Date.valueOf(fechaLd.plusDays(90)));
        reTarea.save(tarea3);

        // Creación de la tarea predeterminada Elaboración del
informe
        Tarea tarea4 = new Tarea();
        tarea4.setIdtarea(reTarea.newIdTarea());
        tarea4.setTitulo("Elaboración del informe");
        tarea4.setDescripcion("Elaboración de informe en " +
zona + " para proyecto " + fecha);
        tarea4.setEstado("Pendiente");
        tarea4.setUsuario(login);
        tarea4.setProyecto(proyecto);
        tarea4.setInicioprevisto(fechaDate);

tarea4.setFinprevisto(Date.valueOf(fechaLd.plusDays(120)));
        reTarea.save(tarea4);

        return "redirect:/trabajador";
    }
}

```

Esto es un **Controlador de Spring MVC** para una aplicación web.

Un **Controlador** gestiona las solicitudes HTTP del cliente y devuelve respuestas. Conecta las vistas con la lógica de la aplicación, procesando la entrada del usuario y generando la salida correspondiente. Básicamente, dirige el flujo de la aplicación web.

Definimos varios métodos que manejan diferentes peticiones HTTP, que realizan distintas funciones de esas peticiones.

#### **Métodos y Funciones:**

**Anotación @Controller:** Indica que es un controlador de Spring MVC.

**@Autowired:** Inyectan instancias de los repositorios y el PasswordEncoder de la clase Seguridad automáticamente.

**devolverLogin(Authentication aut):** Método que devuelve el objeto Usuario correspondiente al usuario autenticado actualmente. Si no hay usuario autenticado, devuelve `null`.

**peticionRaiz(Authentication aut):** Maneja las peticiones a la ruta raíz ( "/" ) . Agrega el usuario autenticado al modelo y devuelve una vista llamada "index" del HTML, la que será nuestra interfaz principal.

**peticionSesion():** Maneja las peticiones a la ruta "/login". Devolviendo la interfaz "login".

**peticionSupervisor():** Maneja las peticiones a la ruta "/supervisor". Obtiene todos los proyectos y los agrega al modelo,devolviendo la interfaz "miron", es decir, Supervisor.

**peticionTrabajador(Authentication aut):** Maneja las peticiones de la ruta "/trabajador". Obtiene los proyectos del usuario autenticado y los agrega al modelo,Devolviendo la interfaz "gestortareas".

**peticionDenegado():** Maneja las peticiones a la ruta "/denegado". Devolviendo la interfaz "denegado".

**peticionPerfil(Authentication aut):** Maneja las peticiones a la ruta "/perfil". Obtiene el usuario autenticado y lo agrega al modelo, devolviendo la interfaz "perfil".

**peticionGuardarPerfil(Usuario u):** Maneja las peticiones para guardar el perfil del usuario. Guarda el usuario proporcionado y la redirige a la página principal.

**peticionCambiarContraseña():** Maneja las peticiones para cambiar la contraseña. Llevándonos a la interfaz "contra", donde se cambia de contraseña.

**peticionGuardarContraseña(HttpServletRequest request, Authentication aut):** Maneja las peticiones para guardar la contraseña del usuario autenticado. Codifica esa contraseña y guarda el usuario, redirigiendonos al perfil del usuario("perfil").

**peticionEditarTarea(HttpServletRequest request):** Maneja las peticiones para editar una tarea. Lo hace mediante el ID de la tarea, obtiene la tarea según el ID de la solicitud y la agrega al modelo, luego devuelve una vista llamada "editartarea".

## JPA

Usuario:

```
package es.magicwater.jpa;

import jakarta.persistence.*;
import java.util.List;

@Entity
public class Usuario {

    @Id
    private String nif;

    private byte activo;

    private String apellidos;
```



```
private String categoria;

private String email;

private String nombre;

private String password;

private String permiso;

private String tlf;

//bi-directional many-to-one association to Tarea
@OneToMany(mappedBy="usuario")
private List<Tarea> tareas;

public Usuario() {
}

public String getNif() {
    return this.nif;
}

public void setNif(String nif) {
    this.nif = nif;
}

public byte getActivo() {
    return this.activo;
}

public void setActivo(byte activo) {
    this.activo = activo;
}

public String getApellidos() {
    return this.apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

public String getCategoria() {
    return this.categoria;
}

public void setCategoria(String categoria) {
    this.categoria = categoria;
}

public String getEmail() {
    return this.email;
}
}
```

```
public void setEmail(String email) {
    this.email = email;
}

public String getNombre() {
    return this.nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getPassword() {
    return this.password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getPermiso() {
    return this.permiso;
}

public void setPermiso(String permiso) {
    this.permiso = permiso;
}

public String getTlf() {
    return this.tlf;
}

public void setTlf(String tlf) {
    this.tlf = tlf;
}

public List<Tarea> getTareas() {
    return this.tareas;
}

public void setTareas(List<Tarea> tareas) {
    this.tareas = tareas;
}

public Tarea addTarea(Tarea tarea) {
    getTareas().add(tarea);
    tarea.setUsuario(this);

    return tarea;
}

public Tarea removeTarea(Tarea tarea) {
    getTareas().remove(tarea);
    tarea.setUsuario(null);
}
```

```

        return tarea;
    }
}

```

**Anotación @Entity:** Marca la clase como una entidad de persistencia, haciendo que los objetos de la clase se puedan almacenar en una base de datos.

**@Id:** Marca el campo nif como la PK(clave primaria) de la tabla en la base de datos.

**@OneToMany(mappedBy="usuario"):** Anotación que establece una relación bidireccional entre la entidad Usuario y la entidad Tarea. Haciendo que el usuario pueda tener múltiples tareas asociadas. **MappedBy** indica el nombre del campo en la clase Tarea que mapea esta relación.

**List<Tarea> tareas:** Lista de objetos Tarea asociados al usuario.

**getters y setters:** Bueno, con estos métodos accederemos y modificaremos los atributos del Usuario. Como con `getNombre()` y `setNombre()`.

**addTarea() y removeTarea():** Métodos para agregar y eliminar tareas de la lista de tareas asociadas al usuario.

La clase Usuario representa la entidad **usuario** de la bbdd que almacena información sobre los usuarios del sistema. Teniendo varios atributos que lo forman. Además establece una relación uno a muchos con la entidad Tarea, haciendo que un usuario pueda tener múltiples tareas asociadas.

## REPOSITORIOS

### ProyectoRepositorio:

```

@Repository
public interface ProyectoRepositorio extends
JpaRepository<Proyecto, Integer> {
    @Query("SELECT p FROM Proyecto p INNER JOIN Tarea t ON
p.idproyecto = t.proyecto.idproyecto WHERE t.usuario.nif = ?1")
    List<Proyecto> proyectosUsuario(String nif);

    @Query("SELECT MAX(p.idproyecto) + 1 FROM Proyecto p")
    Integer newIdProyecto();
}

```

**@Repository:** Anotación para indicar que es un repositorio de datos.

**ProyectoRepositorio extends JpaRepository<Proyecto, Integer>:** Repositorio para Proyecto, utilizando Spring Data JPA. El parámetro `<Proyecto, Integer>` indica que la entidad Proyecto es la clase de la entidad que maneja este repositorio, e Integer es el tipo de dato de su clave primaria.

**1 @Query():** Consulta para obtener proyectos asociados a un usuario.

**@Query("SELECT MAX(p.idproyecto) + 1 FROM Proyecto p") Integer newIdProyecto():** Consulta para obtener el siguiente ID disponible para un nuevo proyecto.

TareaRepositorio:

```
public interface TareaRepositorio extends JpaRepository<Tarea, Integer> {  
  
    @Query("SELECT MAX(t.idtarea) + 1 FROM Tarea t")  
    Integer newIdTarea();  
}
```

### @Repository

**extends JpaRepository<Tarea, Integer>:** Repositorio para Tarea, utilizando Spring Data JPA.

**@Query("SELECT MAX(t.idtarea) + 1 FROM Tarea t"):** Consulta que busca el máximo ID de tarea existente y devuelve el ID disponible para ser asignado a una nueva tarea.

**Integer newIdTarea():** Este método declara una consulta personalizada para obtener el próximo ID disponible para una nueva tarea. Retorna un entero que representa el próximo ID de tarea disponible.

UsuarioRepositorio:

```
@Repository  
public interface UsuarioRepositorio extends  
JpaRepository<Usuario, String> {  
  
}
```

### @Repository

**JpaRepository<Usuario, String>:** Repositorio para Usuario, utilizando Spring Data JPA.

## SEGURIDAD

---

```
package es.magicwater.seguridad;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import  
org.springframework.security.config.annotation.web.builders.H  
ttpSecurity;
```

```

import
org.springframework.security.config.annotation.web.configurers
s.AbstractHttpConfigurer;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.provisioning.JdbcUserDetailsManager;
import
org.springframework.security.provisioning.UserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

import javax.sql.DataSource;

@Configuration
public class DatabaseWebSecurity {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UserDetailsManager users(DataSource dataSource) {
        JdbcUserDetailsManager users = new
JdbcUserDetailsManager(dataSource);
        users.setUsersByUsernameQuery("select nif, password,
activo from usuario where nif=?");
        users.setAuthoritiesByUsernameQuery("select nif,
permiso from usuario where nif=?");
        return users;
    }

    // Filtros por URL.
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
        http.csrf(AbstractHttpConfigurer::disable);

        http.authorizeHttpRequests(auth -> auth
            .requestMatchers("/css/**").permitAll()
            .requestMatchers("/imagenes/**").permitAll()
            .requestMatchers("/", "/login",
"/signup").permitAll()
            .requestMatchers("/trabajador/**").hasAuthority("TRABAJADOR")

```

```

.requestMatchers("/supervisor/**").hasAuthority("SUPERVISOR")
    .anyRequest().authenticated();
    http.formLogin(formLogin ->
formLogin.loginPage("/login").permitAll());
    http.logout(logout ->
logout.logoutUrl("/logout").logoutSuccessUrl("/").permitAll()
);
    http.exceptionHandling((exception)->
exception.accessDeniedPage("/denegado"));

    return http.build();
}
}

```

Esta clase **DatabaseWebSecurity** configura la seguridad de la aplicación web utilizando **Spring Security**, define reglas de autorización y autenticación, y proporciona beans para la gestión de usuarios y la codificación de contraseñas.

**Anotación @Configuration:** Indica que esta clase proporciona la configuración para la aplicación Spring.

**@Bean:** Define los beans para el codificador de contraseñas (**passwordEncoder**) y el gestor de detalles de usuario (**users**).

**PasswordEncoder:** Bean que proporciona un codificador de contraseñas, **BCrypt**, utilizado para codificar y verificar las contraseñas de los usuarios.

**UserDetailsManager:** El bean users es un gestor de detalles de usuario que utiliza una fuente de datos JDBC (DataSource) para gestionar los usuarios. Se configuran consultas SQL personalizadas para obtener detalles de usuarios y sus autoridades.

**SecurityFilterChain:** Bean que indica la configuración de seguridad para las solicitudes HTTP. Define reglas de acceso de los URL, configura el inicio de sesión y el cierre de sesión, el manejo de excepciones y la configuración CSRF (Cross-Site Request Forgery). CSRF es un ataque web que aprovecha la sesión activa de un usuario autenticado para realizar acciones no autorizadas.

**HttpSecurity:** Permite configurar la seguridad HTTP. Se utiliza para definir las reglas de autorización y autenticación para las solicitudes HTTP.

**http.csrf(AbstractHttpConfigurer::disable):** Deshabilita la protección CSRF.

**http.authorizeHttpRequests():** Define reglas de autorización para las solicitudes HTTP. Permite o restringe el acceso a las URL según el el usuario.

**http.formLogin():** Configura la autenticación basada en formularios y establece la página de inicio de sesión.

**http.logout():** Configura la funcionalidad de cierre de sesión y establece la URL de redirección después del cierre de sesión.

**http.exceptionHandling():** Configura el manejo de excepciones, redirigiéndose a la página de denegación de acceso si se produce un error de acceso no autorizado.

# FRONT-END

En los HTML creados, hemos utilizado principalmente Bootstrap, Thymeleaf y CSS para configurar el aspecto y la funcionalidad de las páginas web.

**Bootstrap** nos permite agregar herramientas de código abierto con sus componentes predefinidos de *CSS* y *JavaScript*. para desarrollar sitios web sencillos y responsivos, es decir, se adapta automáticamente para proporcionar una experiencia óptima al usuario en cualquier tipo de dispositivo.

**Thymeleaf** nos permite integrar datos dinámicos en las páginas HTML.

**CSS** nos permite definir el aspecto y el diseño visual de la página.

Juntas, estas tecnologías permiten crear una experiencia web dinámica y atractiva para los usuarios.

**HOME  
(INDEX)**

¿Cómo utilizamos Thymeleaf?

```
<div style="max-width: 500px; margin: 0 auto;">
  <div class="div-parrafo">
    <p th:if="{login == null}" class="parrafo2">Usuario: sin
inicio de sesión</p>
    <p th:if="{!(login == null)}">
      Usuario:
      <span th:text="{login.getNif()}"></span>
      <br>
      <span th:text="{login.getNombre()}"></span>
      <span th:text="{login.getApellidos()}"></span>
      <br>
      <span th:text="{login.getCategoria()}"></span>
    </p>
  </div>
</div>

<p style="text-align: left;">
  <a th:if="{login == null}" th:href="{@{/login}}"
class="parrafo2 boton" style="text-align: center;">Iniciar
sesión</a>
  <a th:if="{!(login == null)}" th:href="{@{/logout}}"
```

```

class="parrafo2 boton">Cerrar sesión</a>
  <a th:if="{!(login==null)}" th:href="{@{/perfil}}"
class="parrafo2 boton">Perfil</a>
</p>

```

- `th:if="{login == null}"`: Esta línea condicional de Thymeleaf se utiliza para verificar si el objeto login es nulo. Si es nulo, se muestra el mensaje *"Usuario: sin inicio de sesión"*.
- `th:if="{!(login == null)}"`: Esta línea condicional verifica si el objeto login no es nulo. Si no es nulo, se muestran detalles sobre el usuario.
- `th:text="{login.getNif()}"`: Aquí, `th:text` se usa para establecer el texto del elemento HTML en el valor devuelto por `login.getNif()`. Lo mismo se aplica a otras propiedades del objeto login, como nombre, apellidos y categoría.
- `th:href="{@{/login}}"`: línea utilizada para establecer la URL del enlace login. El prefijo `@{ }` se utiliza para definir una URL relativa. Así establece la ruta definida en el **Controlador**

Thymeleaf reemplaza los valores correspondientes antes de que se envíe la respuesta HTML al cliente, permitiendo la generación dinámica de contenido HTML basado en datos proporcionados por el servidor.

## ¿Cómo funciona el CSS?

```

<style>
  body {
    font-family: 'Arial', sans-serif;
    margin: 20px;
    padding: 20px;
    background-color: white;
  }
</style>

```

En el código, hay una sección `<style>` que contiene las reglas CSS para estilizar los elementos HTML. Esta sección se encuentra dentro del elemento `<head>` del HTML. Las reglas CSS creadas aquí se aplicarán a los elementos HTML correspondientes cuando se renderice la página en un navegador web. Esto es mediante que el HTML utiliza `class="css"` para comunicarse con el CSS. Y se aplica en el código CSS: `.css{estilos}` o `#css{}`

**Para conectar con Bootstrap:** Dentro del `<head>` añadimos este código:

```

<link
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css
/bootstrap.min.css" rel="stylesheet"
integrity="sha384-/Y6pD6FV/Vv2HJnA6t+vs1U6fwYXjCFtcEpHbNJ0lyAF
sXTsjBbfaDjzALeQsN6M" crossorigin="anonymous">

```



# CONCLUSIÓN

El desarrollo de la aplicación de gestión de tareas para MagicWater representa un paso significativo hacia la mejora de la eficiencia operativa y la organización interna de la empresa. Una vez cumplidas las especificaciones, la aplicación no solo facilita la asignación y el seguimiento de tareas, sino que también permite una mejor coordinación entre los equipos y una supervisión más efectiva por parte de los supervisores. Con la capacidad de automatizar tareas estándar, la aplicación se ha convertido en una herramienta invaluable para garantizar la calidad y la eficacia de los proyectos de evaluación de la calidad del agua de Magic Water. En última instancia, este proyecto ilustra el compromiso de la empresa con la innovación y la excelencia en su campo, sentando las bases para un futuro de éxito continuo y crecimiento sostenible.



# BIBLIOGRAFIA

**Amelia, (29-feb-2024). Magic Water versión 3. [consultado el 29 de febrero de 2024].**

**Disponible en:**

<https://classroom.google.com/c/NjE1MjUxMzQxMjU4/a/NjY2MzY1NTc0NTU3/details>

**w3schools, (15-ene-2024). CSS Tutorial. [Consultado el 1 de marzo de 2024]. Disponible en:**

<https://www.w3schools.com/Css/default.asp>

**Openla, (ene-2022). Chat GPT. [Consultado el 1 de marzo de 2024]. Disponible en:**

<https://chat.openai.com/?sso=>