Anexo V: Documentación técnica

Trabajo de Fin de Grado de Ingeniería Informática

Julio de 2024

Autor:

Diego Plata Klingler

Tutores:

Sergio Alonso Rollán

Javier Prieto Tejedor

Tabla de contenido

Índice de figuras

1. Introducción

El objetivo de este anexo es entrar en detalle en el código y la estructura del proyecto. Esto se hace para que cualquiera que quiera trabajar/mantener con él le sea más fácil.

El proyecto se puede desglosar en dos partes principales, que son el Dispositivo y el Servidor. Dentro del Servidor se pueden diferenciar dos partes distintas, el Frontend y el Backend. El Frontend será lo que vea el usuario a través de las interfaces gráficas. El Backend, en cambio, será lo que no es visible al usuario. Se realizarán todas las tareas de procesamiento y lógica del servidor, como la gestión de usuarios o la gestión de la base de datos.

Cabe resaltar un componente más, aunque no se sitúa en el mismo nivel que los mencionados anteriormente. Se trata de Docker [1], que dockerizará [2] la base de datos no relacional (MongoDB) con las mediciones de los sensores y el bróker MQTT que permite la comunicación entre Servidor y Dispositivo.

2. Estructura del sistema

Este apartado describe en detalle la arquitectura general del sistema, los módulos principales y sus interacciones, así como los patrones de diseño y tecnologías empleadas. La estructura del sistema proporciona una visión que facilita la comprensión entre los distintos componentes del software.

2.1 Servidor

En este apartado se detallarán las partes que componen el servidor.

2.1.1 Frontend

El servidor web es una parte fundamental del sistema, donde se realizan la mayoría de las interacciones entre el usuario y el sistema. Para el desarrollo del frontend del servidor, se han utilizado principalmente HTML y CSS. HTML (HyperText Markup Language) es el estándar para estructurar el contenido en la web, mientras que CSS (Cascading Style Sheets) se utiliza para controlar el estilo y la presentación del contenido. En la siguiente figura se puede ver cómo están distribuidos los ficheros dentro del servidor:

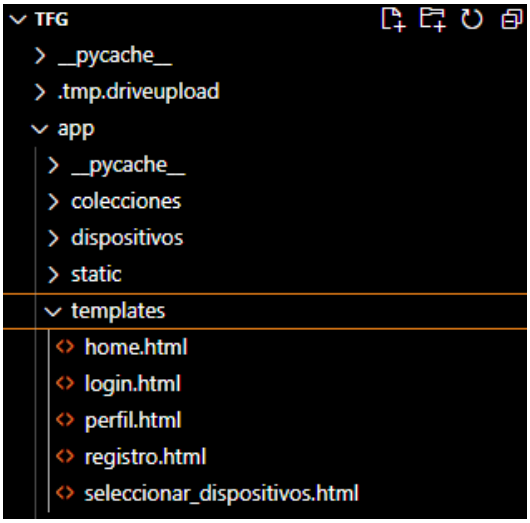


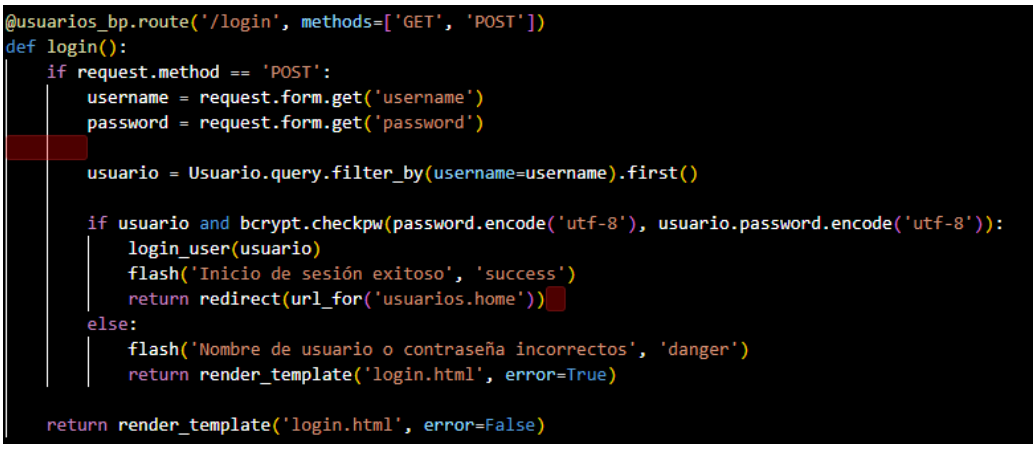
Figura 1: Estructura de la vista

En la carpeta *templates* podemos encontrar todos los HTML que serán visibles para el usuario.

- **Home.html**: Página principal del sistema. En ella se muestra la predicción meteorológica de la ciudad seleccionada, además de la lista de los dispositivos que el usuario ha registrado.
- **Login.html:** Pantalla de inicio de sesión del sistema.
- **Perfil.html:** Página del perfil del usuario. En ella puede modificar sus datos personales, los dispositivos registrados o eliminar su cuenta.
- **Registro.html:** Página de registro de usuarios del sistema.
- **Seleccionar_dispositivos.html:** Página que aparece tras registrar por primera vez una cuenta. En ella se mostrará una lista con los dispositivos disponibles en el sistema.

2.1.2Backend

Es el propio servidor que se encarga de realizar ciertos cálculos necesarios para el correcto funcionamiento del sistema, entre los que se encuentran los cálculos para las predicciones o la validación de los datos. La estructura del servidor es bastante sencilla, ya que se ha dividido en paquetes, cada uno con su funcionalidad. Dentro de estos paquetes tenemos un fichero con los *endpoints* del servidor. Un *endpoint* es una *url* a la que se hacen peticiones para interactuar con el servidor. Un ejemplo puede ser el *endpoint* de */login*, que inicia sesión en una cuenta con los datos introducidos en el formulario.



```python
@usuarios_bp.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        usuario = Usuario.query.filter_by(username=username).first()

        if usuario and bcrypt.checkpw(password.encode('utf-8'), usuario.password.encode('utf-8')):
            login_user(usuario)
            flash('Inicio de sesión exitoso', 'success')
            return redirect(url_for('usuarios.home'))
        else:
            flash('Nombre de usuario o contraseña incorrectos', 'danger')
            return render_template('login.html', error=True)

    return render_template('login.html', error=False)
```

Figura 2: Ejemplo de endpoint

Para la documentación de la API del Backend del servidor se ha usado una herramienta de documentación de APIs llamada Swagger [3]. Esta herramienta permite documentar APIs de manera sencilla, permitiendo probar los distintos *endpoints* con valores de prueba. Para acceder a la documentación generada por Swagger hay que acceder a la siguiente dirección: `http://localhost:5000/apidocs/#/`

Lo que veremos será algo así:

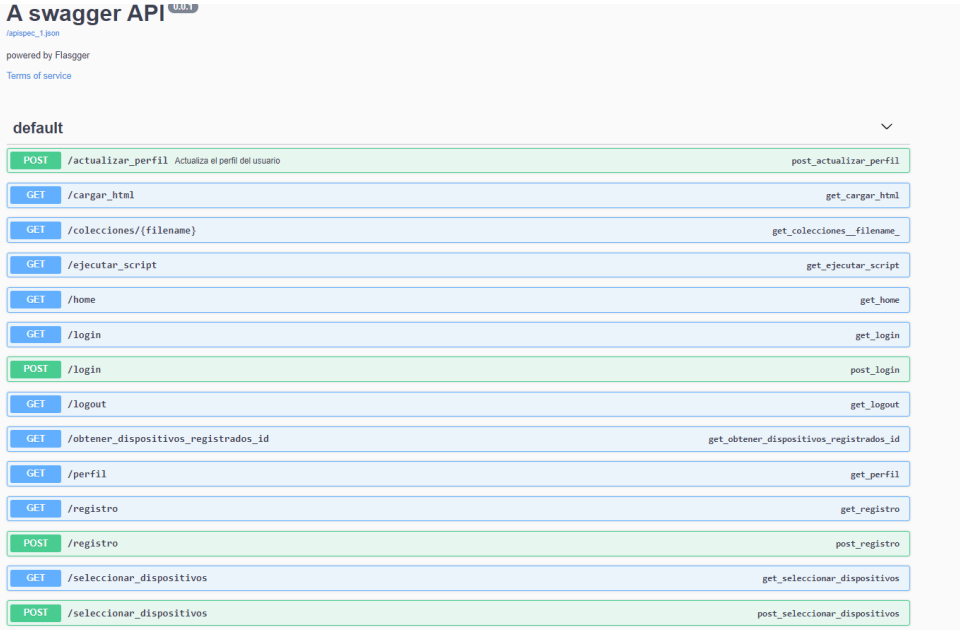


Figura 3: Vista Swagger

2.2 Dispositivo

Por otro lado, tenemos el dispositivo. La estructura del dispositivo es muy sencilla, ya que únicamente se trata de un código escrito en Arduino, y que realiza toda la parte de recogida de datos de los sensores, envío mediante MQTT, y recepción de mensajes con las órdenes de riego.

El código de Arduino configura varios componentes: un caudalímetro para medir la cantidad de agua regada, un sensor capacitivo para medir la humedad del suelo, un sensor de humedad y temperatura DHT22, y un relé para controlar la bomba de agua. La conexión a la red WiFi permite al dispositivo comunicarse con un bróker MQTT, enviando periódicamente los datos recopilados y recibiendo mensajes para iniciar el riego. La estructura del programa se basa en dos funciones principales: setup() y loop(). En setup(), el dispositivo se conecta a la red WiFi, configura el bróker MQTT y suscribe el dispositivo a los temas relevantes. Además, se inicializan los pines del sensor y del relé.

Durante la ejecución en loop(), el dispositivo envía los datos de los sensores cada diez segundos al tema MQTT correspondiente. Cuando se recibe una orden de riego, el dispositivo activa el relé, iniciando el flujo de agua y midiendo la cantidad hasta alcanzar el objetivo especificado. Este proceso es monitoreado y controlado mediante estados para asegurar que se realice de manera eficiente y se detenga una vez que se haya movido el volumen de agua requerido.

2.3 Docker

Docker es una plataforma de virtualización a nivel de sistema operativo que permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker se puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno sabiendo que su código se ejecutará.

Docker se basa en varios componentes, entre los que destacan las imágenes, los contenedores y los *Dockerfiles*. Las imágenes Docker son plantillas de solo lectura que contienen el sistema operativo, el entorno de ejecución y la aplicación que se va a ejecutar. Los contenedores Docker, por otro lado, son instancias ejecutables de imágenes Docker. Los contenedores contienen todo lo necesario para ejecutar una aplicación y pueden ser iniciados, detenidos, movidos o eliminados. Un *Dockerfile* es un archivo de texto que contiene una serie de instrucciones sobre cómo construir una imagen Docker. Define el entorno y los pasos necesarios para instalar las dependencias y la aplicación.

En este proyecto se ha usado un Docker para usar una base de datos MongoDB para guardar las mediciones de los sensores, y un bróker MQTT para la comunicación entre dispositivo y servidor, ya que Docker nos permite crear entornos portátiles de manera eficiente.

Para la creación de la imagen hemos usado un *Dockerfile* con las instrucciones necesarias para que funcionen ambas cosas. Primero hemos seleccionado una imagen de Ubuntu que soporte tanto MongoDB como Mosquitto-MQTT. Sobre esta, hemos realizado las instalaciones necesarias para hacer funcionar los scripts Python que tenemos, mediante el gestor de paquetes pip. Por último, hemos copiado los ficheros desarrollados por nosotros y los configuramos para que cada vez que se arranque el contenedor estén funcionando.

Con la imagen creada, solo habría que crear un contenedor que permita la ejecución en cualquier entorno. Los comandos utilizados son:

```
docker build -t imagen_tfg .

docker run --name contenedor_tfg -p 1883:1883 -p 27017:27017
```

Con estos comandos se crea y ejecuta un contenedor que podrá ser ejecutado en cualquier entorno.

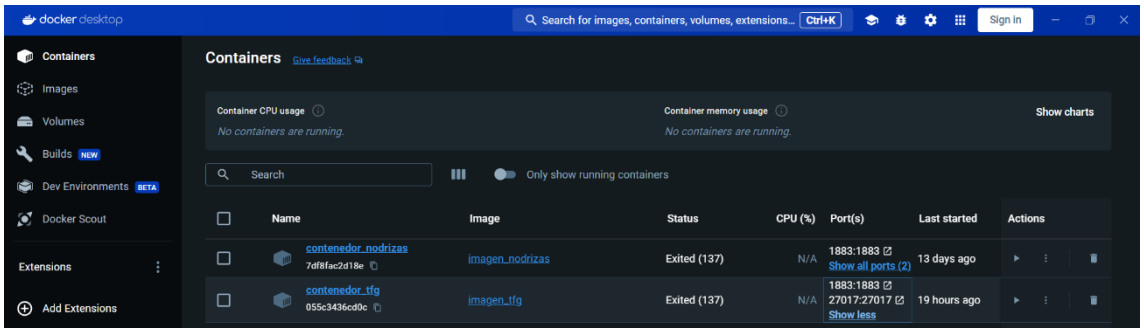


Figura 4: Docker desktop

3. Instalación

Para poder instalar y probar el proyecto en otro ordenador, habría que instalar las dependencias indicadas en el requirements.txt mediante el comando:

```
pip install -r requirements.txt
```

Con esto podríamos probar el servidor en nuestro ordenador, pero falta por instalar el dispositivo y el Docker. Para el dispositivo habría que tener instalado Docker en nuestro sistema y crear un contenedor mediante la imagen con el comando proporcionado en el anterior apartado. Para la instalación del dispositivo habría que tener descargado Arduino en nuestro dispositivo y cargar el código proporcionado.

Es muy probable que haya que cambiar las direcciones IP del bróker.

4. Referencias

[1] «Docker: Accelerated Container Application Development». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://www.docker.com/

[2] J. A. Castañeda, C. I. O. Álvarez, y J. C. I. G. Ibarra, «DOCKERIZANDO UN LABORATORIO VIRTUAL DE PROGRAMACIÓN (VPL) Y MOODLE EN GOOGLE CLOUD», *Encuentro Int. Educ. En Ing.*, ago. 2019, doi: 10.26507/ponencia.212.

[3] «API Documentation & Design Tools for Teams | Swagger». Accedido: 3 de julio de 2024. [En línea]. Disponible en: https://swagger.io/