TRABAJO DE GRADO
Opción Seminario-Diplomado.

Sistema de Monitoreo Agrícola Inteligente SmartField

Corporación Universitaria Remington.
Facultad de Ingeniería
Ingeniería en Sistemas

Danny Fernando Rodríguez Rodríguez
Tutor: Diego Fernando Marín Lozano

Opción de Trabajo de grado Seminario-Diplomado
2025

Tabla de Contenido

Resumen

El presente trabajo es el desarrollo e implementación de una solución a la necesidad de modernizar, automatizando la práctica agrícola, como lo es el monitoreo, control y riego del cultivo de manera automática por medio de un análisis predictivo en base a la información aportada por los sensores, en cuanto a la detección de nutrientes y el entorno del cultivo. Esto permite a los trabajadores hacer un seguimiento del terreno para así tener una respuesta en cuanto uno de los elementos básicos para crecer de la planta como el agua y los nutrientes esté cerca de llegar a insuficiente.

Este proyecto está diseñado para ofrecer a los agrónomos una solución integral, modulable y escalable que responde a las necesidades en el campo, permitiendo mejorar la productividad de sus tierras gracias a las decisiones empleadas de los agricultores con base a la predicción del sistema, logrando mejorar un equilibrio de los recursos, reducir los costos operativos y mitigar el impacto ambiental asociado a químicos o abonos con concentración alta, que fuerza la productividad de la cosecha dejando cicatrices que afecta la fertilidad a futuro. Se busca establecer un equilibrio de los nutrientes, eliminar la incertidumbre y orientar a los nuevos agricultores sobre los requerimientos que necesita cada cultivo, para así puedan ejercer buenas prácticas agrícolas sin dañar o desperdiciar el entorno.

Palabras clave

Monitoreo agrícola, riego automatizado, análisis predictivo, microservicios, sensores, IoT Internet de las Cosas, agricultura sostenible, optimización de recursos, cambio climático, gestión de cultivos, automatización agrícola, tecnologías agrícolas, producción agrícola eficiente.

Pregunta orientadora de la búsqueda

¿Como puede un sistema de monitoreo agrícola inteligente basado en la recolección de datos en tiempo real y control automatizado en el riego, mejorar la eficiencia y productividad de los cultivos a largo plazo, en condiciones variables de clima y recursos?

Sustentación teórica de la pregunta

El monitoreo agrícola inteligente es una disciplina emergente que busca integrar tecnologías avanzadas para optimizar la gestión de los recursos agrícolas y naturales, con el fin de lograr una mayor productividad y sostenibilidad en la producción. A medida que la demanda global de alimentos aumenta, los recursos naturales se ven cada vez más comprometidos (FAO, 2018), lo que hace que el uso de herramientas tecnológicas para monitorear, analizar y predecir las condiciones de los cultivos sea una necesidad urgente. La implementación de sistemas inteligentes en la agricultura permite la recolección y el análisis de datos en tiempo real sobre factores como el clima, la humedad del suelo, la temperatura y los nutrientes, lo que facilita a los agricultores la toma de decisiones más detallada, la reducción del desperdicio de recursos, la reducción de costos y el equilibrio en el ecosistema (Wolfert, 2017) y (Zhang, 2002). Esto contribuye a mejorar la rentabilidad sin sobreexplotar los recursos, evitando el uso excesivo de productos químicos que, cuando se aplican incorrectamente puedan dañar el entorno. (Rodríguez, M., Pérez, T., & Fernández, J., 2022) A través de la automatización, la inteligencia artificial y el Internet de las Cosas (IoT), (Kamilaris, 2017)los sistemas de monitoreo agrícola permiten una gestión precisa y oportuna, lo que significa en un aumento de la productividad y en una mayor sostenibilidad ambiental, asegurando que el suelo fértil siga un ciclo de cultivo saludable a largo plazo, para así contribuir al bienestar global y a la seguridad alimentaria en el futuro. (FAO, 2018)

Agricultura de precisión: Utiliza tecnologías avanzadas para optimizar la producción y los recursos agrícolas, permite tomar decisiones basada en la información procesada en tiempo real sobre las condiciones del terreno y del entorno. Este concepto se basa en el uso de sensores para monitorear el campo, sistemas de posicionamiento global para la ubicación entre las parcelas. Además, se emplea también el uso de tecnologías de información y comunicación como las herramientas TIC para procesar y analizar los datos recolectados de los sensores (Zhang, 2002).

Automatización del riego en los cultivos: En la agricultura de precisión se basa en sensores que miden los parámetros como la humedad del suelo, la temperatura y la calidad del entorno. Estos sensores permiten ajustar el riego de manera precisa, optimizando el uso y evitar desperdicios de este valioso recurso (Pereira, 2002).

Monitoreo en tiempo real: Permite la obtención de datos precisos sobre las condiciones del cultivo y el ambiente, para ser más específicos, existen los sensores de humedad, temperatura y PH proporcionando los datos en tiempo real, lo que permite al sistema tomar decisiones sobre el riego y al usuario sobre la cantidad de fertilizante a usar (Wolfert, 2017).

Tecnologías emergentes para la agricultura: Como por ejemplo el Big Data, el internet de las cosas (IoT) y la inteligencia artificial, son las que están revolucionando la agricultura de precisión. El análisis de grandes cantidades de datos a través de estas tecnologías permite predecir los patrones de crecimiento, enfermedades de los cultivos y la gestión adecuada de los recursos agrónomos permitiendo un equilibrio entre la productividad y costos agrarios (Kamilaris, 2017).

Microservicios: Los microservicios son una arquitectura de software en la que una aplicación se descompone en una colección de servicios pequeños autónomos e independientes, cada uno con su propia funcionalidad, permitiendo que la arquitectura evolucione para cubrir las necesidades, mantenerse operativa y estable por largo periodo de tiempo. Son una aproximación al diseño de las aplicaciones en la que una sola aplicación se construye como un conjunto de servicios pequeños e independientes distribuidos. Cada microservicio está diseñado para ejecutar una función específica y se comunica con otros servicios a través de interfaces utilizando APIs RESTful o mensaje basado en eventos (Newman S. , 2015).

Flask es un micro framework para Python que se utiliza en el desarrollo de aplicaciones web, incluidas las que siguen la arquitectura de microservicios. Flask es compatible con Docker, permitiéndonos empaquetar el microservicio en un contenedor que incluye todas las dependencias necesarias para que funcione en otro computador.

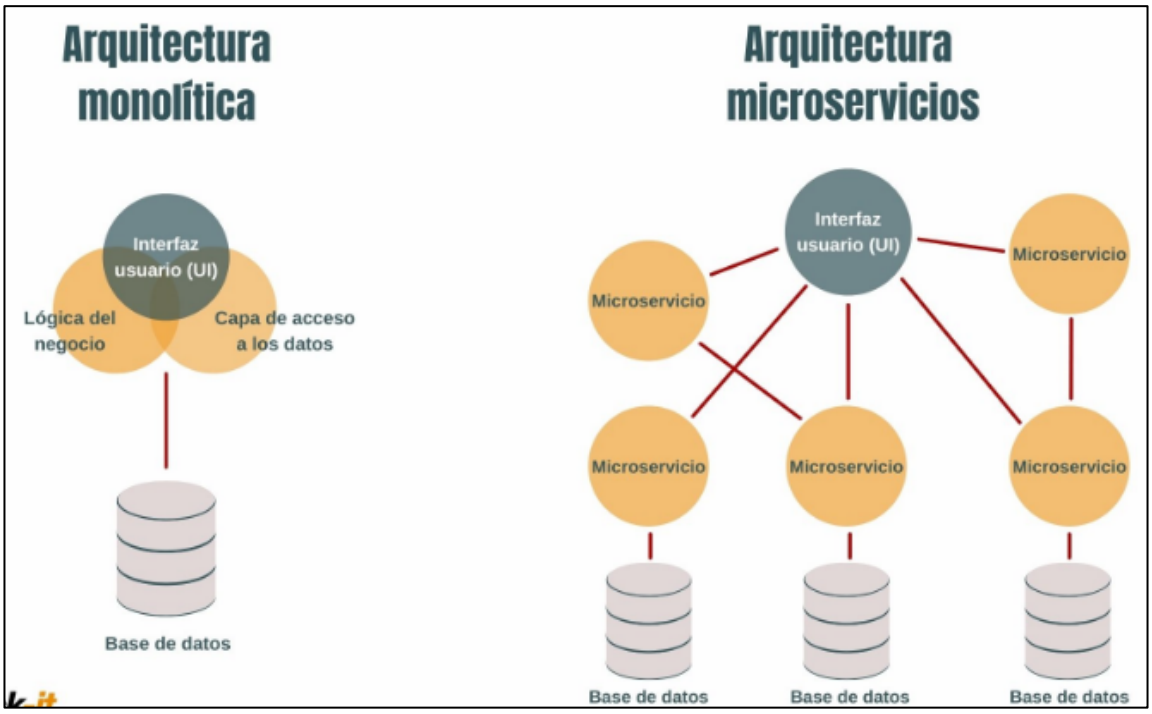
Docker es una plataforma que permite empaquetar aplicaciones y sus dependencias en contenedores, que son capaces de poder ejecutarse en cualquier entorno, siendo esto posible gracias a Docker compose.

Docker compose es una herramienta que nos permite ejecutar aplicaciones con múltiples contenedores Docker. Utilizando un archivo YAML, se especifican los servicios, redes y volúmenes necesarios, lo que facilita su configuración y despliegue de microservicios para el proyecto.

Base de datos NoSQL y relacionales: El uso de estas herramientas en sistemas distribuidos (microservicios) se ha incrementado en aplicaciones que requieren alta disponibilidad, y formas de actualización de manera escalable. PostgreSQL, MongoDB y Redis son opciones populares que nos ayuda a gestionar diferentes tipos de datos, desde información que está estructurada hasta datos no estructurados que apenas ingresa en los sensores (Stonebraker, 2007).

Microservicios vs. Arquitectura Monolítica: A diferencia de las aplicaciones monolíticas donde todo componente de software está integrado una base de código, los microservicios se distribuyen en múltiples servicios independientes. Esto significa que cada servicio tiene su propia base de datos, su lógica y procesos de despliegue, en la imagen (ver figura 1) podemos observar un ejemplo de su estructura:

Figura 1 Arquitectura monolítica vs microservicios



A continuación, vamos a realizar una comparación de las dos arquitecturas (ver figura 2)

Figura 2 diagrama comparativo de arquitecturas

Problema

En la agricultura, con el pasar del tiempo ha habido innovaciones en la tecnología que permite facilitar la actividad agropecuaria, sin embargo, muchos productores aun dependen de métodos tradicionales para sus cultivos, como el uso de herramientas manuales, siembra a mano y la observación directa al comportamiento de las plantas, estado de las nubes y el entorno (Gómez, 2020). Estos métodos, aunque funcionen, no está absuelto en cuanto a las perdidas debido a que carece de precisión en cuanto en satisfacer la necesidad de la planta, en cuanto cantidad de agua, abono y productos anti plagas (Smith, R, & Johnson, P, 2019).

En la actualidad, los trabajadores se enfrentan a los efectos del cambios climático y la poca orientación sobre la técnica adecuada de cultivo (Martínez, 2021), esto se debe a que las nuevas generaciones, no tienen acceso a la formación adecuada para darse a conocer el equilibrio de los nutrientes que requiere la planta volviendo un problema en la práctica, llevando a errores como sembrar en terreno no tan fértil, no dar suficiente agua o demasiada, o quemar todo el cultivo por una mala distribución de más de abono. (Rodríguez, M., Pérez, T., & Fernández, J., 2022) .

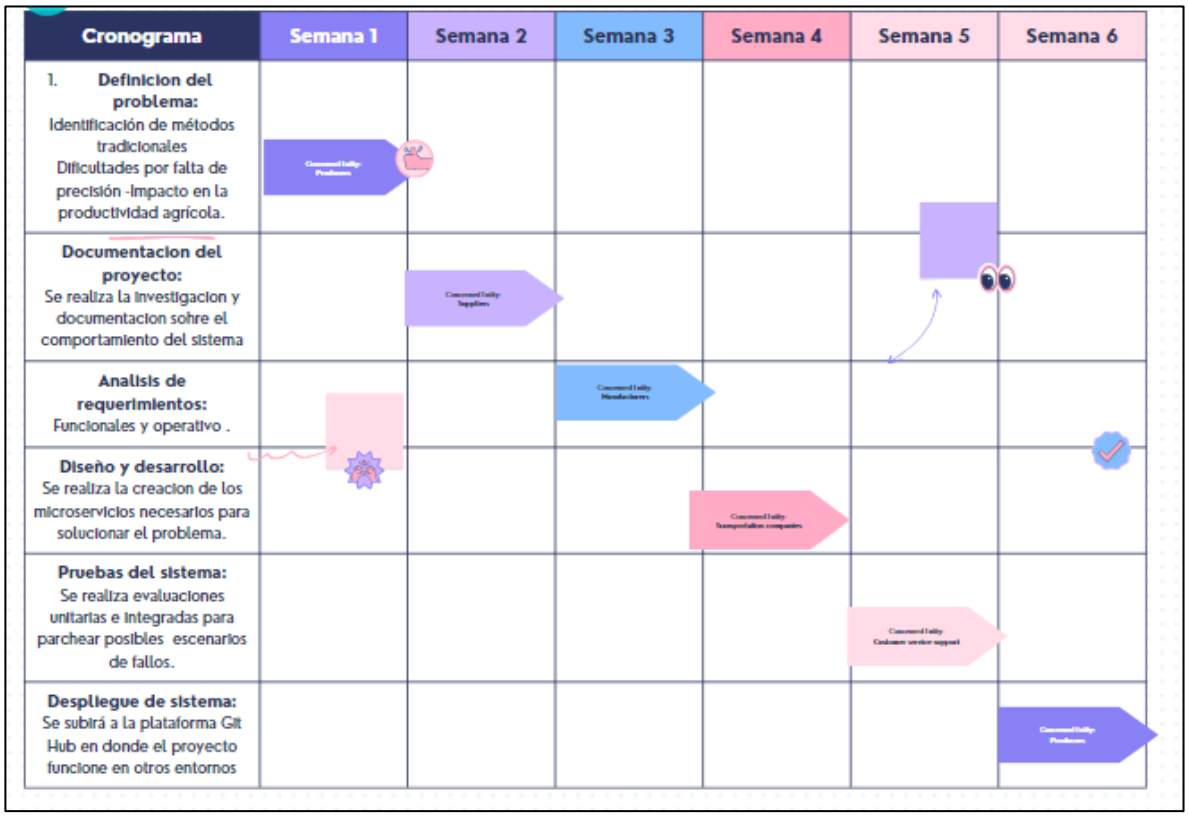
Metodología

Para el desarrollo del sistema de monitoreo agrícola inteligente se emplea las herramientas modernas compatible con Python (como Flask, venv y Docker). Pero para asegurar el proceso y seguimiento del desarrollo se empleará la metodología cascada Waterfall (ver figura 3). Esta metodología se enfoca en seguir una secuencia de desarrollo tradicional en cual cada fase debe completarse para seguir con la siguiente tarea permitiendo así que pueda desarrollar el proyecto de manera ordenada.

Figura 3 metodología cascada (Waterfall)

Para cumplir con el objetivo de desplegar la solución digital a la problemática del proyecto se prioriza las tareas que requiere mucha concentración y análisis, como vemos en el cronograma (ver figura 4) la parte investigativa va de primero, la codificación, aunque también requiere bastante lógica se estará desarrollando junto herramientas y documentaciones en la web, de esta forma en base a la retroalimentación puede ser un buen aliado para el desarrollo.

Figura 4 cronograma de actividades



Desarrollo

Gracias a la investigación realizada anteriormente, se tiene una idea de cómo debería estar estructurada la primera parte del desarrollo del proyecto, que es en la elaboración de los microservicios.

Cada microservicio tiene sus propias características, como la lógica, puerto y almacenamiento específico para que funcione de manera independiente, permitiendo su escalabilidad sin que colapse todo el sistema en caso de que falle un microservicio.

Una vez comprendida la problemática presentada, se procede al diseño de los diagramas que detallan el comportamiento y el proceso de almacenamiento del sistema, así como la recopilación de los requisitos funcionales necesarios para asegurar que el sistema SmartField cumpla con las expectativas y necesidades requeridas en el entorno agrícola, aportando herramientas novedosas como nos pueden ofrecer Python, Flask, Venv y Docker.

Diagrama de arquitectura de proyecto.

En el diagrama (ver figura 5) se puede ver que tiene diferente base de datos. Esto se debe ya que de por si los datos que recoge directamente los sensores aun no son comprensibles por lo tanto debe ser procesado para poder interpretar la medición de nutrientes en el cultivo, de lo contrario tendríamos conflictos con la integridad de la información al no ser compatible los caracteres que registra el sensor con los registros de acceso de usuarios y riego.

Figura 5 diagrama arquitectura del proyecto.

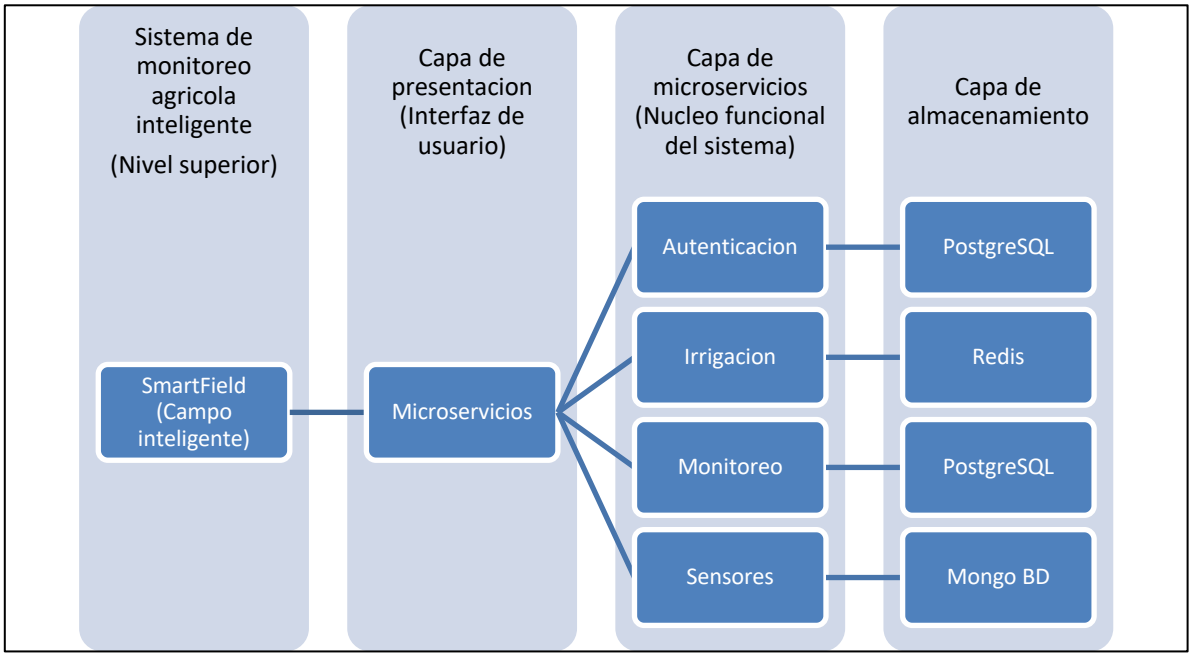


Diagrama de bases de datos.

El diagrama entidad relación representa un modelo de base de datos para un sistema de monitoreo agrícola inteligente, a continuación, se describen los elementos principales del diagrama y la relaciones entre ellos:

- Usuario: Esta entidad contiene la información sobre los usuarios del sistema
- Sensor: Representa los sensores utilizados para el monitoreo agrícola
- Riego: Esta entidad almacena los datos relacionados con los eventos de riego
- Lectura: Aquí se almacenan las lecturas realizadas por los sensores
- Parcela: Representa la tierra fértil donde se realiza las practicas agrícolas
- Cultivo: Guarda datos como el tipo de cultivo y fecha de siembra realizada en la parcela

Ahora vamos a ver qué relaciones tiene cada atributo en la base de datos:

Las relaciones entre las entidades se representan así:

(1:N) que significa de uno a muchos

(N:1) que significa de muchos a una entidad

Ahora sí, describamos las relaciones que tiene cada atributo:

- Un usuario puede estar relacionado a múltiples parcelas (1:N), pero cada parcela está vinculada a un único agricultor.
- Un sensor puede tener múltiples lecturas (1:N), pero cada lectura está asociada a un único sensor
- Las parcelas pueden tener varios eventos de riego (N:1), pero cada evento de riego corresponde a una única parcela
- cada parcela puede cultivar diferentes tipos de cultivos a lo largo del tiempo (N:1), y cada cultivo está asociado a una parcela especifica.

A continuación, se presenta el diagrama de base de datos en base a los atributos y relaciones descritas anteriormente (ver figura 6).

Figura 6 ilustración de la base de datos del proyecto.

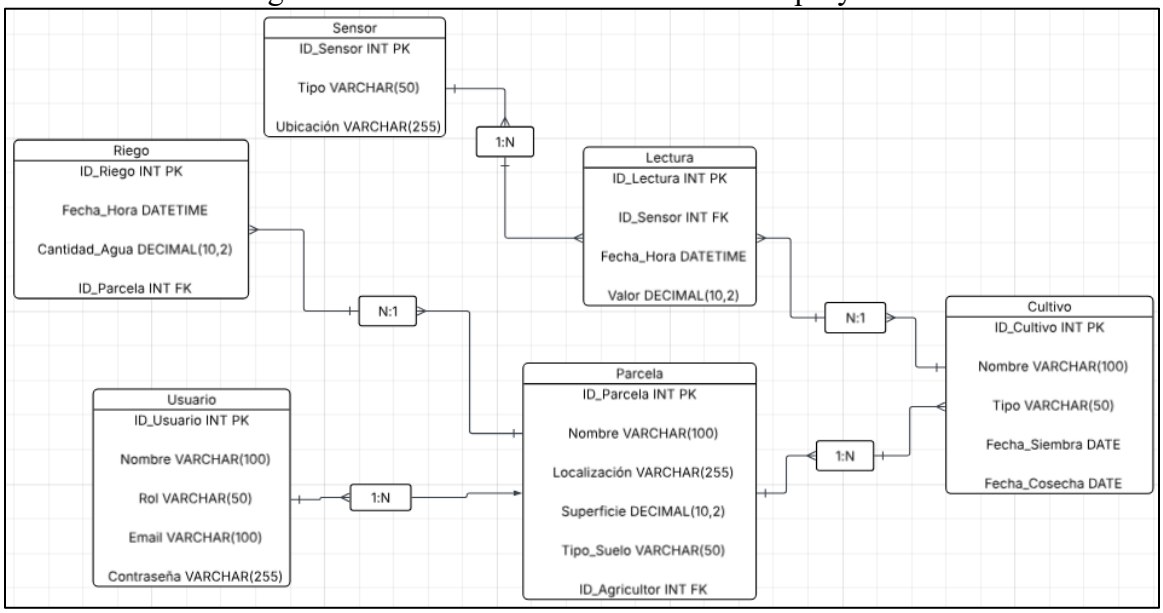
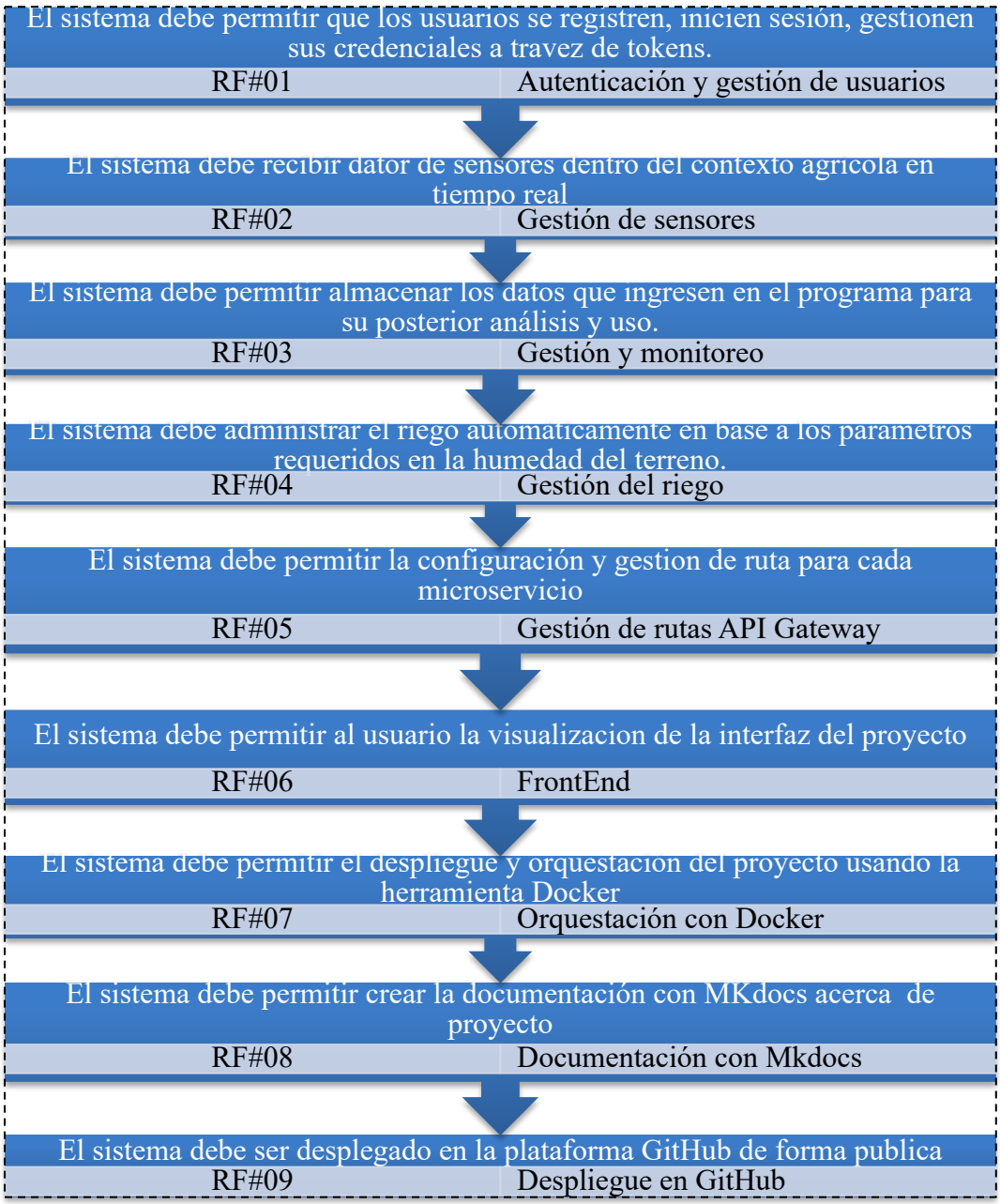


Diagrama de requerimientos funcionales

A continuación, se presenta el diagrama de los requerimientos funcionales del proyecto, se desarrolla de forma ordenada abarcando desde la creación de los microservicios (como la autenticación, sensores, monitoreo e irrigación), la comunicación entre estos microservicios gracias al enrutamiento a través de Api Gateway, la interfaz por medio del frontend que hace posible la interacción del usuario con el programa, la orquestación con Docker que permite que su código fuente sea ejecutable en cualquier computador, la creación de una guía básica del sistema para los usuarios y desarrolladores. Y por último el despliegue hacia la plataforma GitHub en donde ahí estará expuesto nuestro programa SmartField.

Esto último hará que el programa sea clonado, ejecutado y modificado sin afectar el repositorio base publicado, permitiendo un feedback que es como un buzón de sugerencias y retroalimentación de varios desarrolladores hacia nuestro proyecto. Para así darnos una orientación, que permita que la aplicación crezca gracias a su escalabilidad y se adapte en tanto a las necesidades agrícolas como a las nuevas tecnologías que va emergiendo. (ver figura 7)

Figura 7 diagrama de requerimientos funcionales



Requerimiento 1: Autenticación.

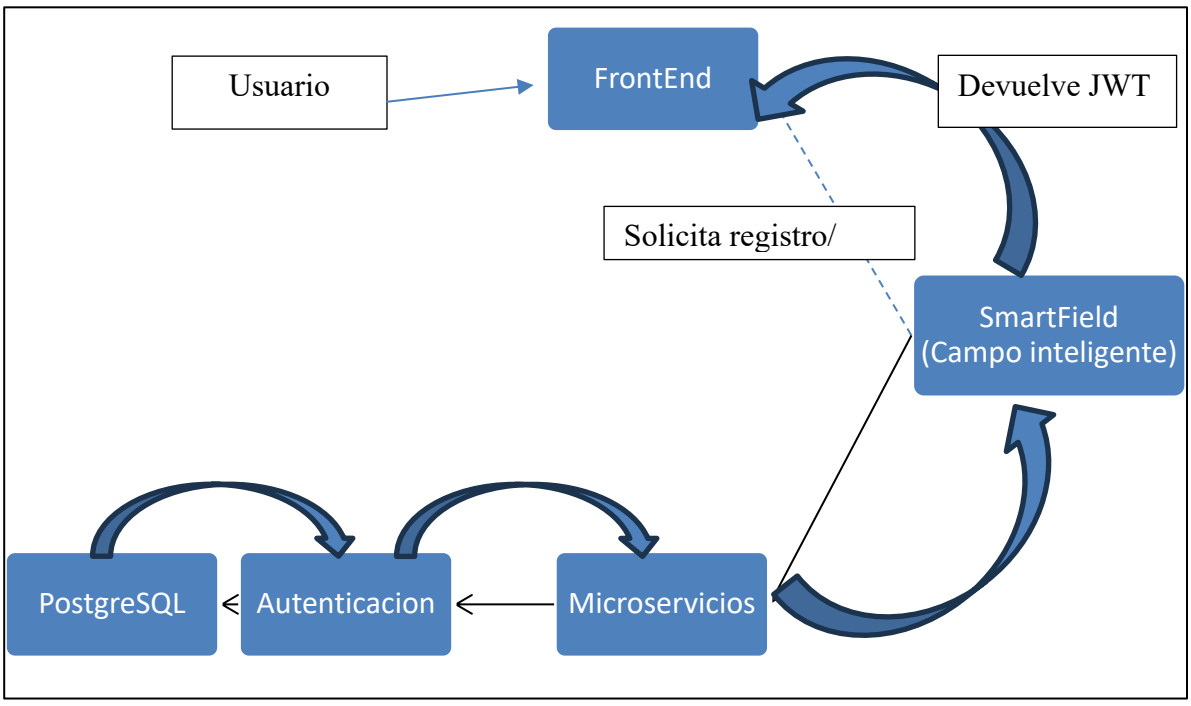
Para el desarrollo de autenticación, se usa las tecnologías y frameworks como Fast API, PostgreSQL y JWT lo que permite que solo los usuarios registrados recibirán un token (llave virtual) para que puedan ingresar al sistema SmartField.

El microservicio autenticación necesita:
Registro de usuarios, inicio de sesión, generador de jwt y validación de las credenciales.

Diagrama de autenticación

A continuación, se presenta el diagrama de cómo se comporta el sistema cuando un usuario hace una petición de autenticación a SmartField. Inicia su recorrido a través del frontend en donde se realiza una consulta hacia el microservicio autenticación, si el usuario que intenta ingresar está registrado, el microservicio autenticación le otorga una llave (Token) que le permita el ingreso e interacción con otros microservicios. (Ver figura 8)

Figura 8 Diagrama de autenticación.



Código fuente que muestra el registro de las credenciales del usuario, se crea y almacena de manera segura gracias a una encriptación como lo es el hashed_password que funciona por ejemplo como si para acceder a tu información requiriese tu huella digital, cada huella es única y solo tu huella podrá tener acceso, lo cual la integridad de la información está bastante segura y a salvo de amenazas externas (ver figura 9)

Figura 9 código fuente registro usuario en microservicio autenticación.

```python
@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    if not username or not password:
        return jsonify({"message": "Username and password are required"}), 400

    hashed_password = generate_password_hash(password, method='sha256')
    user = User(username=username, password=hashed_password)

    try:
        AuthService.create_user(user)
        return jsonify({"message": "User created successfully!"}), 201
    except Exception as e:
        return jsonify({"message": str(e)}), 500
```

Código fuente que muestra el inicio de sección del usuario, aquí se comprueba si las credenciales corresponden con las que están almacenadas, si algo (como un bot o virus programado para robar información) o alguien intenta entrar en el código fuente no podrá interactuar con la información almacenada ya que está encriptada a travez del hash_password, tampoco acceso a los microservicios al requerir la llave (Token) que permite el acceso a las funcionalidades de SmartField, generando un mayor obstáculo al atacante haciendo que el sistema funcione de manera segura y confiable sin mostrar vulnerabilidades (ver figura 10).

Figura 10 código fuente de iniciar sesión y generador de tokens

```python
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    if not username or not password:
        return jsonify({"message": "Username and password are required"}), 400

    user = AuthService.get_user_by_username(username)
    if not user or not check_password_hash(user.password, password):
        return jsonify({"message": "Invalid credentials"}), 401

    token = AuthService.generate_token(user)
    return jsonify({"token": token}), 200

if __name__ == "__main__":
    app.run(debug=True)
```

Requerimiento 2: Gestión de Sensores

Se desarrolla las variables como los nutrientes básicos de la planta (humedad, temperatura y fertilidad del terreno) usando Flask y MongoBD. Primero que todo para acceder a esta funcionalidad es requerida la llave (Token) que es asignada siempre al usuario que logre autenticarse de manera exitosa, este microservicio obtiene datos por así decirlo "crudos" directamente de los sensores por eso tenemos como ejemplo el cómo deberían ser procesados para que el sistema pueda interpretar su lectura, aunque esa tarea la realiza el microservicio monitoreo, aquí se muestra un ejemplo de cómo se captura la información y como es procesada para que sea legible con "random.uniform" en cada variable capturada (ver figura 11)

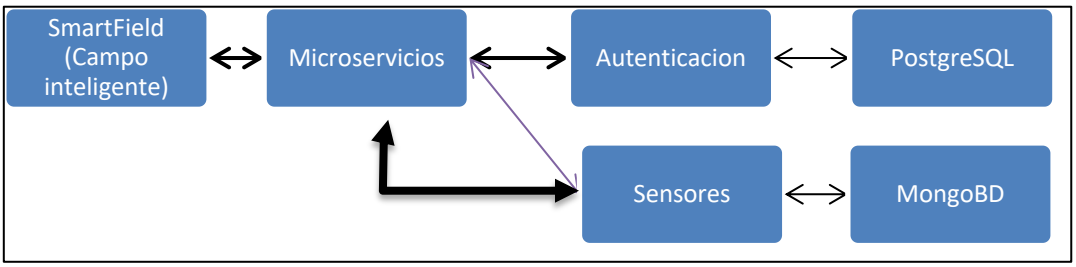
Figura 11 código fuente de gestión de sensores

```python
@app.route('/sensor-data', methods=['GET'])
@jwt_required()
def get_sensor_data():
    # Simulando datos del sensor
    temperature = random.uniform(20, 30)
    humidity = random.uniform(40, 80)
    soil_moisture = random.uniform(10, 100)

    return jsonify({
        "temperature": temperature,
        "humidity": humidity,
        "soil_moisture": soil_moisture
    })
```

Diagrama gestión de sensores

A continuación, se muestra el diagrama del microservicio sensor con su base de datos (ver figura 12)

Figura 12 diagrama de microservicio sensor con su base de datos



Requerimiento 3: Creación microservicio monitoreo.

Este microservicio se encargará de procesar y almacenar la información recibida de la base de datos de los sensores, para que así el usuario pueda comprender las métricas del cultivo. En base al ejemplo del código fuente anterior (ver figura 12) aquí se realiza la transformación de datos ilegibles que llegan del sensor en información que se puede interpretar no solo por el usuario sino también por el sistema, permitiendo así una lectura y a través de ello una predicción de las necesidades que requiere el cultivo para así reaccionar a tiempo en cuanto una de esas lecturas se encuentre deficiente (ver figura 13).

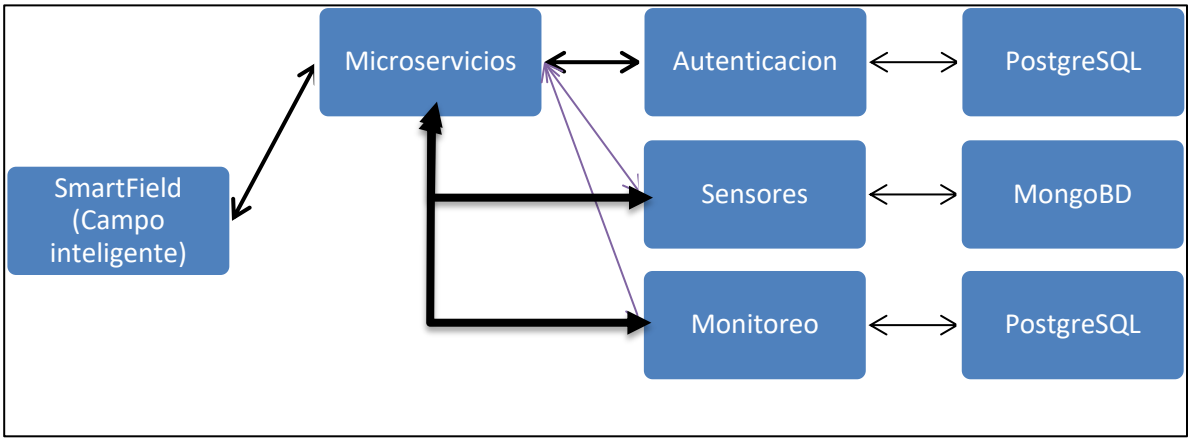
Figura 13 código fuente del microservicio monitoreo

```python
@app.route('/monitor', methods=['POST'])
@jwt_required()
def monitor_data():
    data = request.get_json()
    crop_type = data.get('crop_type')
    temperature = random.uniform(20, 30)  # Temperatura simulada
    humidity = random.uniform(40, 80)  # Humedad simulada
    nutrient_level = random.uniform(30, 100)  # Nivel de nutrientes simulado

    # Guardar los datos de monitoreo
    new_data = MonitoringData(
        crop_type=crop_type,
        temperature=temperature,
        humidity=humidity,
        nutrient_level=nutrient_level
    )
    db.session.add(new_data)
    db.session.commit()

    return jsonify({
        "crop_type": crop_type,
        "temperature": temperature,
        "humidity": humidity,
        "nutrient_level": nutrient_level
    }), 200
```

Diagrama de microservicio monitoreo

A continuación, se muestra la implementación del microservicio monitoreo con su base de datos (ver figura 14)

Figura 14 diagrama gestión y monitoreo



Requerimiento 4: Gestión del Riego.

El microservicio irrigación se encarga de administrar automáticamente el control del riego en base a los parámetros asignados para cumplir su función, se desarrolla con la base de datos Redis permitiendo realizar un reporte del riego al activarse. (ver figura 15)

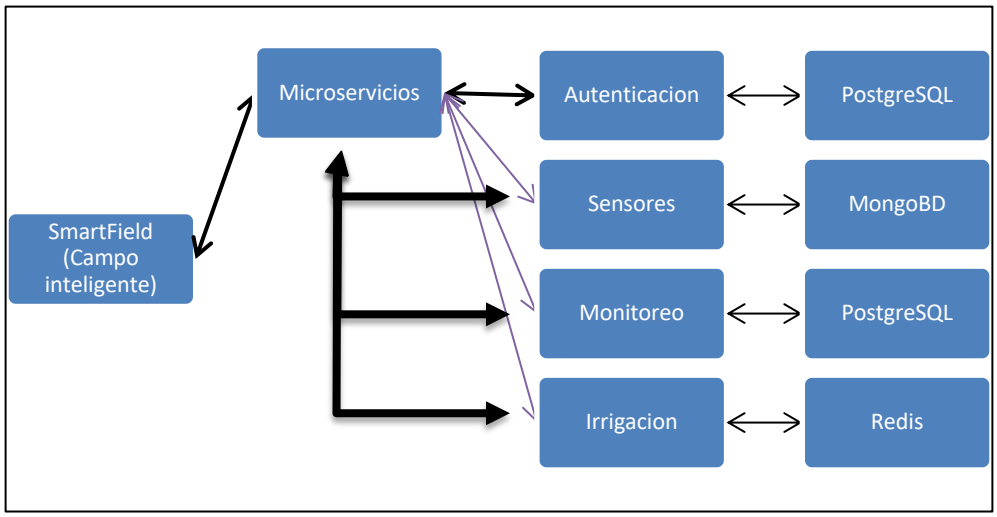
Figura 15 código fuente microservicio irrigación

```python
# Modelo de configuración de riego SmartField
class IrrigationConfig(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    crop_type = db.Column(db.String(50), nullable=False)
    threshold = db.Column(db.Float, nullable=False)  # Umbral de humedad para riego

@app.route('/activate-irrigation', methods=['POST'])
@jwt_required()
def activate_irrigation():
    data = request.get_json()
    crop_type = data.get('crop_type')
    # Simulación de lectura de humedad
    current_humidity = random.uniform(0, 100)  # Simulación de lectura en % de humedad
    config = IrrigationConfig.query.filter_by(crop_type=crop_type).first()

    if not config:
        return jsonify({"error": "Crop type not found"}), 404

    if current_humidity < config.threshold:
        # Activar riego
        # Lógica para activar sistema de riego real aquí
        return jsonify({"message": f"Riego activado para {crop_type}, humedad actual: {current_humidity}%"}), 200
    else:
        return jsonify({"message": f"Humedad suficiente, riego no necesario. Humedad actual: {current_humidity}%"}), 200
```

Diagrama de microservicio de irrigación

A continuación, se muestra el diagrama irrigación implementado (ver figura 16)

Figura 16 diagrama gestión de riego



Requerimiento 5: Gestión Api Gateway

El API Gateway facilita que todos los servicios sean accesibles a través de un único punto de entrada (localhost5000) lo que significa que el usuario no necesita saber a qué microservicio especifico está haciendo la solicitud. El API Gateway se encarga de decidir a que servicio enviar la solicitud simplificando el proceso tanto para el usuario como para los desarrolladores. Además de centralizar este enrutamiento, el API Gateway mejora la seguridad y el control permitiendo gestionar aspectos como la autenticación, la autorización y el balanceo de carga de forma controlada.

A continuación, se muestra una imagen que muestra cómo el API Gateway redirige las solicitudes hacia los diferentes microservicios. En este diagrama, se pueden ver los distintos microservicios que interactúan a través del punto de entrada único proporcionado por el API Gateway (ver figura 17).

Figura 17 código fuente de Api.

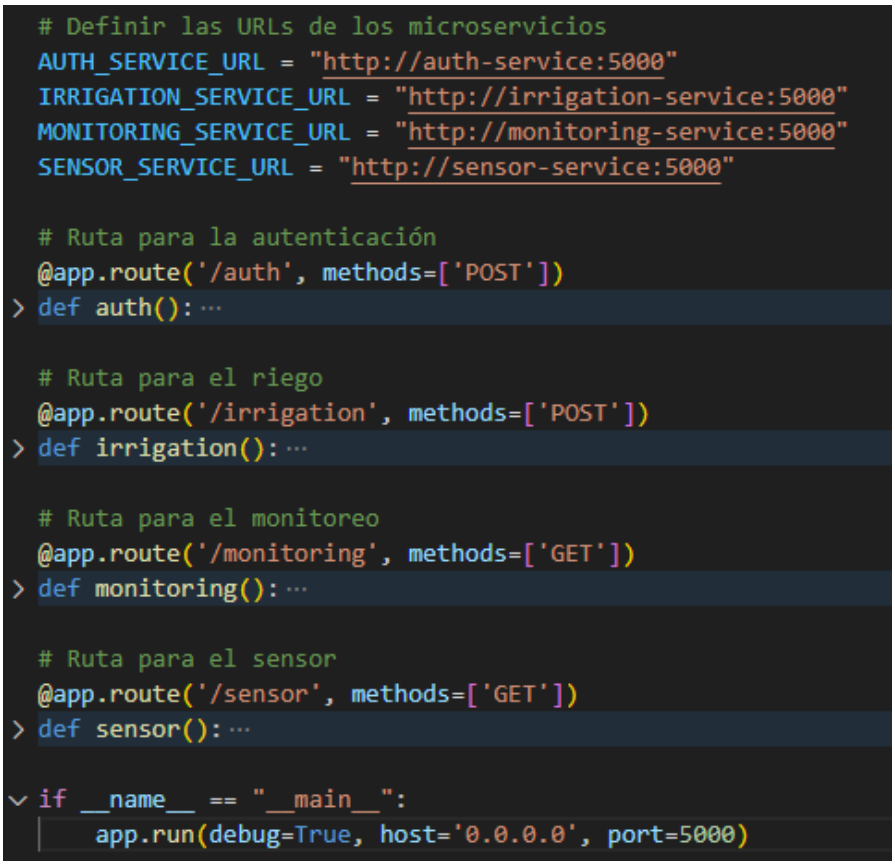
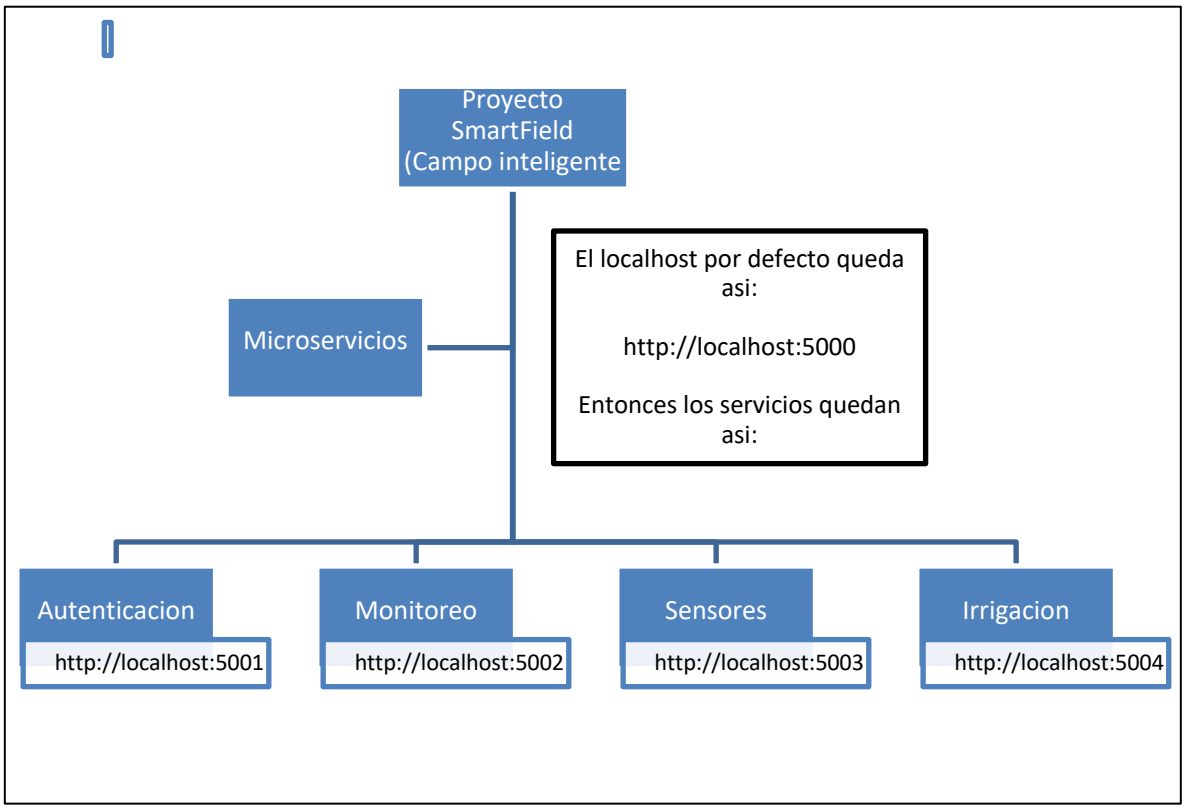


Diagrama de microservicios enrutados

A continuación, se muestra como quedaría enrutado los microservicios, una vez implementado la configuración del Api Gateway (ver figura 18)

Figura 18 diagrama de microservicios enrutados con Api Gateway



Requerimiento 6: Frontend

Se utiliza el framework React con javascript para desarrollar la interfaz de usuario para hacer posible la navegación del sistema interactuando con los microservicios como la autenticación monitoreo y reporte de riego.

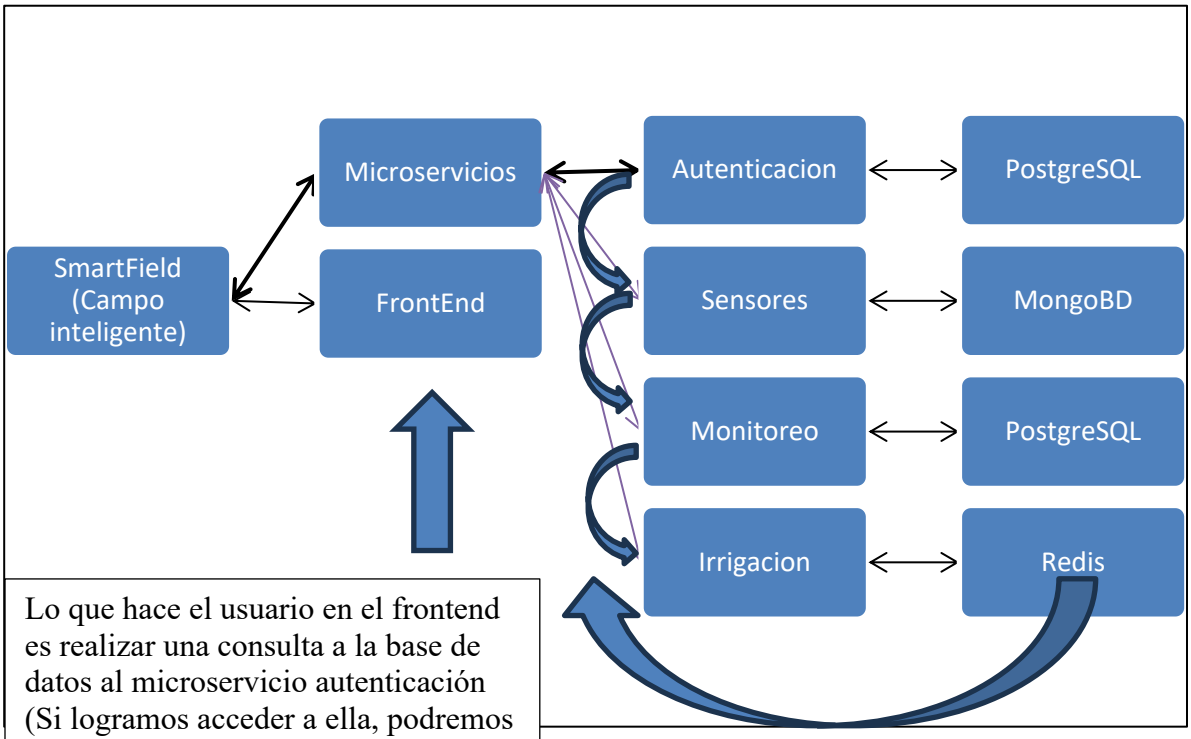
El Frontend actúa como interfaz entre el usuario y los microservicios. Cuando el usuario ingresa con sus credenciales, se consulta el microservicio de autenticación. Si la autenticación es exitosa, el usuario puede interactuar con otros microservicios.

Por ejemplo, el microservicio de Sensor recoge datos como humedad, temperatura y nutrientes, estos datos son procesados por el microservicio de monitoreo. Con base en los resultados obtenidos (como la humedad), el microservicio de irrigación decide si activar o desactivar el riego. Finalmente, la información procesada se devuelve al Frontend para que el usuario vea las condiciones del cultivo en tiempo real.

Diagrama de Frontend

A continuación, se muestra el diagrama de como interactúa el usuario con el sistema (ver figura 19)

Figura 19 FrontEnd interactuando con el usuario



BackEnd Login:

Así quedaría el codigo fuente del BackEnd de la interfaz de usuario (ver figura 20)

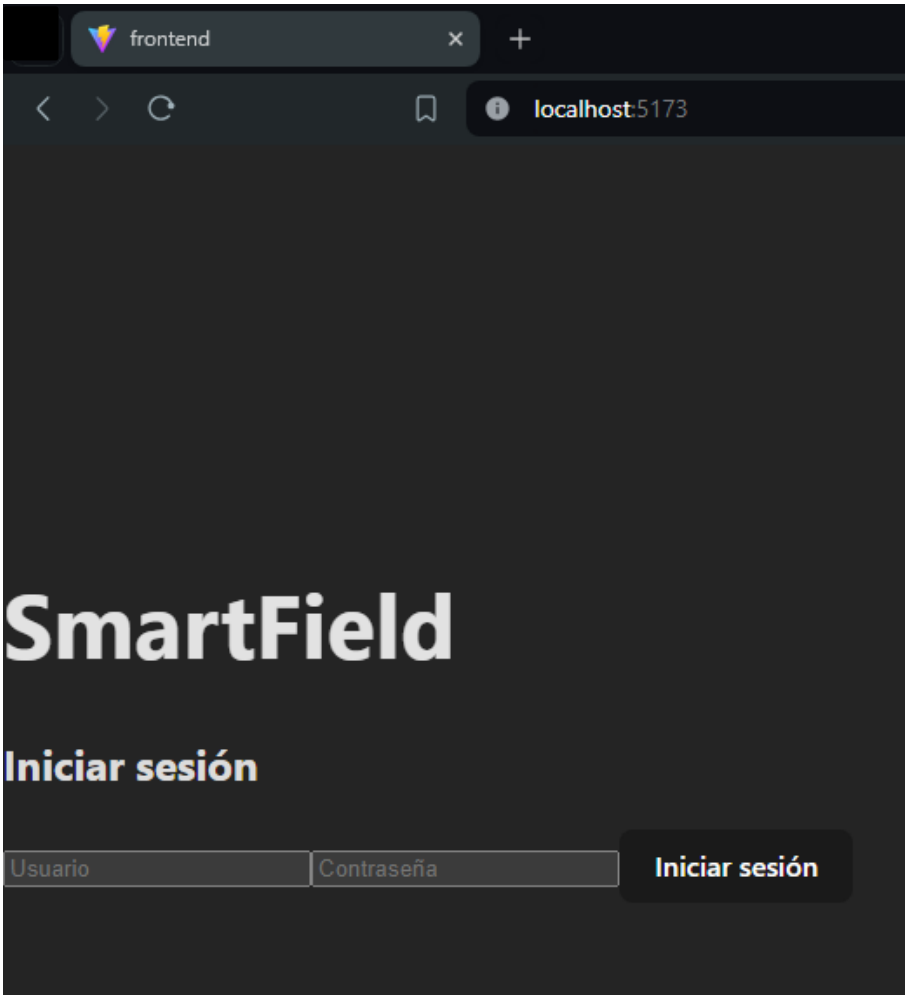
Figura 20 BackEnd Login

```
    <div>
      <h2>Iniciar sesión</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Usuario"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
        <input
          type="password"
          placeholder="Contraseña"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <button type="submit">Iniciar sesión</button>
      </form>
      {message && <p>{message}</p>}
      {error && <p style={{ color: 'red' }}>{error}</p>}
    </div>
  )
}
```

Frontend Login

A continuación, se muestra en (ver figura 21) como quedaría el FrontEnd de la autenticación de usuario:

Figura 21 FrontEnd Login Authentication SmartField.



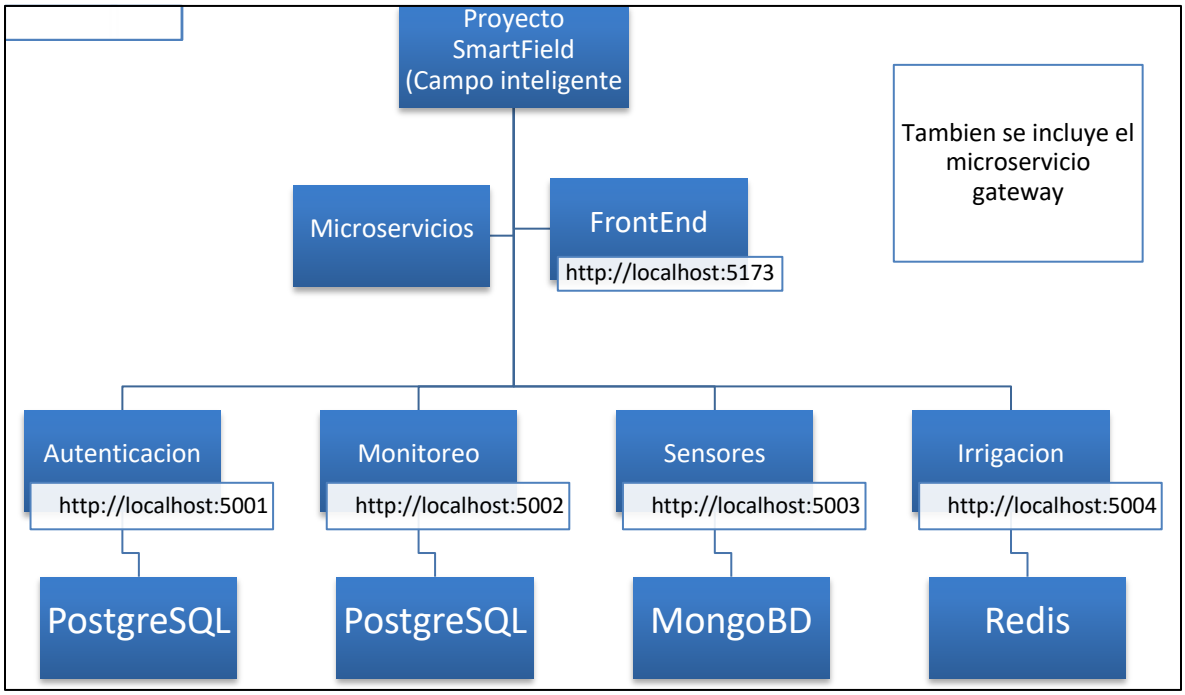
Requerimiento 7: Orquestación con Docker

La contenedorización de la aplicación se realiza mediante Docker, que nos permite empaquetar cada microservicio relacionado al proyecto junto con todas sus dependencias, configuraciones y herramientas necesarias para su ejecución, cada microservicio está contenido en una imagen Docker especifica lo que facilita la instalación, distribución y ejecución de los servicios en diferentes entornos de desarrollo.

Cada microservicio tiene su propio archivo Dockerfile, que define como se construye la imagen del contenedor, un Docker file incluye instrucciones sobre que imagen base usar, que dependencias instalar, como configurar el contenedor y que comandos ejecutar cuando el contenedor se inicie a través del archivo Docker-compose. (ver figura 22).

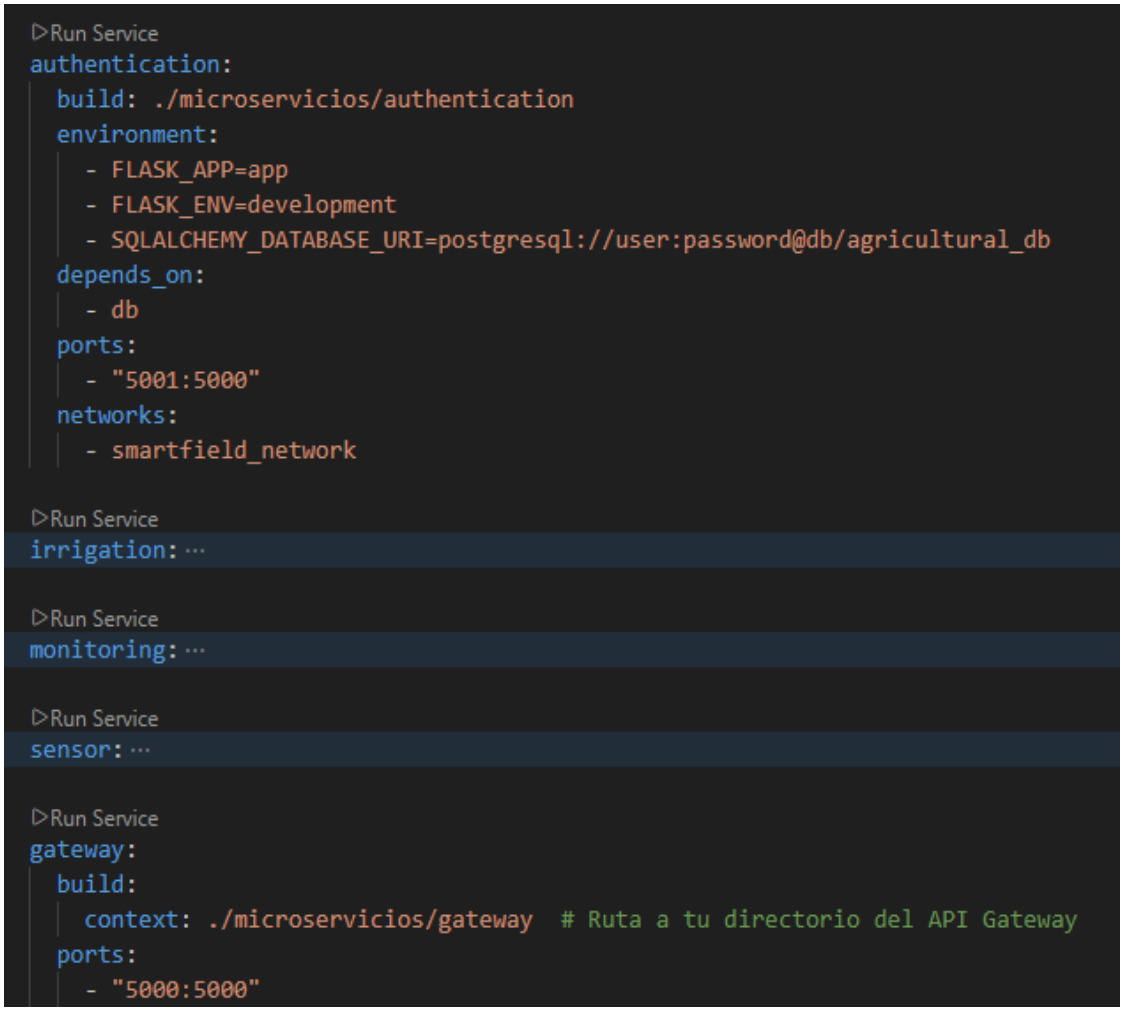
Diagrama microservicios orquestados

Figura 22 Diagrama orquestación del proyecto con Docker



A continuación, vamos a ver el código fuente del Docker-compose (ver figura 23)

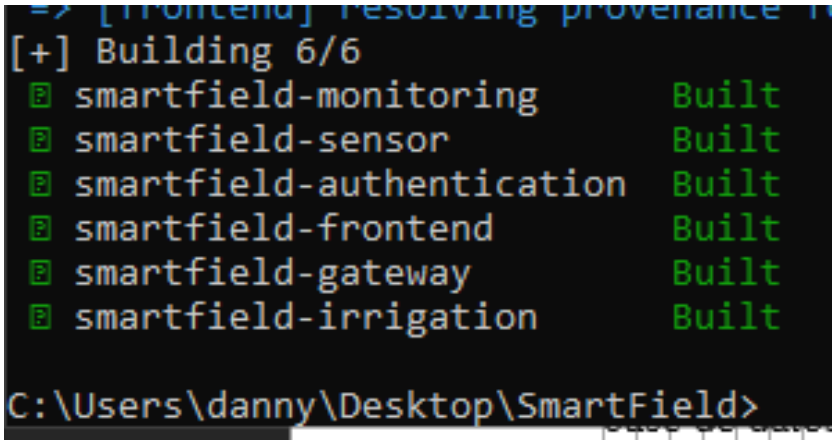
Figura 23 Código fuente para la orquestación de microservicios.



```
Run Service
authentication:
  build: ./microservicios/authentication
  environment:
    - FLASK_APP=app
    - FLASK_ENV=development
    - SQLALCHEMY_DATABASE_URI=postgresql://user:password@db/agricultural_db
  depends_on:
    - db
  ports:
    - "5001:5000"
  networks:
    - smartfield_network

Run Service
irrigation: ···

Run Service
monitoring: ···

Run Service
sensor: ···

Run Service
gateway:
  build:
    context: ./microservicios/gateway  # Ruta a tu directorio del API Gateway
  ports:
    - "5000:5000"
```

En esta figura (ver figura 24) están construidas los contenedores del proyecto una vez ejecutado el comando "docker-compose build --no-cache --parallel=false". Este comando ayuda al proyecto construirse sin tener conflictos con el cache que puede resultar un obstáculo para la creación de imágenes.

Figura 24 imágenes del proyecto orquestada



Despliegue con Docker

Una vez tenido las imágenes creadas como lo podemos ver en la (figura 25) realizamos el despliegue con "Docker-compose up -d" que nos permite levantar los contenedores en segundo plano, ya que si lo hacemos de forma directa la consola se puede atascar y no completar el despliegue, lo cual es recomendable usar el -d para que así se pueda implementar en GitHub de forma completa.

Figura 25 contenedores creados en base al proyecto SmartField

Requerimiento 8: Documentación con MKdocs

Se realiza la documentación usando el MKdocs, para asegurar que funcione, cada microservicio debe tener un api_documentation.md para ser enrutado como vemos en la imagen (ver figura 26)

Figura 26 código fuente del mkdocs.yml

C:\Users\danny\Desktop\SmartField\mkdocs.yml • Untracked

```yaml
site_name: SmartField Documentation
site_url: https://smartfield.com
site_description: Documentación para el proyecto de microservicios SmartField

# Tema de la documentación (Material, en este caso)
theme:
  name: material

# Estructura de navegación
nav:
  - Home: index.md
  - Autenticación:
      - API Documentation: microservicios/authentication/api_documentation.md
  - Riego:
      - API Documentation: microservicios/irrigation/api_documentation.md
  - Monitoreo:
      - API Documentation: microservicios/monitoring/api_documentation.md
  - Sensores:
      - API Documentation: microservicios/sensor/api_documentation.md
  - Guías:
      - Guía de Instalación: docs/installation_guide.md
```

Es posible toparse con incompatibilidades de caracteres en los archivos .md así que para asegurarse de que funcione, el proyecto tendrá un archivo.py que convierte los caracteres en UTF-8 como vemos en la imagen (ver figura 27).

Figura 27 código fuente que convierte los caracteres incompatibles en UTF-8

convertir_a_utf8.py U

convertir_a_utf8.py > ...

```python
import os

def convertir_a_utf8(filepath):
    with open(filepath, 'r', encoding='latin-1') as file:
        contenido = file.read()
    with open(filepath, 'w', encoding='utf-8') as file:
        file.write(contenido)

for root, dirs, files in os.walk('docs'):
    for file in files:
        if file.endswith('.md'):
            filepath = os.path.join(root, file)
            convertir_a_utf8(filepath)
            print(f"Archivo convertido a UTF-8: {filepath}")

```

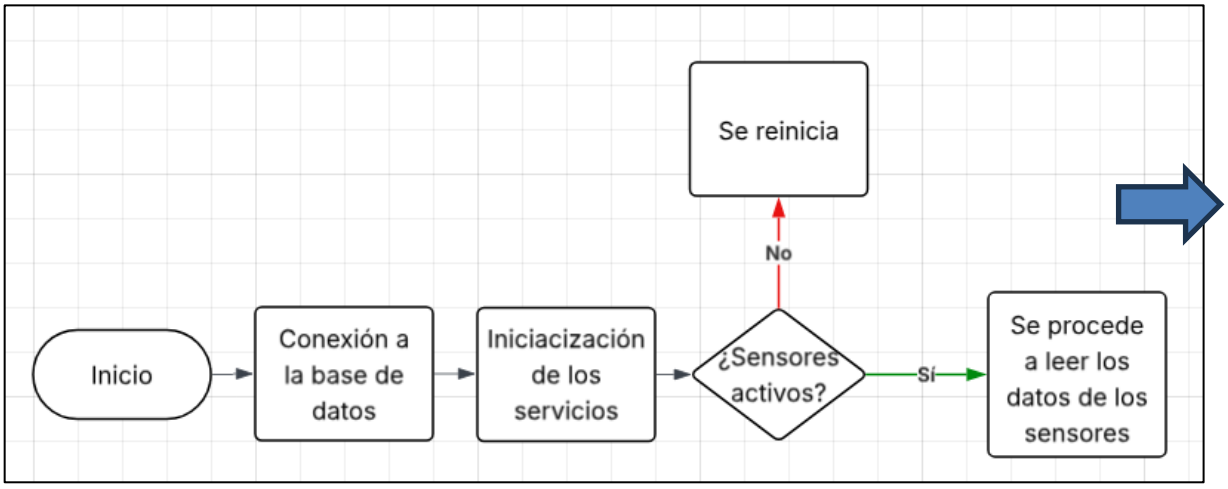
Al ver la imagen (ver figura 28) podemos ver la página de documentación del proyecto SmartField

Figura 28 Visualización de la documentación del proyecto.

Diagrama de flujo del proyecto

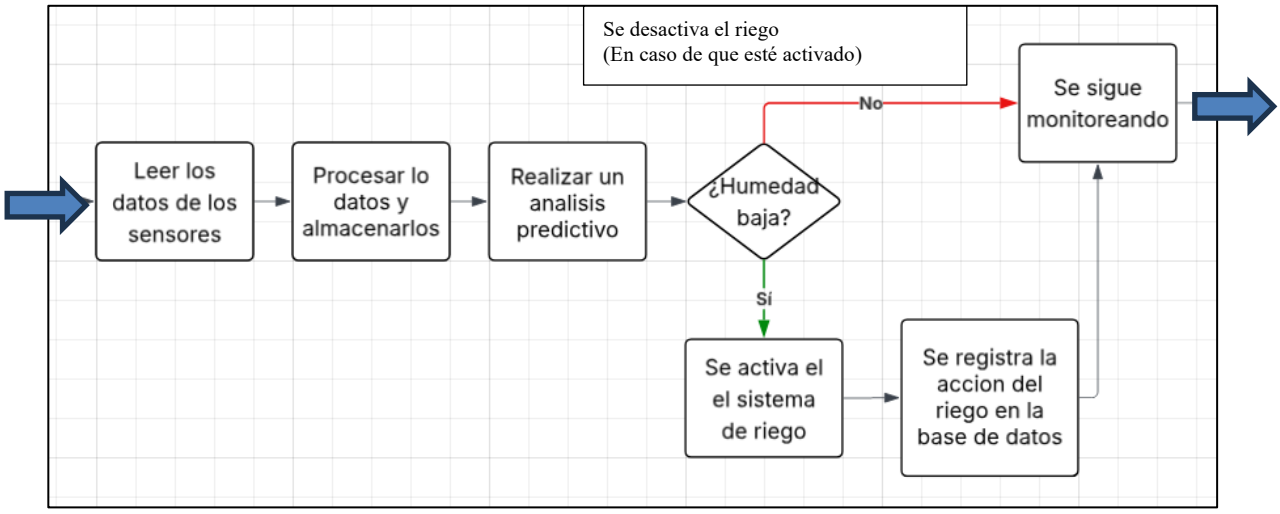
Podemos observar (ver figura 31) el recorrido del sistema, empieza iniciando la conexión e inicialización de los componentes del necesarios del proyecto, como la disponibilidad de la base de datos y operatividad de los microservicios para luego verificar el estado del sensor, una vez activado, se procese a recaudar datos del sensor hacia la planta que lo monitorea.

Figura 31 Diagrama de flujo: Inicio de componentes SmartField



La flecha azul indica que el recorrido del sistema continua. En la siguiente figura (ver figura 32) aquí capturamos, procesamos y almacenamos la información que nos sirve para realizar un cálculo predictivo sobre los niveles hídricos. En base al resultado se activa una condicional que nos permite ejecutar el riego, registrarlo y continuar con el seguimiento en tiempo real. Ya cuando los niveles hídricos estén dentro del rango requerido, el dispositivo de riego automáticamente se apaga.

Figura 32 Diagrama de flujo: recolección, procesamiento y ejecución de instrucciones sobre el riego.



Después avanzamos hacia el siguiente flujo (ver figura 33), en donde se continua el monitoreo y para asegurar el seguimiento en tiempo real se envía los valores como la temperatura, nivel hídrico y nivel de nutrientes como el potasio, nitrógeno, calcio entre otros.

Luego de la condicional del sensor, si está activo permite al programa ejecutar un ciclo (Se devuelve hasta la acción de leer los sensores que está en la figura 31) que continúa monitoreando, hasta que se desactive el sensor específico del cultivo. Y aquí concluye el sistema SmartField, permitiendo actualizar, registrar y enviar el reporte al usuario siempre y cuando el sensor esté habilitado.

Figura 33 Diagrama de flujo: envió de la información al usuario.

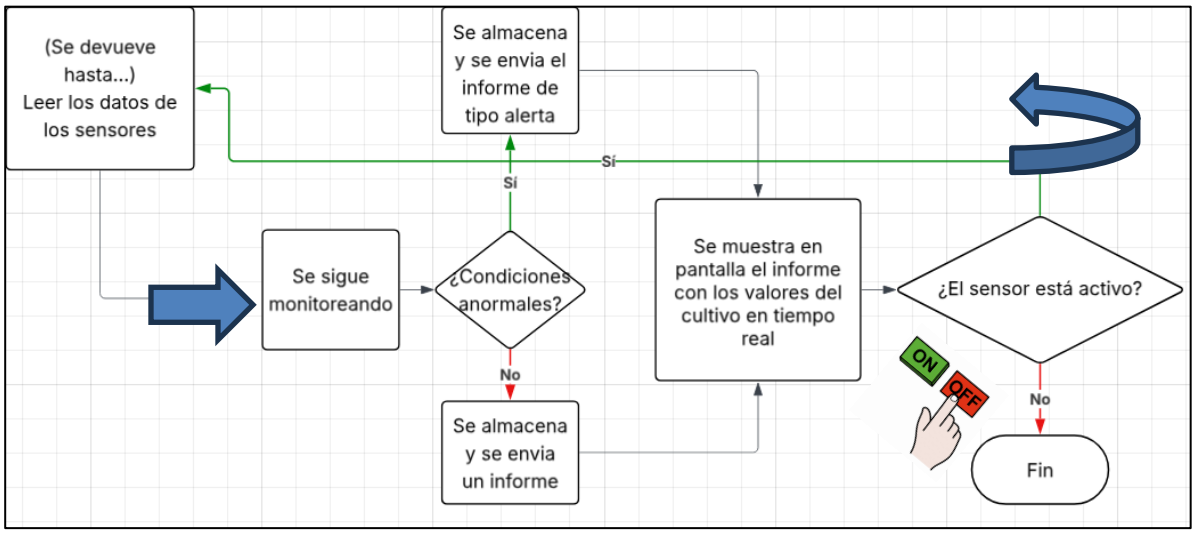


Figura 34 empleabilidad manual y/o aplicativa del sensor

Nota: El sistema se detiene si el sensor que monitorea el cultivo o la planta se deshabilita ya sea de forma manual o aplicativo como lo podemos observar en la (Figura 34).

Conclusiones

El sistema de monitoreo agrícola inteligente representa una solución bastante eficiente para los desafíos que enfrenta la agricultura moderna, a través de la integración de tecnologías avanzadas como Python, Flask, venv y Docker podemos realizar la lógica del sistema permitiendo el monitoreo en tiempo real, el análisis predictivo y la automatización del riego en los cultivos. Este sistema ofrece a los agricultores una herramienta poderosa para la optimización del uso de los recursos agrónomos e hídricos, mejorar la productividad del terreno y reducir el impacto ambiental con el uso equilibrado de productos que se emplea en los cultivos, ya sea para acelerar la producción o eliminar algún parasito en la planta.

El sistema no solo contribuye a mejorar el proceso agrícola, sino que también maneja la sostenibilidad al permitir una gestión más precisa de los recursos empleados en la agricultura, como el agua, fertilizantes y pesticidas o matamalezas. De igual manera, al ofrecer a los agricultores información precisa y actualizada en tiempo real sobre el estado de sus cultivos en diferentes parcelas y condiciones ambientales, se facilita la respuesta del usuario en la toma de decisiones informada que pueden reducir perdidas sea del cultivo, recurso o la fertilidad del terreno para permitir, un ciclo agro-sostenible.

El desarrollo y el despliegue de un sistema de monitoreo agrícola inteligente tiene el potencial de transformar la agricultura en ser parte de la era digital, optimizando y automatizando los procesos de cultivo como en control sobre el riego. Apoyando la sostenibilidad ecológica a largo plazo sin generar daño permanente.

Referencias

Cruz, A. (S.F). *Primeros pasos con FastApi Aquí continúa tu camino en el desarrollo de* aplicaciones *web en Python con FastApi.* (A. Cruz, Ed.) Chicago.

FAO. (2018). The future of food and agriculture: Trends and challenges. *Food and Agriculture Organization of the United Nations.*

Galan, J., Martin, A., Bernal, C., & Lopez, E. (2017). *Los MOOC y la Educación Superior.* (E. Octaedro, Ed.) Retrieved from https://www.google.com.co/books/edition/Los_MOOC_y_la_Educaci%C3%B3n_Superior/tgiIDwAAQBAJ?hl=es&gbpv=0

Gómez, A. (2020). Innovaciones tecnológicas en la agricultura moderna: Un análisis crítico de los métodos tradicionales. *Revista de Innovación Agricola*, 35(2), 112-118.

Kamilaris, A. F. (2017). The rise of smart farming: A review of Internet of Things, Big Data, and Machine Learning technologies in agriculture. *Journal of Agricultural Engineering*, 48(2), 31-46.

Leon, A. R., Acosta, J. L., & Diaz, R. A. (2021). Apliacacion de la metodologia incremental en el desarrollo de sistema de informacion. Retrieved from http://scielo.sld.cu/scielo.php?pid=S2218-36202021000500175&script=sci_arttext

Martínez, L. (2021). Capacitación agrícola en tiempos de cambio climático. *International Journal of Agricultural Education*, 29(1), 77-84.

Muñoz, P. C. (2009). *Plataformas de teleformación y herramientas telemáticas.* España.

Navarro, M., Berbek, P., & Sanchez, J. (2024). *Investigacion y conocimientos en la eduacacion actual.* (S. Dykinson, Ed.) Chicago.

Newman, S. (2015). Building Microservices: Designing Fine Grained Systemns. *O' Reilly Media*.

Newman, S. (2019). *Monolith to Microservices:Evolutionary Patternsto Transform Your Monolith.* Chicago: O'Reilly Media, Incorporated.

Ortega, J. M. (2020). *Tecnologías para arquitecturas basadas en microservicios.* (J. M. Ortega, Ed.)

Pereira, L. S. (2002). Improved management of irrigation in arid and semi-arid regions. . *Agricultural Water Management*, 56(3), 151-164.

Rocio , E. (2005). *Trabajo colaborativo en educación universitaria:.* (N. E. Educativa, Ed.) Mexico.

Rodríguez, M., Pérez, T., & Fernández, J. (2022). Desafíos en la agricultura en el siglo XXI: Nutrientes y fertilización en tiempos de crisis climatica. *Agroecología*, 12(3), 52-66.

Sanchez, J. J. (2022). *Aprender Docker, un enfoque práctico.* España : Marcombo.

Smith, R, & Johnson, P. (2019). Eficiencia y sostenibilidad en la agricultura. Tecnología vs métodos tradicionales. *Agricultural Engineering*, 48(4), 205-217.

Stonebraker, M. &. (2007). One size fits all: An idea whose time has come and gone. *ACM SIGMOD Record*, 35(3), 39-45.

UNESCO. (2024). (UNESCO, Ed.) Retrieved from https://www.google.com.co/books/edition/Informe_de_seguimiento_de_la_educaci%C3%B3n/gNINEQAAQBAJ?hl=es&gbpv=0

Usaola, M. (2015). (M. P. Usaola, Ed.) España. Retrieved from https://www.google.com.co/books/edition/_/UiAvCwAAQBAJ?hl=es-419&gbpv=0

Wolfert, S. G.-J. (2017). Big Data in Smart Farming a review. *Agricultural Systems*, 153, 69-80.

Zhang, Q. W. (2002). Precision agriculture: A worldwide overview. *Precision Agriculture*, 3(4), 379-389.