
Resum

El projecte **RiegAI** és un sistema de reg automàtic que permet regar grans quantitats de plantes emmagatzemades en taules de cultiu, optimitzant el temps de l'operari i evitant la necessitat de fer-ho presencialment. Utilitza sensors per a mesurar la humitat de la terra de les plantes, la temperatura i el nivell d'aigua. A més, integra una API de clima per a ajustar el reg en funció de les condicions meteorològiques externes, garantint un ús eficient i sostenible de l'aigua.

El sistema automatitza l'ompliment i buidatge de les taules mitjançant la programació de dies i horaris específics, i ofereix una interície gràfica intuitiva on els usuaris poden monitoritzar i controlar el reg des de qualsevol dispositiu. També incorpora un assistent d'intel·ligència artificial que respon les preguntes que tinga el propi operari del que siga.

Paraules clau: Reg automàtic, IoT, Arduino, intel·ligència artificial, sensors, Node.js, React.js, MQTT, MongoDB, automatització, sostenibilitat, tecnologia agrícola.

Abstract

The **RiegAI** project is an automatic irrigation system designed to water large quantities of plants stored in cultivation tables, optimizing the operator's time and eliminating the need for physical presence. It uses sensors to measure soil moisture, water temperature, and water level. Additionally, it integrates a weather API to adjust irrigation based on external weather conditions, ensuring efficient and sustainable water usage.

The system automates the filling and emptying of the tables through specific day and time schedules, and provides an intuitive graphical interface where users can monitor and control irrigation from any device. It also incorporates an artificial intelligence assistant that answers any questions the operator may have.

Keywords: Automatic irrigation, IoT, Arduino, artificial intelligence, sensors, Node.js, React.js, MQTT, MongoDB, automation, sustainability, agricultural technology.

Resumen

El proyecto **RiegAI** es un sistema de riego automático que permite regar grandes cantidades de plantas almacenadas en mesas de cultivo, optimizando el tiempo del operario y eliminando la necesidad de hacerlo presencialmente. Utiliza sensores para medir la humedad del suelo, la temperatura y el nivel del agua. Además, integra una API climática para ajustar el riego en función de las condiciones meteorológicas externas, garantizando un uso eficiente y sostenible del agua.

El sistema automatiza el llenado y vaciado de las mesas mediante la programación de días y horarios específicos, y ofrece una interfaz gráfica intuitiva donde los usuarios pueden monitorizar y controlar el riego desde cualquier dispositivo. También incorpora un asistente de inteligencia artificial que responde a las preguntas que tenga el propio operario.

Palabras clave: Riego automático, IoT, Arduino, inteligencia artificial, sensores, Node.js, React.js, MQTT, MongoDB, automatización, sostenibilidad, tecnología agrícola.

Índice general

1	Introducció	1
1.1	Motivació	1
1.2	Estat de l'Art	2
1.3	Objectius Generals i Específics	5
2	Marc Teòric	6
2.1	Introducció al Marc Teòric	6
2.2	Hardware Utilitzat	7
2.2.1	Arduino Nano 33 IoT	7
2.2.2	Sensors i Actuadors	9
2.2.3	Fonts d'Alimentació	15
2.3	Tecnologies de Backend	17
2.3.1	Node.js i Express	17
2.3.2	API REST	19
2.3.3	MongoDB	21
2.3.4	MQTT	23
2.3.5	WebSockets	24
2.3.6	API d'OpenAI	26
2.3.7	Altres Mòduls i Llibreries	28
2.4	Tecnologies de Frontend	30
2.4.1	React.js	30
2.4.2	React Router DOM	32
2.4.3	React-Toastify	33
2.5	Infraestructura de Desenvolupament	34
2.5.1	Arduino IDE	34
2.5.2	Mosquitto Broker	35
2.5.3	Visual Studio Code	36
2.5.4	Postman	37
2.5.5	MongoDB Compass	38
3	Implementació del Sistema	39
3.1	Arquitectura Hardware	39
3.1.1	Descripció dels components	39
3.1.2	Esquemes i tables dels components	46
3.2	Implementació del Software	47
3.2.1	Backend	52
3.2.2	Frontend	75
4	Validació	80
4.1	Validació del Backend	80
4.2	Validació del Frontend	83
4.3	Demostració de Fluxos de Treball	90

4.3.1	Flux 1: Registre i Login d'un Usuari	90
4.3.2	Flux 2: Plenatge d'una Taula	91
5	Conclusions	95
5.1	Conclusions	95
5.2	Treball Futur	95
6	Bibliografía	96

Índice de tablas

1	Comparativa entre empreses i el sistema RiegAI	4
2	Assignació de pins i ús en el sistema RiegAI.	46
3	Relació del treball amb els ODS	101

1 Introducció

1.1 Motivació

Com a jove de camp que sóc, he crescut envoltat de natura i plantes, ajudant des de ben menut al meu pare en el negoci familiar, un viver anomenat **Viveros Biosca**, situat a Muro d'Alcoi. Aquest entorn m'ha permés observar de prop què implica la cura i el manteniment de les plantes, així com les oportunitats de millora en els processos agrícoles, els quals soLEN ser molt rudimentaris.

En particular, un dels processos més tediosos al viver és l'ompliment manual de les taules de reg, que estan dissenyades per a contindre grans quantitats de plantes. Actualment, al viver existeixen un total de 15 taules de cultiu, i omplir aquestes taules manualment amb una mànega pren aproximadament entre 90 i 100 minuts. Aquest procés limita la possibilitat de realitzar altres tasques importants durant aquest temps, ja que omplir les taules requereix estar presencialment subjectant la mànega.

La motivació principal d'aquest projecte naix precisament d'aquesta necessitat: **optimitzar el temps i l'esforç emprats en el reg de les taules**. Desenvolupar un sistema automatitzat que permeta omplir i buidar aquestes taules de manera autònoma no sols permet estalviar temps, sinó que també obri la possibilitat d'invertir aquest temps en la realització d'altres activitats del viver. Aquest projecte, per tant, reflecteix el meu interès per aplicar la tecnologia per a resoldre problemes quotidians en un entorn agrícola i contribuir a la millora dels processos en el nostre negoci familiar.

1.2 Estat de l'Art

En aquesta secció es presenten empreses i sistemes de reg automàtic ja desenvolupats que tenen similituds amb el projecte **RiegAI**. S'analitzen les seues característiques principals i es destaquen les diferències en aspectes com el cost, la interfície d'usuari, les funcionalitats, etc.

Regaber (Mat Holding)

Regaber, una empresa líder en sistemes de reg per degoteig a Espanya i Portugal, ofereix solucions centrades en l'agricultura sostenible. La seva proposta inclou sistemes que optimitzen l'ús de l'aigua basant-se en dades climàtiques per augmentar l'eficiència i reduir el malbaratament d'aigua (Regaber, [2024](#)).

Diferències:

- Enfocament en l'agricultura a gran escala, mentre que el nostre projecte està pensat per taules de cultiu més petites i específiques.
- No incorpora interfícies d'usuari personalitzades ni assistents d'IA.
- No disposa de programació automàtica de reg per dies i hores.

Llaboria Group

Aquesta empresa dissenya i instal·la sistemes de reg personalitzats segons les necessitats dels seus clients, incloent programació avançada basada en temps, volums i dades de sensors o meteorològiques Group, [2024](#).

Diferències:

- Tot i que ofereixen programació avançada, no s'esmenta cap integració d'IA per optimitzar el reg.
- No disposen d'una interfície desenvolupada específicament per gestionar taules de cultiu com la nostra.
- Els seus serveis estan enfocats a una àmplia varietat d'aplicacions agrícoles, mentre que el nostre projecte se centra en un ús més específic.

Parcs i Jardins Catalunya

Parcs i Jardins Catalunya es dedica a la instal·lació de sistemes de reg automàtic per a jardins i espais verds, amb l'objectiu d'optimitzar el consum d'aigua i reduir costos i Jardins Catalunya, [2024](#).

Diferències:

- El seu enfocament principal és en espais verds urbans, no en taules de cultiu.
- No ofereixen funcionalitats de programació detallada de reg o buidatge per dies i hores específiques.
- No inclouen cap sistema d'integració amb IA ni monitoratge en temps real.

CitySens

CitySens ofereix solucions de reg automàtic per a plantes d'interior, adaptant-se a diferents tipus de plantes i espais. Els seus sistemes són ideals per a usuaris domèstics que necessiten una opció senzilla per mantenir plantes en espais reduïts CitySens, [2024](#).

Diferències:

- Els sistemes estan orientats exclusivament a plantes d'interior.
- No inclouen funcionalitats de programació avançada ni una interfície personalitzada.
- No hi ha integració d'IA ni comunicació en temps real amb el hardware.

Regs Batlle

Regs Batlle proporciona instal·lacions de sistemes de reg automàtic personalitzats per a jardins, terrasses i horts. Ofereixen opcions de reg per aspersió i degoteig, amb automatització i control de l'aigua Batlle, [2024](#).

Diferències:

- El seu servei està orientat a una àmplia varietat d'espais, però no està enfocat específicament a taules de cultiu.
- No disposa d'una interfície d'usuari per controlar i programar el reg.
- Falta d'integració amb IA o sensors per optimitzar les decisions de reg.

Tabla 1: Comparativa entre empreses i el sistema RiegAI

Característica	Regaber	Llaberia Group	Parcs i Jardins	RiegAI
Interfície personalitzada.	✗	✗	✗	✓
Assistent d'intel·ligència artificial.	✗	✗	✗	✓
Programació per dies i hores.	✗	✓	✗	✓
Enfocament específic en taules de cultiu.	✗	✗	✗	✓
Monitoratge en temps real.	✗	✗	✗	✓
Integració amb sensors.	✓	✓	✓	✓
Solucions per a espais verds.	✗	✗	✓	✗

1.3 Objectius Generals i Específics

Objectius Generals

- Dissenyar i implementar un sistema de reg automàtic basat en tecnologies IoT que permeta monitoritzar i controlar de manera remota l'estat de taules de cultiu, optimitzant l'ús de recursos i reduint la necessitat d'intervenció presencial.
- Proporcionar una solució escalable que permeta l'automatització del reg en múltiples taules amb una mínima necessitat d'ajustos en el sistema existent.

Objectius Específics

- **Hardware:**

- Integrar sensors d'humitat del sòl, temperatura de l'aigua i nivell d'aigua en una taula de cultiu per a recopilar dades en temps real.
- Implementar vàlvules solenoides per a automatitzar l'ompliment i buidatge de les taules de reg, permetent un control precís del flux d'aigua.
- Garantir que el disseny del sistema siga escalable, facilitant la integració de noves taules de cultiu mitjançant l'addició de hardware addicional sense necessitat de modificar significativament el programari.

- **Backend:**

- Desenvolupar un backend en Node.js que procese les dades dels sensors, emmagatzeme els registres històrics en una base de dades MongoDB i gestione les operacions automatitzades del sistema.
- Incorporar la comunicació MQTT per a enviar i rebre comandes entre el hardware i el backend en temps real.
- Configurar un sistema de programació per a permetre que els usuaris estableixen horaris personalitzats per a l'ompliment i buidatge de les taules.

- **Frontend:**

- Crear una interfície gràfica intuïtiva en React.js que permeta als usuaris monitoritzar les condicions de les taules en temps real i configurar les regles de reg de manera senzilla.
- Implementar funcionalitats que permeten la visualització en temps real de dades ambientals i estadístiques històriques del reg.

- **Intel·ligència Artificial:**

- Incorporar un assistent d'intel·ligència artificial que responga preguntes relacionades amb la cura de les plantes i les necessitats de reg.

2 Marc Teòric

2.1 Introducció al Marc Teòric

El propòsit d'aquest apartat és explicar de manera detallada les tecnologies i eines utilitzades en el desenvolupament del sistema **RiegAI**. Cada una de les decisions preses per a la selecció d'aquestes tecnologies té com a objectiu garantir que el projecte siga eficient, que tinga bona interconnexió i que siga escalable.

En primer lloc, es va optar per desenvolupar una **webapp** com a interfície principal del sistema. Aquesta decisió es fonamenta en la seu capacitat per a ser accessible des de qualsevol dispositiu amb connexió a internet, ja siga un ordinador personal o un dispositiu mòbil. A diferència de les aplicacions natives, que requereixen desenvolupaments específics per a cada sistema operatiu, una **webapp** elimina aquesta barrera en estar basada en navegadors web. Això no sols simplifica l'accés per als usuaris, sinó que també redueix els costos i el temps de desenvolupament, permetent centrar els esforços en la funcionalitat principal del sistema.

D'altra banda, les tecnologies seleccionades per al desenvolupament del projecte foren escollides per la seu compatibilitat i capacitat per a treballar de manera eficient entre elles. L'ús de **Node.js** i **Express** per al backend assegura un rendiment ràpid i escalable, mentre que **React.js** s'encarrega de proporcionar una interfície d'usuari dinàmica i responsiva. A més, tecnologies com **WebSockets** i **MQTT** permeten una comunicació en temps real entre els diferents components del sistema, aconseguint una sincronització eficient entre el hardware i el software.

En l'àmbit del hardware, l'elecció de l'**Arduino Nano 33 IoT** com a microcontrolador principal respon a les seues capacitats de connectivitat sense fil i la seu compatibilitat amb una àmplia varietat de sensors i actuadors. Això, combinat amb els sensors d'humitat, temperatura i nivell d'aigua, així com les vàlvules solenoides i el relé de dos canals, assegura que el sistema siga capaç de complir amb les funcions de monitoratge i automatització necessàries.

2.2 Hardware Utilitzat

2.2.1 Arduino Nano 33 IoT

El **Arduino Nano 33 IoT** és una placa de desenvolupament compacta dissenyada específicament per a projectes d'Internet de les Coses (*IoT*). *Arduino Nano 33 IoT Documentation, 2024*. Equipa un microcontrolador **Arm® Cortex®-M0+ SAMD21** de 32 bits, que opera fins a 48 MHz, oferint un equilibri entre rendiment i baix consum energètic. *Arduino Nano 33 IoT Datasheet, 2024*.

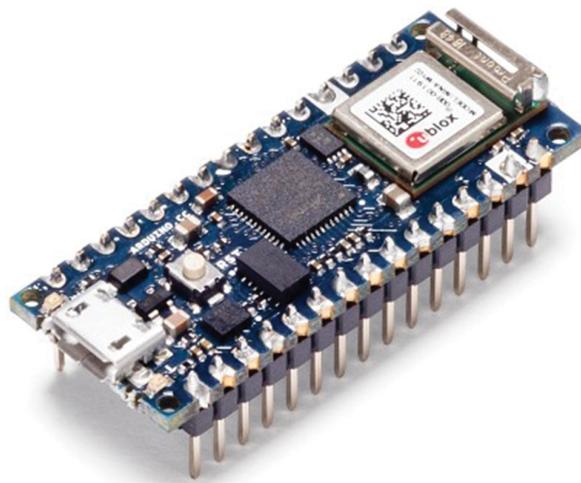


Figura 1: Placa Arduino Nano 33 IoT.

Una característica destacada d'aquesta placa és la seua connectivitat sense fils, proporcionada pel mòdul **u-blox NINA-W102**, que suporta tant **Wi-Fi** com **Bluetooth® Low Energy (BLE)**. Aquesta dualitat facilita la comunicació en temps real i la integració amb dispositius mòbils i xarxes sense fils. *Arduino Nano 33 IoT Documentation, 2024*.

Per a garantir comunicacions segures, la placa incorpora el xip **ATECC608A**, que permet l'emmagatzematge segur de certificats i claus, assegurant autenticació i encriptació robustes en aplicacions IoT. *Arduino Nano 33 IoT Documentation, 2024*.

Pel que fa a la disposició de pins, l'Arduino Nano 33 IoT ofereix les següents interfícies i funcionalitats:

- **Pins digitals (D0-D13):** Configurables com a entrades o eixides per a interactuar amb diversos components.
- **Pins analògics (A0-A7):** Permeten lectures de senyals analògiques, essencials per a sensors de variació contínua.
- **Interfícies de comunicació:**
 - **UART:** Pins D0 (RX) i D1 (TX) per a comunicació serial.

- **I2C:** Pins A4 (SDA) i A5 (SCL) per a connectar múltiples dispositius en un bus comú.
- **SPI:** Pins D11 (MOSI), D12 (MISO) i D13 (SCK) per a comunicació d’alta velocitat amb perifèrics compatibles.

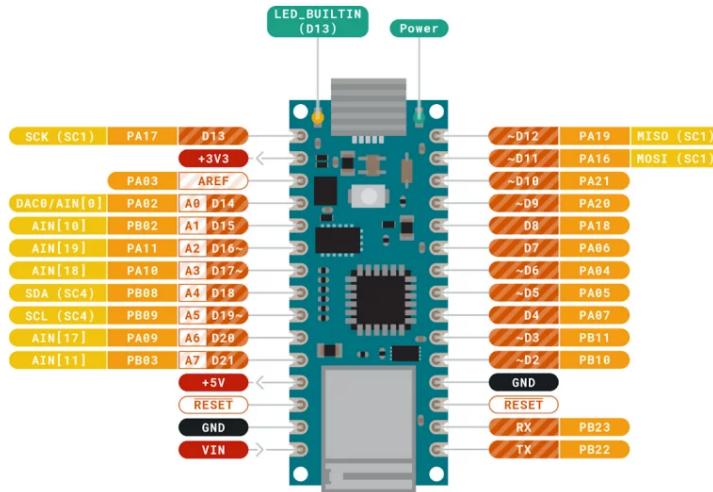


Figura 2: Disposició dels pins de l’Arduino Nano 33 IoT.

És important destacar que la placa opera a **3.3V** i no tolera nivells lògics de 5V en els seus pins d’entrada/eixida, per la qual cosa s’ha d’evitar connectar senyals de 5V directament per a prevenir danys *Arduino Nano 33 IoT Datasheet, 2024*; *Arduino Nano 33 IoT Documentation, 2024*.

2.2.2 Sensors i Actuadors

- **Sensor d'Humitat del Sòl**

El **Sensor d'Humitat del Sòl Capacitiu v1.2** és una eina per a mesurar el contingut d'humitat al sòl. A diferència dels sensors resistius tradicionals, aquest utilitza el principi de capacitància, la qual cosa incrementa significativament la seua durabilitat i precisió. A més, compta amb una capa protectora anticorrosiva, que evita els problemes d'oxidació i li atorga una vida útil d'almenys tres anys *Capacitive Soil Moisture Sensor v1.2: Característiques i funcionament, 2024*.

Aquest sensor és compatible amb microcontroladors com **Arduino**, **PIC**, **ESP8266**, i **NodeMCU**, i la seua fàcil integració el fa ideal per a aplicacions de monitoratge d'humitat en projectes d'*IoT* i sistemes de reg automatitzat *Capacitive Soil Moisture Sensor v1.2: Característiques i funcionament, 2024*.

Característiques principals

- **Model:** Capacitive Soil Moisture Sensor v1.2.
- **Voltatge d'alimentació:** 3.3V - 5V DC.
- **Corrent d'operació:** 5mA.
- **Voltatge d'eixida:** 0V a 5V (analogic).
- **Connector:** PH2.0-3P per a una connexió senzilla mitjançant cables *jumper*.
- **Compatibilitat:** Compatible amb microcontroladors com Arduino, PIC, ESP8266, NodeMCU, entre altres *Descripció tècnica del Capacitive Soil Moisture Sensor v1.2, 2024*.

Funcionament El sensor mesura la humitat del sòl basant-se en la capacitància entre dos elèctrodes integrats. La capacitància disminueix a mesura que el sòl es torna més humit i augmenta quan el sòl està sec. Proporciona un senyal analògic en funció del nivell d'humitat *Capacitive Soil Moisture Sensor v1.2: Característiques i funcionament, 2024*:

- **0V:** Indica un sòl completament humit.
- **5V:** Indica un sòl completament sec.

Pins i connexions

- **VCC:** Entrada d'alimentació, accepta entre 3.3V i 5V.
- **GND:** Connexió a terra.
- **AO (Eixida analògica):** Proporciona un voltatge proporcional al nivell d'humitat del sòl *Descripció tècnica del Capacitive Soil Moisture Sensor v1.2, 2024*.



Figura 3: Sensor d'Humitat Capacitiu v1.2 per a Monitorització del Sòl.

• Sensor de Temperatura de l'Aigua

El **DS18B20** és un sensor de temperatura digital àmpliament utilitzat per la seua precisió, fiabilitat i fàcil integració amb microcontroladors com Arduino. Gràcies al seu encapsulat d'acer inoxidable, és resistent a l'aigua, la qual cosa el converteix en una opció ideal per a mesurar la temperatura en entorns líquids com sistemes de reg automatitzat *DS18B20 Datasheet, 2024*.

Característiques principals

- **Tipus de sensor:** Digital.
- **Rang de mesura:** -55 °C a +125 °C.
- **Precisió:** ±0.5 °C en el rang de -10 °C a +85 °C.
- **Resolució ajustable:** De 9 a 12 bits.
- **Voltatge d'operació:** 3.0V a 5.5V.
- **Protocol de comunicació:** 1-Wire.
- **Encapsulat impermeable:** Sonda d'acer inoxidable amb un cable d'1 metre *Tutorials, 2024*.

Protocol de comunicació: 1-Wire El **DS18B20** utilitza el protocol **1-Wire**, que permet la transmissió de dades a través d'un únic cable de dades, facilitant la connexió de múltiples sensors en un mateix bus. Cada sensor inclou un codi únic de 64 bits que permet identificar-lo dins del sistema. Aquest protocol és ideal per a aplicacions que requereixen múltiples sensors en un entorn compacte *DS18B20 Datasheet, 2024*.

Pins i connexions El sensor DS18B20 compta amb tres cables per a la seua connexió:

- **Roig (VCC)**: Alimentació, connectat al pin de 5V o 3.3V del microcontrolador.
- **Negre (GND)**: Terra, connectat al GND del sistema.
- **Groc (Data)**: Línia de dades connectada a un pin digital del microcontrolador. Requereix una resistència pull-up de 4.7 k entre la línia de dades i VCC per a garantir una comunicació estable [Tutorials, 2024](#).



Figura 4: Sensor de temperatura DS18B20 amb encapsulat impermeable.

• Sensor de Nivell d'Aigua

El **HC-SR04** és un sensor ultrasònic àmpliament utilitzat per a mesurar distàncies de manera precisa i sense contacte físic. El seu funcionament es basa en l'emissió i recepció d'ones de so a 40 kHz, la qual cosa permet determinar la distància a objectes en un rang de 2 cm a 400 cm amb una precisió aproximada de 3 mm [Ultrasonic Ranging Module HC-SR04 Datasheet, 2024](#).

Característiques principals

- **Voltatge d'operació**: 5V DC.
- **Corrent d'operació**: 15 mA.
- **Freqüència d'operació**: 40 kHz.
- **Angle de mesura efectiu**: 15°.
- **Dimensions**: 45 mm x 20 mm x 15 mm.

Funcionament El sensor emet un senyal ultrasònic de 8 cicles a 40 kHz a través del seu transmissor. Aquest senyal viatja per l'aire i, en trobar-se amb un objecte, es reflecteix i és captat pel receptor del sensor. El temps transcorregut entre l'emissió i la recepció del senyal s'utilitza per a calcular la distància a l'objecte mitjançant la fórmula:

$$\text{Distància} = \frac{\text{Temps alt} \times 340 \text{ m/s}}{2}$$

On "Temps alt" és el període en què el senyal Echo es manté en nivell alt, i 340 m/s és la velocitat del so a l'aire *HC-SR04 Ultrasonic Sensor Module User Guide, 2024*.

Connexions i Pins El HC-SR04 compta amb quatre pins:

- **VCC**: Alimentació del sensor (5V DC).
- **Trig (Trigger)**: Entrada per a disparar el senyal ultrasònic; cal aplicar un pols TTL d'almenys 10 µs.
- **Echo**: Sortida que proporciona un pols TTL la duració del qual és proporcional al temps que tarda el senyal en tornar.
- **GND**: Connexió a terra.



Figura 5: Sensor ultrasònic HC-SR04.

• Vàlvules Solenoides DC 12V i 4A

Les vàlvules solenoides de **DC 12V i 4A** són actuadors electromecànics dissenyats per a controlar el flux de líquids mitjançant un mecanisme intern d'obertura i tancament accionat per un camp magnètic. Aquestes vàlvules són ideals per a sistemes automatitzats, com ara el reg automàtic, gràcies a la seua precisió, fiabilitat i capacitat per a gestionar un cabal controlat *Datasheet Tècnica de Vàlvules Solenoides DC 12V i 4A, 2024*.

Característiques Principals

- **Voltatge d'operació:** 12V DC.
- **Corrent nominal:** 4A.
- **Tipus de vàlvula:** Normalment tancada (NC).
- **Material del cos:** Plàstic ABS o llautó.
- **Temperatura màxima d'operació:** 80°C.
- **Pressió nominal:** 0.02 MPa a 0.8 MPa.
- **Connexions hidràuliques:** Entrades i eixides amb rosca estàndard de 1/2 polzada.

Funcionament Aquestes vàlvules funcionen mitjançant un electroimant intern que, en rebre corrent, genera un camp magnètic que mou un pistó o diafragma. Aquest moviment obri el pas de l'aigua a través de la vàlvula. En interrompre la corrent, un ressort torna el pistó a la seua posició inicial, tancant el flux *Guia d'Integració en Projectes IoT, 2024*.

Connexions Elèctriques

- **Terminal positiu (+):** Connecta al pol positiu de la font d'alimentació (12V DC).
- **Terminal negatiu (-):** Connecta al pol negatiu o terra.



Figura 6: Vàlvula solenoide DC 12V i 4A.

- **Relé de Dos Canals (SRD-05VDC-SL-C)**

El mòdul de relé de dos canals **SRD-05VDC-SL-C** permet controlar dispositius de gran corrent o voltatge mitjançant senyals de baix nivell generades per un microcontrolador. És ideal per a projectes d'automatització i control com sistemes de reg, domòtica i aplicacions industrials *Datasheet Oficial del SRD-05VDC-SL-C, 2024*.

Característiques Principals

- **Model del relé:** SRD-05VDC-SL-C.
- **Voltatge d'operació:** 5V DC.
- **Corrent d'operació:** 15-20 mA per canal.
- **Capacitat de càrrega:**
 - * **AC:** 250V, 10A.
 - * **DC:** 30V, 10A.
- **Indicadors LED:** Estat d'activació de cada relé.
- **Nombr de canals:** 2.
- **Aïllament:** Òptic per a protegir el microcontrolador *Guia d'Ús del Mòdul SRD-05VDC-SL-C, 2024*.

Pins de Connexió

- **VCC:** Alimentació, connecta a 5V DC.
- **GND:** Connexió a terra.
- **IN1 i IN2:** Controlen els relés 1 i 2 respectivament.

Terminals de Càrrega

- **COM:** Entrada per a la línia de corrent.
- **NO:** Normalment obert, es tanca en activar el relé.
- **NC:** Normalment tancat, s'obri en activar el relé.

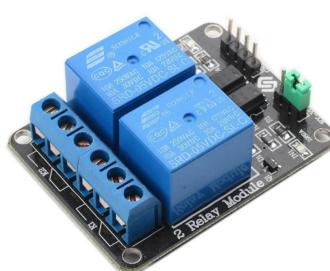


Figura 7: Mòdul de relé de dos canals SRD-05VDC-SL-C.

2.2.3 Fonts d'Alimentació

Per a garantir el funcionament continu i estable dels components del sistema **RiegAI**, s'han seleccionat dues fonts d'alimentació amb característiques específiques que satisfan les necessitats d'energia de la maquinària.

Transformador de 12V i 2A Aquest transformador és responsable de proporcionar energia a les vàlvules solenoides, que requereixen un voltatge de 12V DC per a operar. El corrent nominal de 2A assegura un subministrament adequat fins i tot sota càrrega màxima *Transformador de 12V i 2A: Full tècnic, 2024*.

Característiques Principals

- **Voltatge d'eixida:** 12V DC.
- **Corrent nominal:** 2A.
- **Ús principal:** Alimentació de les vàlvules solenoides.
- **Compatibilitat:** Dissenyat per a ser utilitzat en projectes electrònics que demanden fonts de voltatge constant i fiable.



Figura 8: Transformador de 12V i 2A utilitzat per a les vàlvules solenoides.

Transformador de 5V i 2A Aquest transformador proporciona energia al microcontrolador **Arduino Nano 33 IoT** i a altres components electrònics del sistema, com sensors i mòduls de comunicació *Transformador de 5V i 2A: Full tècnic oficial, 2024*.

Característiques Principals

- **Voltatge d'eixida:** 5V DC.
- **Corrent nominal:** 2A.
- **Ús principal:** Alimentació de l'Arduino Nano 33 IoT i altres perifèrics.
- **Compatibilitat:** Ideal per a sistemes que operen a nivells de baix voltatge, com microcontroladors i sensors.



Figura 9: Transformador de 5V i 2A utilitzat per a l'Arduino i sensors.

2.3 Tecnologies de Backend

2.3.1 Node.js i Express

Node.js és un entorn d'execució per a JavaScript construït sobre el motor V8 de Google Chrome. Dissenyat per a crear aplicacions ràpides i escalables, utilitza un model d'entrada/eixida asíncron i basat en esdeveniments, la qual cosa el fa ideal per a sistemes d'alt rendiment *Node.js v20.5.1 Documentation, 2023*.

Característiques principals de Node.js

- **I/O no bloquejant i basat en esdeveniments:** Gràcies a la seua llibreria *libuv*, permet manejar múltiples sol·licituds sense bloquejar el fil principal.
- **Motor V8:** Compila JavaScript directament a codi màquina, garantint un rendiment excel·lent.
- **Single-threaded però escalable:** Gestiona moltes connexions simultàniament mitjançant la seua arquitectura asíncrona.
- **Ecosistema de mòduls amb NPM:** Ofereix una àmplia col·lecció de mòduls i biblioteques per a facilitar el desenvolupament.
- **Multiplataforma:** Compatible amb Windows, Linux i macOS.

Casos d'ús comuns

- Aplicacions en temps real com xats.
- Servidors d'APIs RESTful i GraphQL.
- Aplicacions de transmissió de dades (*streaming*).



Figura 10: Logotip oficial de Node.js.

Express.js és un marc de treball per a Node.js que simplifica el desenvolupament d'aplicacions web i APIs en proporcionar eines per a manejar sol·licituds HTTP, rutes i *middleware* de forma eficient *Express.js Documentation, 2023*.

Característiques principals d'Express.js

- **Sistema de *middleware***: Permet afegir funcionalitats específiques com autenticació, validació o maneig d'errors.
- **Enrutament robust**: Facilita la creació i gestió de rutes HTTP.
- **Compatibilitat amb plantilles**: Suport per a motors com Pug, EJS i Handlebars.
- **Flexibilitat**: Es pot estendre fàcilment amb paquets addicionals.
- **Extensió de l'HTTP natiu**: Simplifica la interacció amb sol·licituds i respostes HTTP.

Casos d'ús comuns

- APIs RESTful.
- Aplicacions de servidor amb vistes dinàmiques.
- Aplicacions web completes amb *backend* i *frontend* integrats.

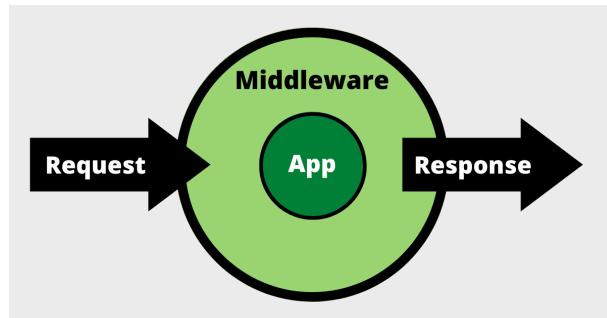


Figura 11: Logotip oficial d'Express.js.

Node.js i Express.js es complementen perfectament. Mentre que Node.js proporciona el motor per a executar JavaScript i manejar el servidor, Express afegeix una capa superior que simplifica el desenvolupament.

2.3.2 API REST

REST (*Representational State Transfer*) és un estil d'arquitectura per a sistemes hipermèdia distribuïts, introduït per primera vegada per Roy Fielding en la seua tesi doctoral l'any 2000 Fielding, 2000. Una **API REST** és una interfície de programació d'aplicacions que utilitza els principis de REST per a permetre la comunicació entre sistemes a través de protocols estàndard com HTTP.

Una **API REST** és un conjunt de regles que permeten la interacció entre aplicacions mitjançant operacions predefinides i recursos accessibles a través de *URI* (Uniform Resource Identifier). Les APIs RESTful utilitzen mètodes HTTP estàndard per a realitzar operacions sobre els recursos:

- **GET**: Recuperar una representació d'un recurs.
- **POST**: Crear un nou recurs.
- **PUT**: Actualitzar o reemplaçar un recurs existent.
- **DELETE**: Eliminar un recurs existent.

Aquestes APIs es caracteritzen per ser **sense estat** (*stateless*), la qual cosa significa que cada sol·licitud del client al servidor ha de contindre tota la informació necessària per a entendre i processar la petició. El servidor no emmagatzema cap context de la sessió del client entre sol·licituds *Understanding RESTful API Architecture*, 2024.

Les APIs REST s'utilitzen per a facilitar la comunicació entre diferents components de programari, permetent que aplicacions web, mòbils i altres serveis puguen interactuar amb servidors per a accedir, manipular i emmagatzemar dades. Són fonamentals en el desenvolupament d'aplicacions modernes, ja que permeten:

- **Interoperabilitat**: Aplicacions desenvolupades en diferents llenguatges i plataformes poden comunicar-se entre si.
- **Escalabilitat**: Gràcies a la seu naturalesa sense estat, els servidors poden manejar múltiples sol·licituds simultàniament sense sobrecarregar-se amb informació de sessió.
- **Flexibilitat**: Permeten l'evolució independent de client i servidor, sempre que es mantinga el contracte definit per l'API.

Principis fonamentals d'una API REST

- 1) **Arquitectura Client-Servidor**: Separació de responsabilitats entre client i servidor, la qual cosa permet que ambdós evolucionen de forma independent.
- 2) **Sense Estat (Stateless)**: Cada sol·licitud del client ha de contindre tota la informació necessària; el servidor no emmagatzema estat del client.
- 3) **Memòria Cau (Caché)**: Les respostes han d'indicar si són cachejables o no, optimitzant així l'ús de recursos.

- 4) **Interfície Uniforme:** Ús de mètodes i convencions estàndard per a interactuar amb els recursos.
- 5) **Sistemes en Capes:** L'arquitectura pot tindre múltiples capes, com balancejadors de càrrega, servidors intermedis, etc.
- 6) **Codi sota demanda (opcional):** El servidor pot proporcionar codi executable al client quan siga necessari.

WHAT IS A REST API?

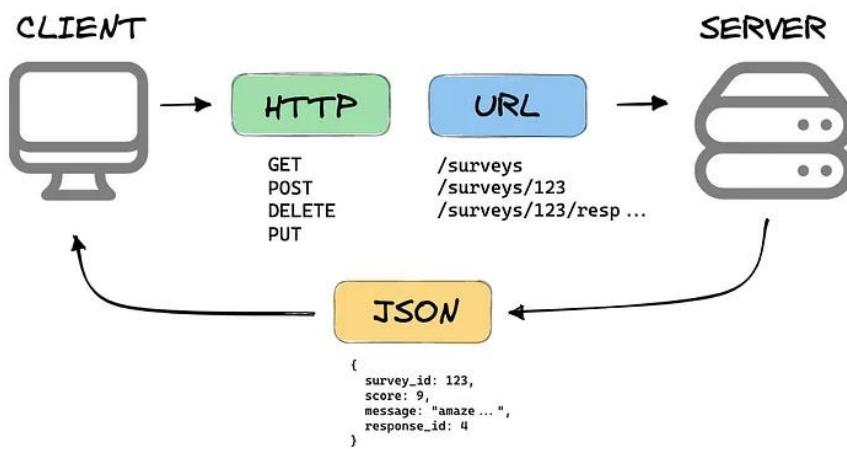


Figura 12: Diagrama conceptual d'una arquitectura REST.

2.3.3 MongoDB

MongoDB és una base de dades NoSQL orientada a documents que permet emmagatzemar i gestionar dades en un format similar a JSON. A diferència de les bases de dades relacionals tradicionals que utilitzen taules i files, MongoDB emmagatzema les dades en documents flexibles, la qual cosa facilita una representació més natural i eficient per a les aplicacions modernes *MongoDB Documentation, 2024*.

Característiques principals

- **Model de dades flexible:** Els documents de MongoDB estan compostos per parells clau-valor, la qual cosa permet modelar estructures jeràrquiques i complexes de forma intuitiva *MongoDB Documentation, 2024*.
- **Consultes ad-hoc:** Ofereix suport per a consultes avançades, incloent filtres, rangs i expressions regulars, amb la possibilitat de retornar camps específics dels documents *Querying Data in MongoDB, 2024*.
- **Índexs:** Els índexs milloren el rendiment de les consultes i es poden configurar en camps individuals o combinats *Indexes in MongoDB, 2024*.
- **Replicació:** Proporciona alta disponibilitat mitjançant conjunts de rèpliques, on cada node actua com una còpia de seguretat i pot assumir el rol de primari si és necessari *Replication in MongoDB, 2024*.
- **Sharding:** Permet escalar horitzontalment dividint les dades en múltiples fragments distribuïts en diferents servidors *Sharding in MongoDB, 2024*.
- **Agregacions:** Inclou un marc per a realitzar anàlisis de dades avançades mitjançant *pipelines*, semblant a les clausules GROUP BY de SQL *Aggregation Framework in MongoDB, 2024*.
- **Transaccions:** Des de la versió 4.0, MongoDB suporta transaccions ACID multidoument, garantint consistència i atomicitat en operacions complexes *Transactions in MongoDB, 2024*.

Usos comuns MongoDB s'utilitza en aplicacions modernes que requereixen flexibilitat, escalabilitat i un model de dades dinàmic. Entre els seus usos destaquen:

- Aplicacions de comerç electrònic que gestionen inventaris i catàlegs de productes.
- Sistemes de gestió de contingut i xarxes socials.
- Anàlisi de dades en temps real i processament de grans volums d'informació.

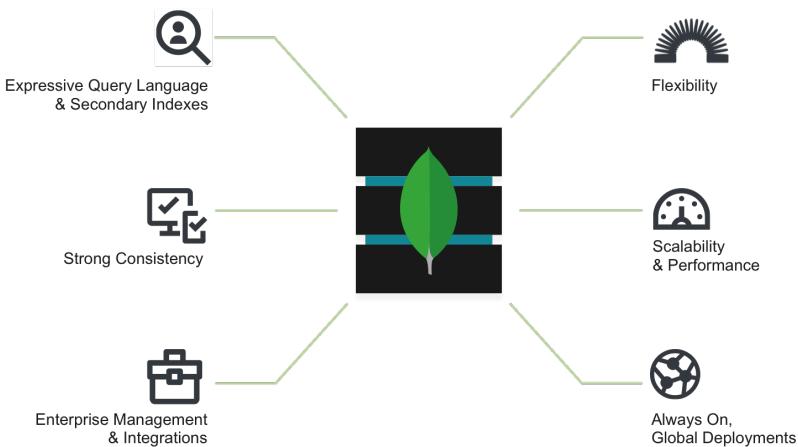


Figura 13: Estructura de dades flexible en MongoDB.

2.3.4 MQTT

Message Queuing Telemetry Transport (MQTT) és un protocol de missatgeria lleuger, basat en el model de publicació/subscripció, dissenyat per a facilitar la comunicació entre dispositius amb recursos limitats i xarxes d'amplada de banda reduïda. És especialment rellevant en l'àmbit de l'Internet de les Coses (*IoT*) per la seua eficiència i baix consum d'energia *MQTT: The Standard for IoT Messaging, 2024*.

Origen i Evolució MQTT va ser desenvolupat l'any 1999 per Andy Stanford-Clark d'IBM i Arlen Nipper d'Eurotech per a supervisar oleoductes en la indústria del petroli i gas. La necessitat d'un protocol que operara eficientment sobre enllaços satel·litals costosos i de baixa qualitat va impulsar la seua creació *¿Qué es MQTT? - Explicación del protocolo MQTT, 2024*. En 2013, IBM va alliberar MQTT 3.1 com a protocol obert, i posteriorment, en 2019, OASIS va llançar la versió 5 de MQTT, que també va ser reconeguda com a estàndard ISO (ISO/IEC 20922) «*¿Qué es MQTT? El protocolo más utilizado para IoT», 2024*.

Arquitectura i Funcionament MQTT opera sota una arquitectura de *publicació/subscripció* que involucra tres components principals:

- **Clients:** Dispositius que actuen com a publicadors (envien missatges) o subscriptors (reben missatges).
- **Broker:** Servidor que rep missatges dels publicadors i els distribueix als subscriptors interessats.
- **Temes (*Topics*):** Canals categòrics als quals els clients es subscriuen per a rebre missatges específics.

Aquesta arquitectura permet un desacoplament entre els emissors i receptors de missatges, facilitant l'escalabilitat i flexibilitat en la comunicació *«¿Qué es MQTT? El protocolo más utilizado para IoT», 2024*.

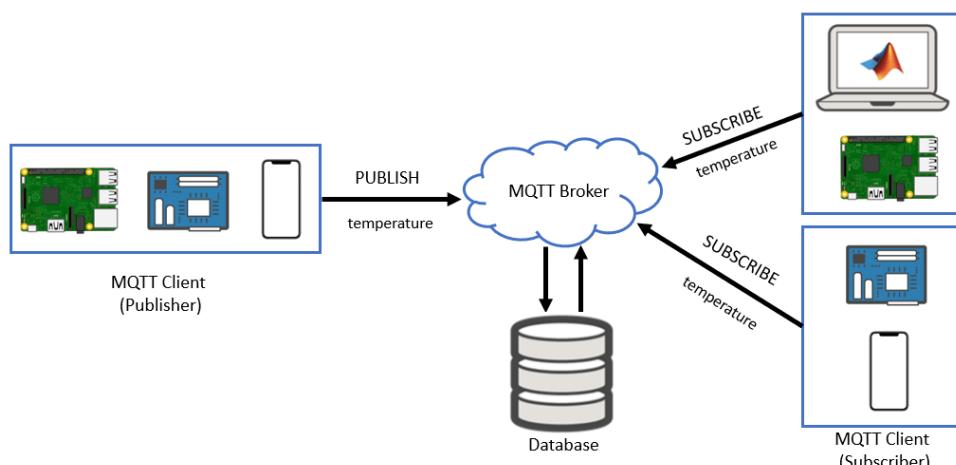


Figura 14: Arquitectura de publicació/subscripció en MQTT.

2.3.5 WebSockets

El protocol **WebSocket** permet una comunicació bidireccional, *full-duplex*, entre un client i un servidor a través d'una connexió persistent. Aquest protocol, estandarditzat per la *Internet Engineering Task Force (IETF)* com a RFC 6455 en 2011, supera les limitacions del model tradicional basat en HTTP, proporcionant una comunicació més eficient i en temps real per a aplicacions modernes Fette y Melnikov, 2011.

Característiques Principals

- **Bidireccionalitat i Full-Duplex:** Permet que tant el client com el servidor envien i reben dades de manera simultània en qualsevol moment.
- **Connexió Persistent:** Una vegada establida, la connexió roman oberta fins que alguna de les parts decideix tancar-la, eliminant la necessitat de realitzar múltiples sol·licituds HTTP.
- **Eficàcia:** Redueix la càrrega associada amb les sol·licituds HTTP en eliminar capçaleres redundants, fent els missatges més compactes i millorant el rendiment *WebSockets*, 2024.
- **Compatibilitat amb TCP:** Utilitza TCP com a protocol de transport, assegurant una comunicació fiable i ordenada.

Funcionament

- 1) **Inici de la Connexió:** El client inicia una connexió WebSocket mitjançant una sol·licitud HTTP especial (*handshake*). Si el servidor l'accepta, respon amb un codi 101 *Switching Protocols*.
- 2) **Transmissió de Missatges:** Una vegada establida la connexió, el client i el servidor poden enviar i rebre missatges en format text o binari.
- 3) **Tancament de la Connexió:** Tant el client com el servidor poden tancar la connexió en qualsevol moment.

Usos Comuns

- Aplicacions de xat en temps real.
- Sistemes de notificacions en viu.
- Actualització de dades financeres o de mercat en temps real.
- Jocs multijugador.
- Edició col·laborativa de documents i eines en línia.

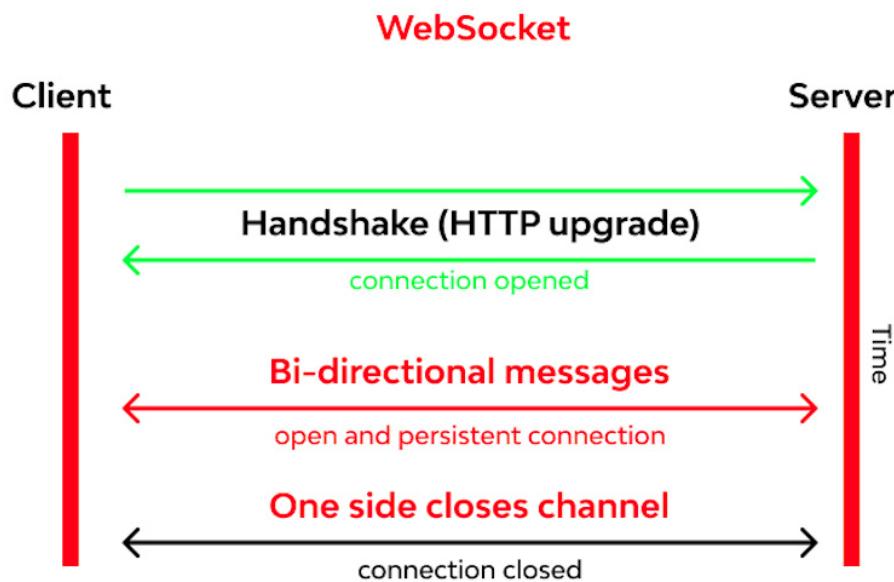


Figura 15: Funcionament del protocol WebSocket.

2.3.6 API d'OpenAI

L'API d'OpenAI és una interfície de programació d'aplicacions dissenyada per a oferir accés a models d'intel·ligència artificial avançats, com ara GPT-3.5 i GPT-4. Desenvolupada per OpenAI, aquesta tecnologia permet generar text coherent i rellevant a partir d'una consulta inicial (*prompt*), utilitzant tècniques d'aprenentatge profund basades en xarxes neuronals OpenAI, [2024c](#).

Principals característiques de l'API d'OpenAI

- **Model de llenguatge avançat:** Capacitat per a processar i generar text amb un alt nivell de comprensió semàntica i gramatical.
- **Personalització mitjançant *prompts*:** Permet especificar l'estil, el ton i el context de la resposta.
- **Escalabilitat:** Preparada per a manejar un gran volum de sol·licituds de manera eficient.
- **Compatibilitat multi-idioma:** Suporta nombrosos idiomes, la qual cosa amplia les possibilitats d'ús a nivell global.
- **Accés controlat per clau d'API:** Garanteix una integració segura amb aplicacions de tercers OpenAI, [2024d](#).

Arquitectura i flux de treball L'API funciona mitjançant un model de sol·licitud-resposta (*request-response*), on els usuaris envien una consulta en format JSON mitjançant una sol·licitud HTTP POST, i reben una resposta generada pel model. Els paràmetres ajustables, com ara `temperature` i `max_tokens`, permeten personalitzar el comportament de la generació de text OpenAI, [2024a](#).

Aplicacions comunes

- Assistència en redacció i creació de continguts.
- Suport tècnic i atenció al client automatitzada.
- Generació de codi i respostes a consultes tècniques.
- Resum i ànalisi de textos extensos.
- Suport en educació mitjançant respostes adaptades al nivell de l'usuari.

Aquesta API utilitza models d'aprenentatge profund que han sigut entrenats amb grans volums de dades, assegurant una resposta contextualment precisa i coherent OpenAI, [2024b](#).

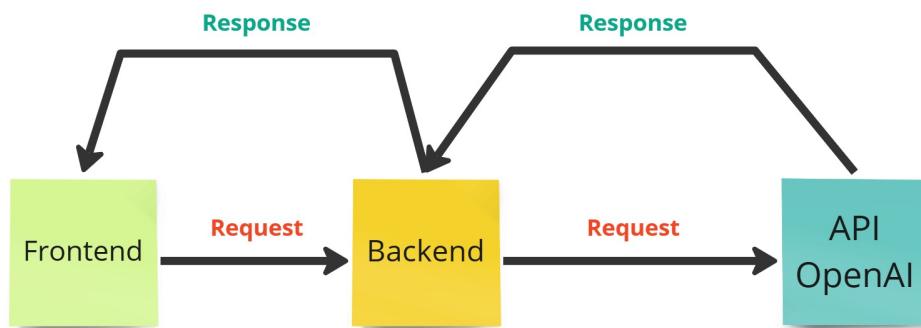


Figura 16: Flux de treball en la interacció amb l'API d'OpenAI.

2.3.7 Altres Mòduls i Llibreries

En el desenvolupament del backend de **RiegAI**, es van utilitzar diverses llibreries que faciliten operacions clau i asseguren un desenvolupament eficient i segur:

Axios: Sol·licituds HTTP **Axios** és una llibreria basada en *Promises* per a realitzar sol·licituds HTTP en navegadors i Node.js. Simplifica la integració amb APIs RESTful, permetent enviar dades, gestionar respostes i manejar errors de forma senzilla. *Axios: Promise based HTTP client for the browser and Node.js, 2024*. Les seues característiques principals inclouen:

- Suport per a interceptors de sol·licituds i respostes.
- Gestió automàtica de dades JSON.
- Configuració de temps d'espera i gestió de sol·licituds asíncrones.

Nodemailer: Enviament de Correus Electrònics **Nodemailer** és un mòdul de Node.js dissenyat per a enviar correus electrònics de manera senzilla. *Nodemailer: Send emails with Node.js, 2024*. És útil per a notificacions, restabliment de contrasenyes o confirmacions de registre. Destaca per:

- Compatibilitat amb serveis com Gmail i Outlook.
- Suport per a correus en HTML i arxius adjunts.
- Ús de protocols segurs com OAuth2 i SMTP.

Multer: Gestió d'Arxius **Multer** és un *middleware* per a gestionar la pujada d'arxius mitjançant el format `multipart/form-data`. *Multer: Middleware for handling multipart/form-data, 2024*. Ideal per a la pujada d'imatges o documents, ofereix:

- Emmagatzematge personalitzable.
- Validació de tipus d'arxiu.
- Suport per a pujar múltiples arxius simultàniament.

JWT: Autenticació Segura **JSON Web Token (JWT)** és un estàndard per a crear tokens compactes que autentiquen usuaris i asseguren la transmissió de dades. *JSON Web Token: Secure Token-Based Authentication, 2024*. Algunes característiques són:

- Signatura digital per a verificar la integritat del token.
- Compatible amb sistemes d'inici de sessió i autorització per rols.

- Estructura dividida en *Header*, *Payload* i *Signature*.

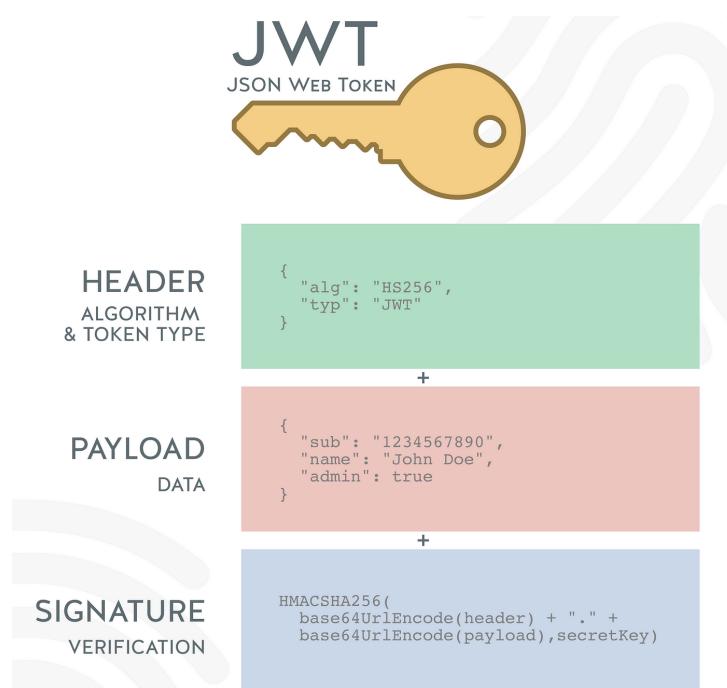


Figura 17: Estructura d'un token JWT.

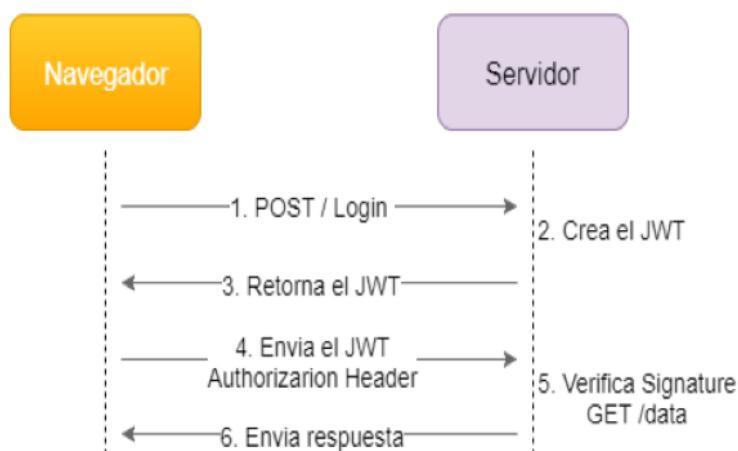


Figura 18: Creació d'un token JWT.

2.4 Tecnologies de Frontend

2.4.1 React.js

React.js, comunament conegut com a **React**, és una biblioteca de JavaScript de codi obert dissenyada per a construir interfícies d'usuari, especialment en aplicacions web d'una sola pàgina. Desenvolupada i mantinguda per Meta (anteriorment Facebook) juntament amb una comunitat de desenvolupadors, React se centra en la creació de components d'interfície d'usuari reutilitzables que faciliten el desenvolupament d'aplicacions complexes i dinàmiques *React: A JavaScript library for building user interfaces, 2024*.

Principals característiques de React:

- **Declarativa:** React permet dissenyar vistes simples per a cada estat de l'aplicació i s'encarrega d'actualitzar i renderitzar de manera eficient els components correctes quan les dades canvien. Això fa que el codi siga més previsible i fàcil de depurar *React: A JavaScript library for building user interfaces, 2024*.
- **Basada en components:** L'arquitectura de React es basa en components encapsulats que gestionen el seu propi estat. Aquests components poden combinar-se per a formar interfícies d'usuari complexes, facilitant la reutilització i el manteniment del codi *React: A JavaScript library for building user interfaces, 2024*.
- **Virtual DOM:** React utilitza un Virtual DOM, una representació en memòria del DOM real, que permet actualitzar i renderitzar de manera eficient només els components que han canviat, millorant el rendiment de l'aplicació *React: A JavaScript library for building user interfaces, 2024*.

JSX: React introduceix JSX, una extensió de la sintaxi de JavaScript que permet escriure codi similar a HTML dins de JavaScript. JSX facilita la creació de components d'interfície d'usuari en combinar la lògica i el marcat en un únic lloc, millorant la llegibilitat i mantenibilitat del codi *React Docs - Start Learning React, 2024*.

Hooks: Els **Hooks**, funcions que permeten utilitzar l'estat i altres característiques de React en components funcionals, sense necessitat d'escriure classes. Els Hooks, com `useState` i `useEffect`, simplifiquen la gestió de l'estat i els efectes secundaris en els components *React Docs - React Hooks Reference, 2024*.

Ecosistema i compatibilitat: React s'integra fàcilment amb altres biblioteques o frameworks, la qual cosa permet desenvolupar noves funcionalitats sense necessitat de reescriure el codi existent. A més, React pot renderitzar en el servidor utilitzant Node.js i potenciar aplicacions mòbils mitjançant React Native *React: A JavaScript library for building user interfaces, 2024*.

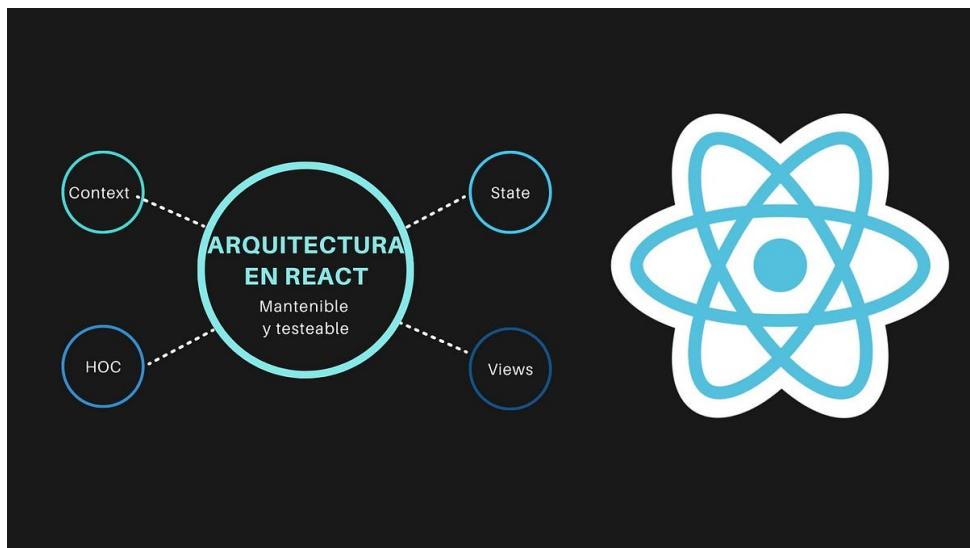


Figura 19: Arquitectura de React.

2.4.2 React Router DOM

React Router DOM és una biblioteca de JavaScript que permet implementar l'encaminament en aplicacions web desenvolupades amb React. Facilita la navegació entre diferents components o "pàgines" sense necessitat de recarregar l'aplicació, millorant així l'experiència de l'usuari *React Router: Declarative routing for React.js, 2024*. En sincronitzar la interfície d'usuari amb la URL, React Router DOM permet que l'aplicació reflectisca l'estat actual en la barra d'adreces del navegador, la qual cosa és essencial per a la navegació directa i l'ús de marcadors.

Característiques principals de React Router DOM:

- **Encaminament declaratiu:** Permet definir les rutes de l'aplicació de manera clara i lleible *React Router: Declarative routing for React.js, 2024*.
- **Rutes niades:** Facilita la creació de rutes dins d'altres rutes, útil per a aplicacions amb múltiples nivells de navegació *React Router: Declarative routing for React.js, 2024*.
- **Rutes dinàmiques:** Permet manejar paràmetres en la URL, facilitant la rendització de components basats en dades específiques proporcionades en la ruta *React Router Tutorial for Beginners, 2024*.
- **Redireccions i maneig d'errors:** Simplifica la configuració de redireccions i el maneig d'errors, millorant l'experiència de l'usuari *A guide to React Router for beginners, 2024*.

React Router DOM és una eina essencial per al desenvolupament d'aplicacions React que requereixen una navegació eficient i dinàmica. El seu enfocament declaratiu i la seua capacitat per a manejar rutes complexes el converteixen en una solució robusta per a la gestió de rutes en aplicacions modernes.

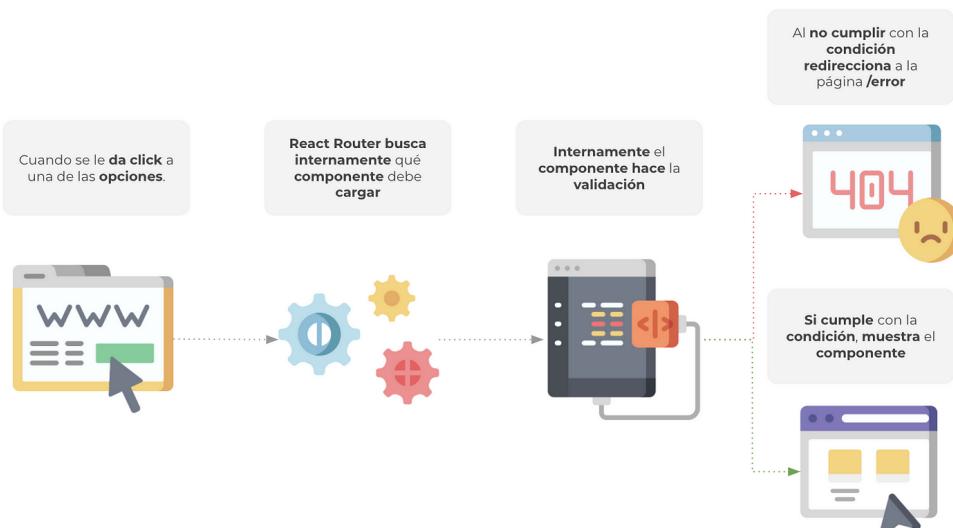


Figura 20: Flux d'encaminament amb React Router DOM.

2.4.3 React-Toastify

React-Toastify és una biblioteca de notificacions per a React que permet als desenvolupadors mostrar missatges d'alerta o feedback als usuaris de manera senzilla, atractiva i personalitzable. Aquesta eina és àmpliament utilitzada en aplicacions React per a millorar l'experiència de l'usuari en proporcionar notificacions ràpides i clares que informen sobre l'estat d'una acció o esdeveniment (èxits, errors, advertències, etc.) *React-Toastify Documentation, 2024*.

Característiques Principals

- **Integració Senzilla:** Fàcil de configurar i implementar en projectes React *React-Toastify Documentation, 2024*.
- **Personalització Completa:** Configuració de tipus de notificació, posició, duració i estils *React-Toastify on npm, 2024*.
- **Compatibilitat amb CSS:** Suport per a afegir classes CSS personalitzades o estils en línia *React-Toastify Documentation, 2024*.
- **Control Programàtic:** Mètodes per a mostrar, actualitzar i eliminar notificacions *Implementing React-Toastify, 2024*.
- **Responsiu i Accessible:** Funciona en múltiples dispositius i compleix amb estàndards d'accessibilitat *React-Toastify Documentation, 2024*.
- **Suport d'Animacions:** Animacions predefinides i personalitzables FreeCodeCamp, *2024*.

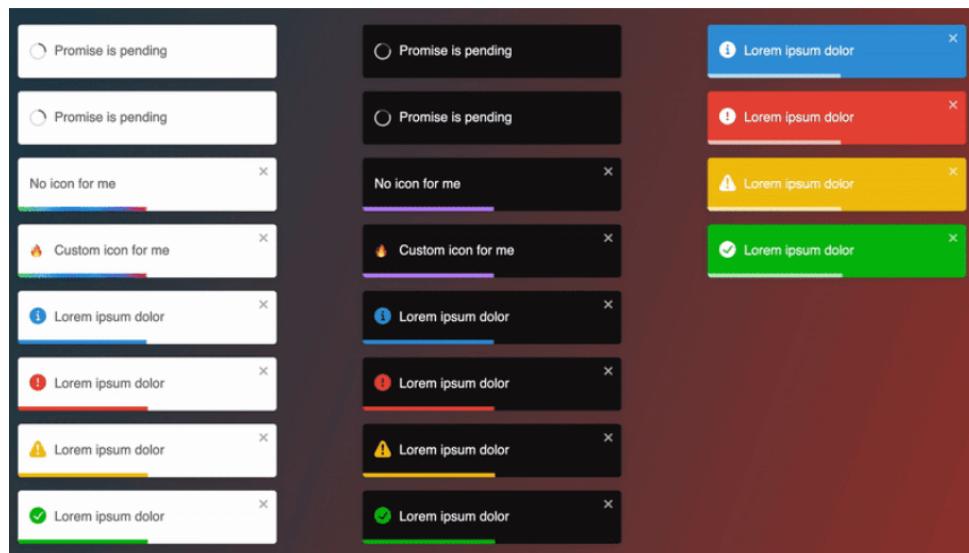


Figura 21: Exemple de notificacions amb React-Toastify.

2.5 Infraestructura de Desenvolupament

2.5.1 Arduino IDE

El **Arduino Integrated Development Environment (IDE)** és l'entorn de desenvolupament oficial d'Arduino, dissenyat per a facilitar la programació i càrrega de codi en les plaques Arduino. Aquest entorn proporciona una interfície amigable que permet als usuaris escriure, compilar i carregar programes, coneguts com "sketches", directament a les plaques de maquinari. L'Arduino IDE és compatible amb múltiples sistemes operatius, incloent Windows, macOS i Linux, i suporta una àmplia varietat de plaques Arduino *Arduino Documentation, 2024*.

Característiques principals

- **Editor de codi:** Ofereix un editor de text amb ressaltat de sintaxi i autocompletat, facilitant l'escriptura i lectura del codi font *Arduino Documentation, 2024*.
- **Compilació i càrrega:** Inclou eines integrades per a compilar el codi i carregar-lo directament en la placa Arduino connectada *Arduino Software, 2024*.
- **Gestió de biblioteques:** Permet la instal·lació i actualització de biblioteques addicionals que amplien les funcionalitats disponibles per als projectes *Arduino Libraries, 2024*.
- **Monitor serial:** Proporciona una eina per a la comunicació en sèrie entre l'ordinador i la placa, útil per a depuració i monitoratge en temps real *Arduino Documentation, 2024*.



Figura 22: Arduino IDE.

2.5.2 Mosquitto Broker

El **Eclipse Mosquitto** és un intermediari de missatges de codi obert que implementa el protocol MQTT (*Message Queuing Telemetry Transport*) en les seues versions 5.0, 3.1.1 i 3.1. MQTT és un protocol de missatgeria lleuger dissenyat per a dispositius amb recursos limitats i xarxes amb ample de banda reduït, utilitzat comunament en aplicacions d'Internet de les Coses (*IoT*) *Eclipse Mosquitto Documentation, 2024*.

Rol en la comunicació MQTT

- **Intermediari de missatges:** Rep missatges de dispositius "publicadors" i els distribueix als "subscriptors" interessats en aquests missatges, gestionant eficientment la comunicació entre múltiples dispositius *MQTT: The Standard for IoT Messaging, 2024*.
- **Escalabilitat i eficiència:** Gràcies al seu baix consum de recursos, és adequat per a dispositius de baixa potència, com ordinadors de placa única, així com en servidors d'alt rendiment *Eclipse Mosquitto Documentation, 2024*.
- **Seguretat:** Suporta autenticació i comunicació segura mitjançant TLS, garantint la integritat i confidencialitat de les dades transmeses *MQTT: The Standard for IoT Messaging, 2024*.

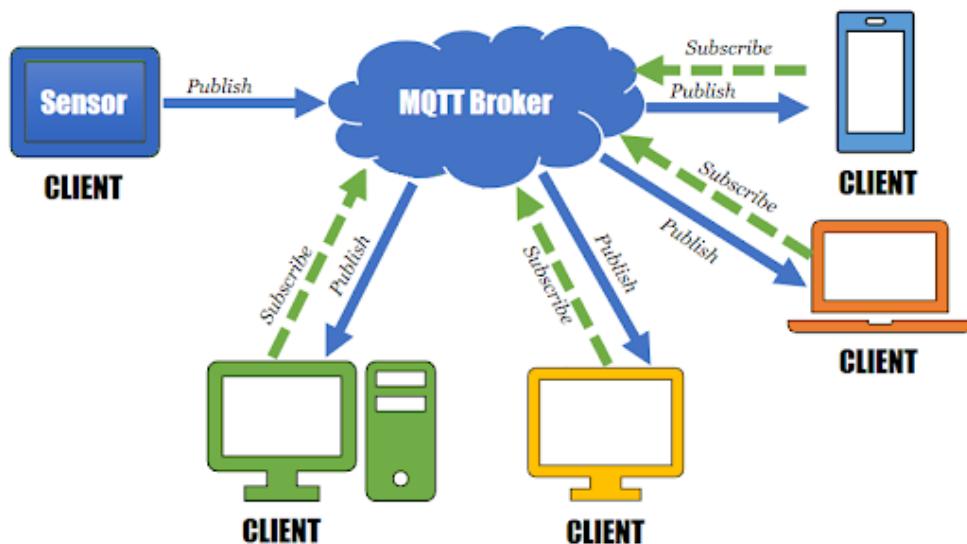


Figura 23: Arquitectura de comunicació amb Mosquitto Broker.

2.5.3 Visual Studio Code

El **Visual Studio Code (VS Code)** és un editor de codi font desenvolupat per Microsoft, optimitzat per a la construcció i depuració d'aplicacions web i al núvol. És gratuït, de codi obert i està disponible per a sistemes operatius com Linux, macOS i Windows. VS Code combina la simplicitat d'un editor de codi amb potents eines de desenvolupament, oferint una experiència lleugera i robusta per a desenvolupadors *Visual Studio Code Documentation, 2024*.

Característiques destacades

- **Editor de codi avançat:** Proporciona ressaltat de sintaxi, autocompletat intel·ligent i refactorització de codi, facilitant l'escriptura i manteniment del codi *Visual Studio Code Documentation, 2024*.
- **Depuració integrada:** Inclou eines de depuració que permeten executar i depurar codi directament des de l'editor, amb punts d'interrupció, inspecció de variables i control d'execució *Debugging in Visual Studio Code, 2024*.
- **Control de versions:** Ofereix integració nativa amb sistemes com Git, permetent gestionar repositoris, realitzar *commits* i sincronitzar canvis de manera eficient *Visual Studio Code Documentation, 2024*.
- **Extensions i personalització:** Compta amb un ampli *marketplace* d'extensions que afegeixen funcionalitats i suport per a diversos llenguatges i eines, permetent personalitzar l'entorn segons les necessitats del desenvolupador *Extensions Marketplace for VS Code, 2024*.

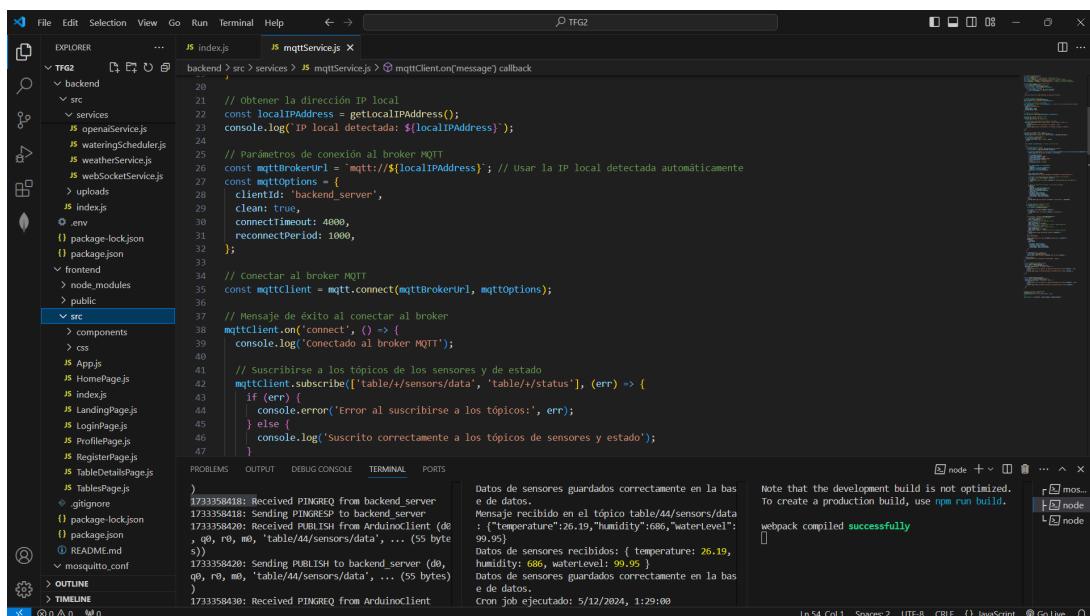


Figura 24: Entorn de treball de Visual Studio Code.

2.5.4 Postman

El **Postman** és una plataforma de col·laboració per al desenvolupament d'APIs que simplifica cada etapa del cicle de vida d'una API, permetent als desenvolupadors dissenyar, provar, documentar i monitoritzar les seues APIs de manera eficient. Inicialment concebuda com una eina per a provar sol·licituds HTTP, Postman ha evolucionat fins a convertir-se en una solució integral per a la gestió d'APIs *Postman Documentation, 2024*.

Ús per a provar i depurar APIs

- **Interfície intuïtiva:** Permet construir i enviar sol·licituds HTTP de manera senzilla, facilitant la interacció amb les APIs i la visualització de les respostes *What is Postman?, 2024*.
- **Automatització de proves:** Ofereix funcionalitats per a crear suites de proves automatitzades, assegurant que les APIs funcionen segons s'espera en diferents escenaris *Postman Documentation, 2024*.
- **Documentació automàtica:** Genera documentació detallada de les APIs, que pot ser compartida amb equips i parts interessades, millorant la comprensió i ús de les mateixes *Postman Product Overview, 2024*.
- **Col·laboració en equip:** Facilita el treball en equip mitjançant espais de treball compartits, permetent als membres col·laborar en el desenvolupament i proves d'APIs *Postman Documentation, 2024*.

Postman és una eina essencial per a desenvolupadors i equips que busquen millorar la qualitat i eficiència en el desenvolupament i manteniment d'APIs.

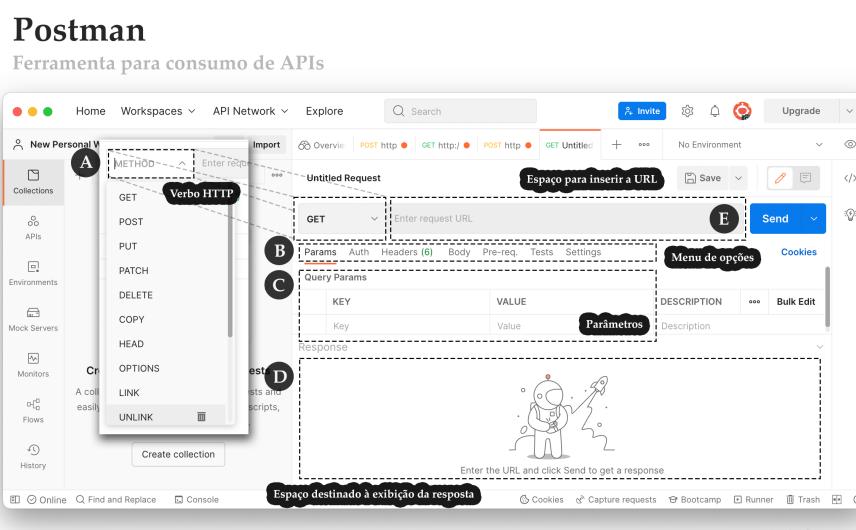


Figura 25: Entorn de Postman per a proves a APIs.

2.5.5 MongoDB Compass

El **MongoDB Compass** és una eina gràfica desenvolupada per MongoDB Inc. que permet visualitzar i explorar dades emmagatzemades en bases de dades MongoDB. Compass està dissenyat per a desenvolupadors i administradors de bases de dades que busquen una manera intuitiva d'interactuar amb les seues dades sense necessitat d'utilitzar comandos en la terminal *MongoDB Compass Documentation, 2024*.

Gestió i anàlisi de dades en la base de dades

- **Exploració visual de dades:** Permet inspeccionar col·leccions, documents i relacions entre dades de manera gràfica *MongoDB Compass Documentation, 2024*.
- **Construcció de consultes:** Inclou una interfície visual per a construir i provar consultes MongoDB utilitzant filtres i projeccions *Building Queries in MongoDB Compass, 2024*.
- **Anàlisi d'esquemes:** Mostra estructures i patrons en les dades emmagatzemades, ajudant a identificar possibles inconsistències o millorar el disseny de la base de dades *Schema Analysis in MongoDB Compass, 2024*.
- **Gestió d'índexs:** Facilita la creació i eliminació d'índexs per a optimitzar el rendiment de les consultes *Index Management in MongoDB Compass, 2024*.

MongoDB Compass és una eina indispensable per a treballar amb bases de dades MongoDB, ja que simplifica les tasques d'administració i anàlisi de dades.

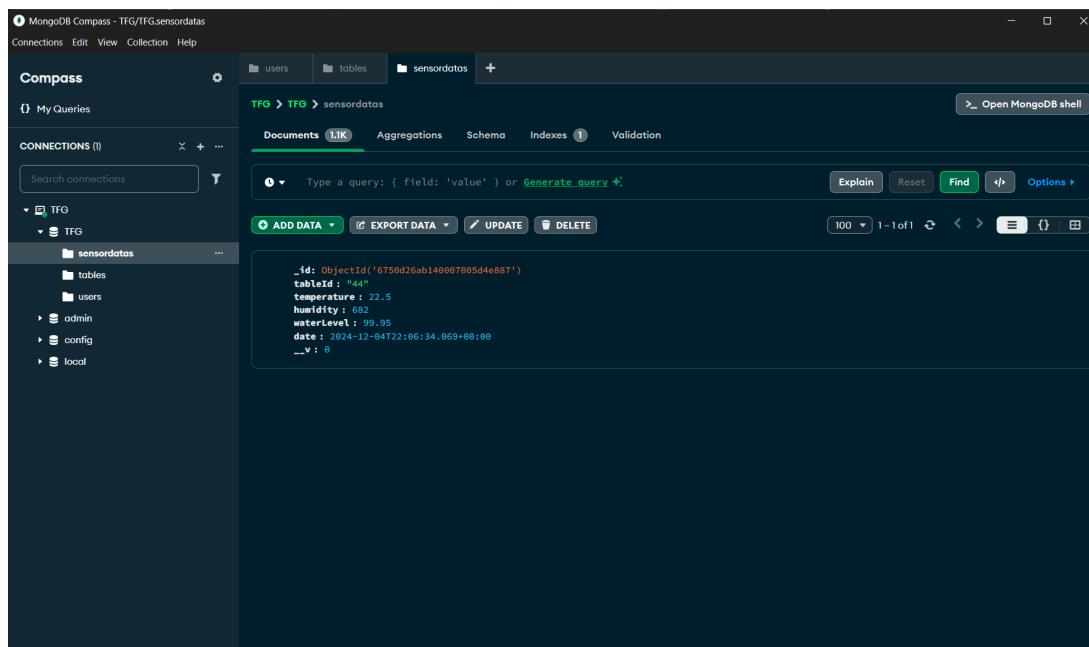


Figura 26: MongoDB Compass

3 Implementació del Sistema

3.1 Arquitectura Hardware

En aquesta secció es presenta una visió general del hardware que forma part del sistema **RiegAI**, destacant com aquests components col·laboren per assolir els objectius del projecte.

3.1.1 Descripció dels components

Arduino Nano 33 IoT

- **Introducció breu del component:** L'**Arduino Nano 33 IoT** és el microcontrolador principal del sistema **RiegAI** i actua com el cervell del projecte. Aquest dispositiu recopila i processa les dades dels sensors connectats, a més controla els actuadors, com les vàlvules solenoides, per executar les accions necessàries. Gràcies a la seu connectivitat Wi-Fi, també permet la integració amb la interfície web, possibilitant el monitoratge i control en temps real. En el nostre projecte, el seu paper és essencial per a garantir un correcte funcionament.
- **Connexió amb altres dispositius:** L'**Arduino Nano 33 IoT** està connectat als sensors i actuadors mitjançant els següents pins:
 - **Sensors:**
 - * El sensor d'humitat del sòl està connectat al pin analògic A0.
 - * El sensor de temperatura DS18B20 està connectat al pin digital D2, amb una resistència pull-up de $4.7\text{k}\Omega$.
 - * El sensor ultrasònic HC-SR04 utilitza els pins digitals D9 (Trig) i D10 (Echo).
 - **Actuadors:**
 - * El relé de 2 canals, que controla les vàlvules solenoides, està connectat als pins digitals D7 i D8.
 - **Alimentació:** El microcontrolador rep alimentació del transformador de 5V mitjançant els pins VIN i GND o bé del propi PC connectat directament mitjançant USB - microUSB.

Aquestes connexions es mostren en la **Figura 34**, on l'Arduino actua com el nucli del sistema interconnectant sensors i actuadors. A més, la distribució dels pins utilitzats per a aquesta configuració es detalla en la **Taula 2**.

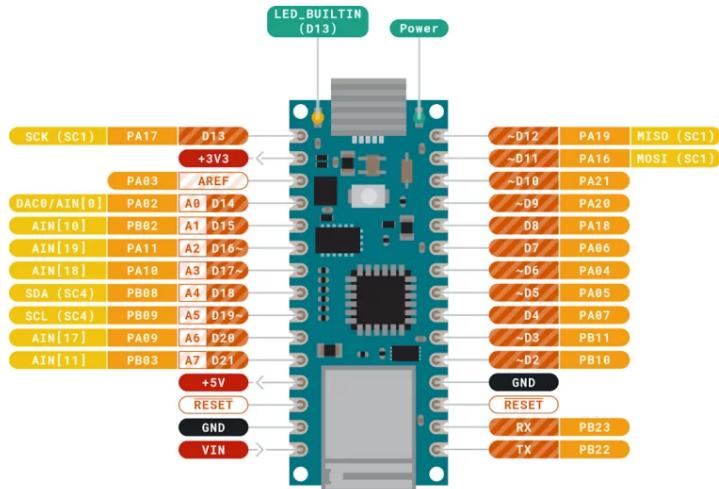


Figura 27: Disposició dels pins de l'Arduino Nano 33 IoT.

- **Justificació de la seua elecció:** Aquest model d'Arduino es va seleccionar per la seua compatibilitat amb projectes IoT, gràcies a la capacitat de connectar-se a xarxes Wi-Fi, essencial per al monitoratge i control remot. A més, destaca pel seu baix consum energètic, la facilitat d'ús amb biblioteques àmpliament compatibles i les seues dimensions compactes, que permeten una integració senzilla fins i tot en espais reduïts.

Sensors i actuadors

Sensor d'humitat del sòl:

- **Introducció breu del component:** El sensor d'humitat del sòl és un dispositiu electrònic que mesura la quantitat d'aigua present al substrat de les plantes. Aquest sensor capacitiu proporciona una lectura contínua de la humitat en percentatge. En el sistema **RiegAI**, aquest sensor és fonamental per determinar si el sòl necessita reg. Les dades proporcionades per aquest dispositiu es veuen en temps real a l'aplicació per a que el propi usuari considere si deu de regar en el moment.
- **Connexió amb altres dispositius:** Aquest sensor està connectat al pin A0 de l'Arduino, com es pot veure en la **Figura 34** i detallat en la **Taula 2**, amb alimentació de 3.3V.

Les lectures són enviades a l'Arduino, que processa les dades i les mostra en la intereficie de l'usuari en temps real per tal de prendre properes decisions com per exemple plenar o buidar la taula.



Figura 28: Sensor d'humitat del sòl amb els seus pins.

- **Justificació de la seua elecció:** Aquest sensor s'ha seleccionat per la seua fiabilitat, baix consum d'energia i capacitat per oferir lectures precises sense deteriorar-se en contacte amb l'aigua. A més, el seu cost assequible facilita la implementació a gran escala.
- **Requisits tècnics i implementació:** Rang de mesura: 0% - 100% d'humitat. Alimentació: 3.3V - 5V. El sensor s'ha d'introduir directament al sòl de la planta, assegurant que estiga correctament connectat als cables d'alimentació i senyal.

Sensor de temperatura (DS18B20):

- **Introducció breu del component:** El sensor DS18B20 és un dispositiu electrònic que mesura la temperatura de l'aigua amb alta precisió, utilitzant el protocol 1-Wire per transmetre dades digitals. En el sistema **RiegAI**, aquest sensor permet controlar la temperatura de l'aigua que circula per les taules, assegurant condicions adequades per a les plantes.
- **Connexió amb altres dispositius:** Aquest sensor es conecta al pin digital D2 i la resistència pull-up es poden visualitzar clarament en la **Figura 34**, amb els

details de la configuració indicats en la **Taula 2**i l'alimentació (3.3V). L'Arduino llegeix les dades del sensor i les mostra directament en la Interfície de la aplicació.

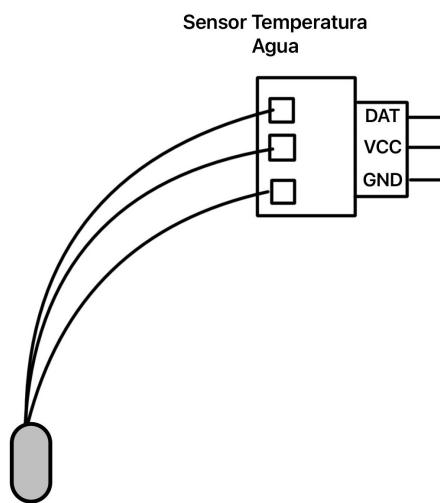


Figura 29: Sensor de temperatura amb els seus pins.

- **Justificació de la seu elecció:** Aquest sensor es va triar per la seua precisió ($+/-0.5^{\circ}\text{C}$), durabilitat i facilitat d'integració al sistema gràcies al protocol 1-Wire.
- **Requisits tècnics i implementació:** Rang de mesura: -55°C a $+125^{\circ}\text{C}$. El sensor s'ha de col·locar en una posició submergida dins de la taula per garantir les lectures correctament.

Sensor de nivell d'aigua (HC-SR04):

- **Introducció breu del component:** El **HC-SR04** és un sensor ultrasònic que mesura distàncies de manera precisa i sense contacte físic. En el sistema **RiegAI**, s'utilitza per determinar el nivell d'aigua en les taules, assegurant que aquestes no es desborden.
- **Connexió amb altres dispositius:** El sensor es connecta als següents pins de l'Arduino Nano 33 IoT:
 - **VCC:** Pin de 3.3V per a l'alimentació.
 - **GND:** Connexió a terra.
 - **Trig:** Pin digital D9 per enviar el senyal ultrasònic.
 - **Echo:** Pin digital D10 per rebre el senyal reflectit.

Aquestes connexions es poden observar en la **Figura 34** i està detallada en la **Taula 2**.

La distància calculada és enviada al microcontrolador per a prendre decisions automàtiques, com activar o desactivar les vàlvules solenoïdes.

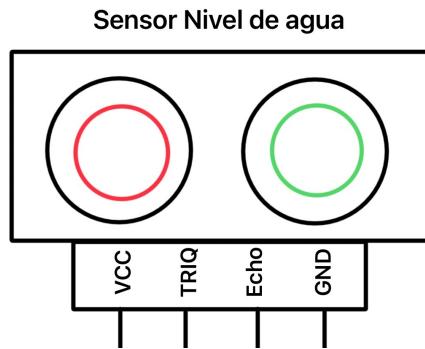


Figura 30: Sensor de ultrasons amb els seus pins.

- **Justificació de la seu elecció:** El **HC-SR04** ha estat seleccionat per la seu alta precisió, facilitat d'ús i compatibilitat amb l'Arduino. A més, ofereix un rang de mesura ampli (2 cm a 400 cm), suficient per cobrir les necessitats del projecte.
- **Requisits tècnics i implementació:** El sensor es disposa en la paret de la taula a la màxima altura i enfocat cap avall per a que aquest puga enviar les ones cap a la zona baixa de la taula i calcular la distància respecte a l'aigua quan vaja ascendint.

Relé de 2 canals:

- **Introducció breu del component:** El relé és un dispositiu electrònic que permet controlar components d'alta potència, com les vàlvules solenoides, utilitzant senyals de baix voltatge. En el sistema **RiegAI**, el relé actua com a intermediari entre l'Arduino i les vàlvules solenoides, activant o desactivant el flux d'aigua.
- **Connexió amb altres dispositius:** Connectat als pins D7 i D8 de l'Arduino per a control de les vàlvules, i alimentat amb 5V. Les connexions es mostren a la **Figura 34**, amb informació complementària disponible en la **Taula 2**.
Els canals del relé estan connectats directament a les vàlvules.

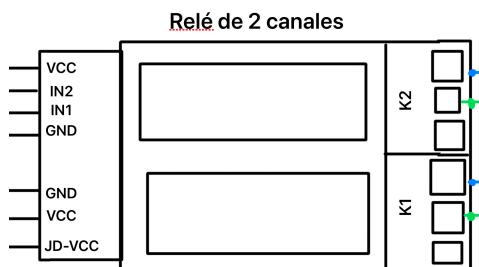


Figura 31: Relé de 2 canals amb els seus pins i connexions.

- **Justificació de la seu elecció:** Compatible amb Arduino, capaç de suportar corrents de fins a 10A, i senzill de configurar.
- **Requisits tècnics i implementació:** Corrents màxims: 10A per canal. Alimentació de control: 5V. Assegurar que els cables d'alta potència estiguin ben aïllats per evitar curtcircuits.

Vàlvules solenoides:

- **Introducció breu del component:** Les vàlvules solenoides són dispositius que controlen el flux d'aigua a través d'un sistema elèctric. En el sistema **RiegAI**, s'utilitzen dues vàlvules: una per omplir la taula i una altra per buidar-la.
- **Connexió amb altres dispositius:** Connectades al relé de 2 canals i alimentades pel transformador de 12V [Figura 34](#). El relé controla l'activació i desactivació de les vàlvules segons les ordres de l'Arduino.

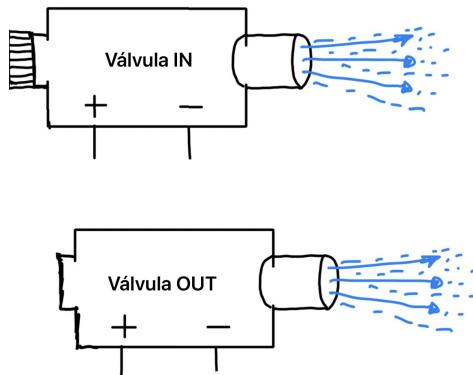


Figura 32: Valvules Solenoide de entrada i eixida amb els seus pols + i -.

- **Justificació de la seu elecció:** Són fiables, tenen una llarga vida útil i són capaces de suportar les pressions requerides en el sistema.
- **Requisits tècnics i configuració:** Tensió d'operació: 12V. Corrents màxims: 4A. Assegurar connexions segures als tubs i als cables d'alimentació.

Fonts d'alimentació

Transformador de 12V:

- **Introducció breu del component:** Proporciona energia per a les vàlvules solenoides, garantint el funcionament òptim del sistema.
- **Connexió amb altres dispositius:** Alimenta directament les vàlvules solenoides a través del relé de 2 canals, es detallen en la [Figura 34](#).
- **Requisits tècnics i configuració:** Tensions d'entrada: 110-220V AC. Sortida: 12V DC amb capacitat de corrent de 4A.

Transformador de 5V:

- **Introducció breu del component:** Alimenta el microcontrolador Arduino i els sensors connectats al sistema.
- **Connexió amb altres dispositius:** Proporciona una tensió constant de 5V a l'Arduino i altres dispositius de baix consum, es detallen en la [Figura 34](#).
- **Requisits tècnics i configuració:** Tensions d'entrada: 110-220V AC. Sortida: 5V DC amb capacitat de corrent de 2A.

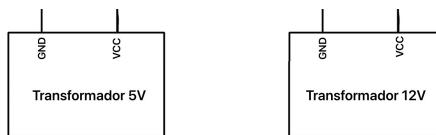


Figura 33: Transformadors de 5 i 12V amb els respectius pols.

3.1.2 Esquemes i tables dels components

Es presenten els esquemes de connexió de tots els components Hardware. També es presenten esquemes individuals dels ports i pins de cadascún dels components per separat.

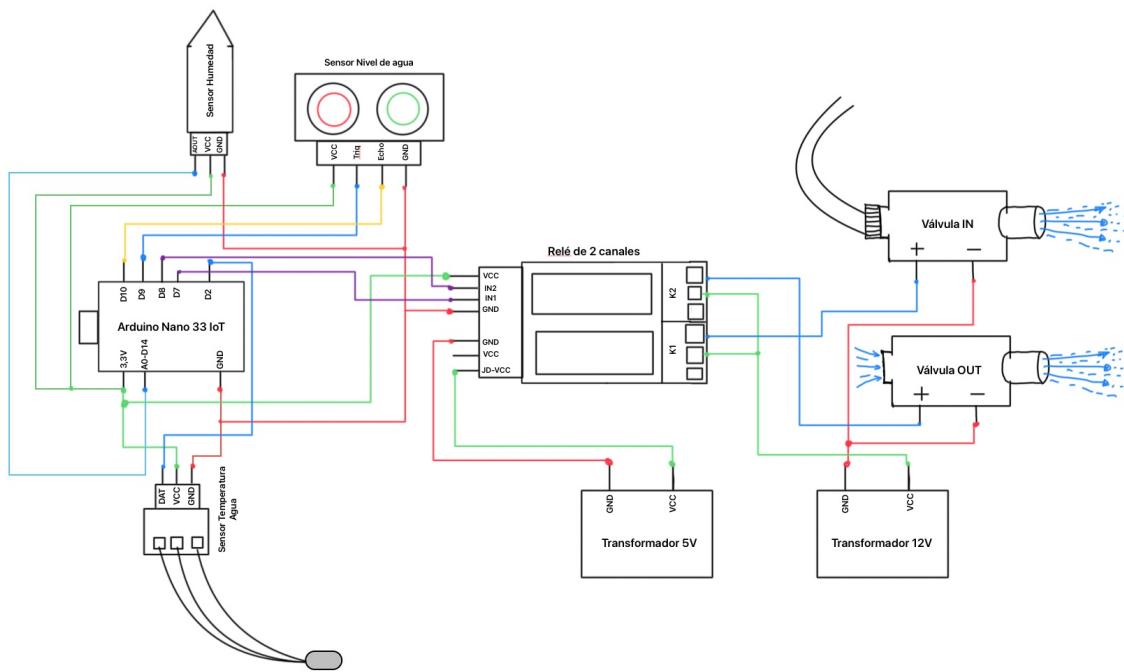


Figura 34: Esquema de connexió dels components Hardware.

Pin	Ús	Descripció
D2	ONE_WIRE_BUS	Connexió del sensor de temperatura DS18B20 mitjançant el protocol OneWire.
A0	soilMoisturePin	Lectura del sensor de humitat del sòl (Capacitive Soil Moisture Sensor v1.2).
D9	TRIG_PIN	Pin de dispar del sensor ultrasònic (HC-SR04).
D10	ECHO_PIN	Pin de recepció del sensor ultrasònic (HC-SR04).
D7	PIN_FILL	Control de la vàlvula de ompliment mitjançant el mòdul de relé.
D8	PIN_DRAIN	Control de la vàlvula de buidatge mitjançant el mòdul de relé.

Tabla 2: Assignació de pins i ús en el sistema RiegAI.

3.2 Implementació del Software

Parts Principals El sistema està dividit en les següents parts, cadascuna amb un rol essencial per al correcte funcionament del projecte **RiegAI**:

Frontend: És la interfície gràfica que permet als usuaris interactuar amb el sistema de forma visual i intuïtiva. Desenvolupada amb **React.js**, ofereix una experiència **UX/UI** moderna i responsive, optimitzada tant per a dispositius mòbils com per a escriptoris. Les principals funcionalitats que proporciona són:

- Gestionar les taules, incloent la creació, modificació i visualització dels estats.
- Visualitzar en temps real les dades dels sensors, com la humitat del sòl, temperatura de l'aigua i nivell d'aigua.
- Controlar les vàlvules solenoïdes per al reg automàtic o el buidatge d'aigua.
- Permetre l'actualització del perfil de l'usuari, incloent la foto de perfil.
- Mostrar estadístiques dels sensors per facilitar l'anàlisi.

Aquesta part es comunica amb el backend mitjançant **API REST** i **WebSockets** per garantir una interacció fluida i en temps real.

Backend: És el motor del sistema, encarregat de gestionar la lògica del projecte, la comunicació amb el hardware i el control de les dades dels usuaris. Desenvolupat amb **Node.js** i el framework **Express**, el backend actua com a intermediari entre el frontend, la base de dades i el hardware. Les seues principals responsabilitats inclouen:

- Gestionar la informació dels usuaris, com noms, correus electrònics i contrasenyes de forma segura, utilitzant **JWT** per a l'autenticació.
- Coordinar la comunicació amb el hardware a través del protocol **MQTT**, enviant ordres a l'Arduino i rebent dades dels sensors.
- Proporcionar comunicació en temps real amb el frontend mitjançant **WebSockets**.
- Gestionar operacions amb les taules, com el control de vàlvules, obtenció de dades i estadístiques històriques.
- Enviar notificacions per correu electrònic mitjançant **Nodemailer**.

Base de Dades: És el nucli d'emmagatzematge del sistema, on es guarda tota la informació relacionada amb usuaris, taules i dades dels sensors. S'utilitza **MongoDB**, una base de dades NoSQL que proporciona flexibilitat per a emmagatzemar dades estructurades i no estructurades de manera eficient. Les col·leccions principals de la base de dades són:

- **users:** Conté la informació dels usuaris, com el nom, correu electrònic, contrasenya encriptada, imatge de perfil i configuracions personals.
- **tables:** Emmagatzema les taules creades pels usuaris, incloent la seua configuració, sensors associats i estat actual.
- **sensorData:** Registra les dades històriques dels sensors, permetent generar gràfics i consultes avançades sobre el rendiment del sistema.

Aquesta base de dades permet consultar i actualitzar la informació en temps real gràcies a la integració amb **WebSockets**.

Arquitectura General El sistema segueix un model **client-servidor**, amb comunicació en temps real utilitzant **WebSockets** i **MQTT**. Aquestes tecnologies permeten l'intercanvi immediat de dades entre el backend i els dispositius del hardware, així com amb el frontend. La següent estructura defineix com interactuen els diferents components:

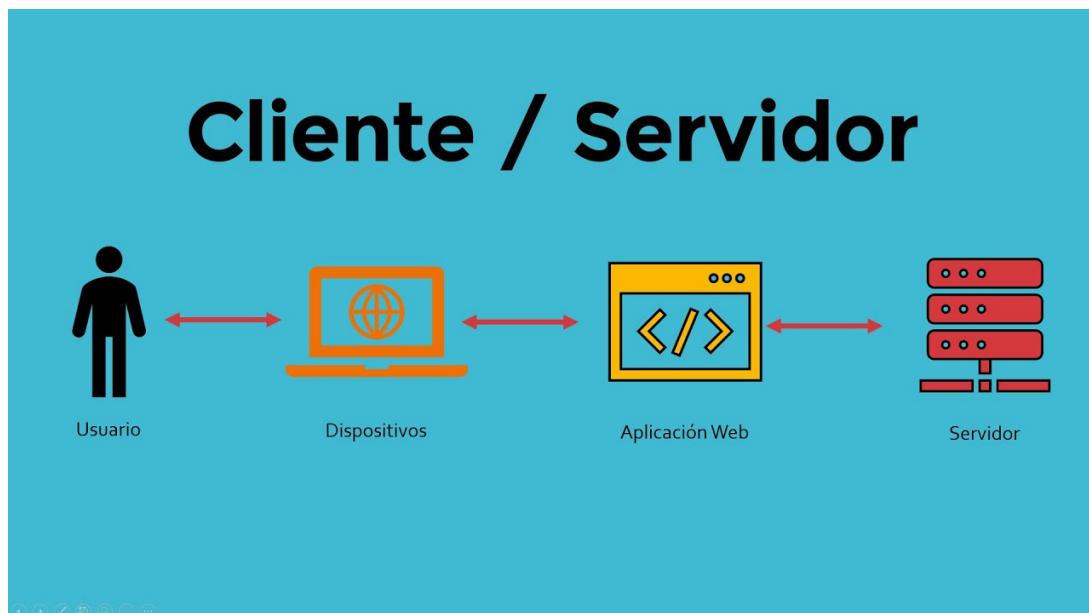
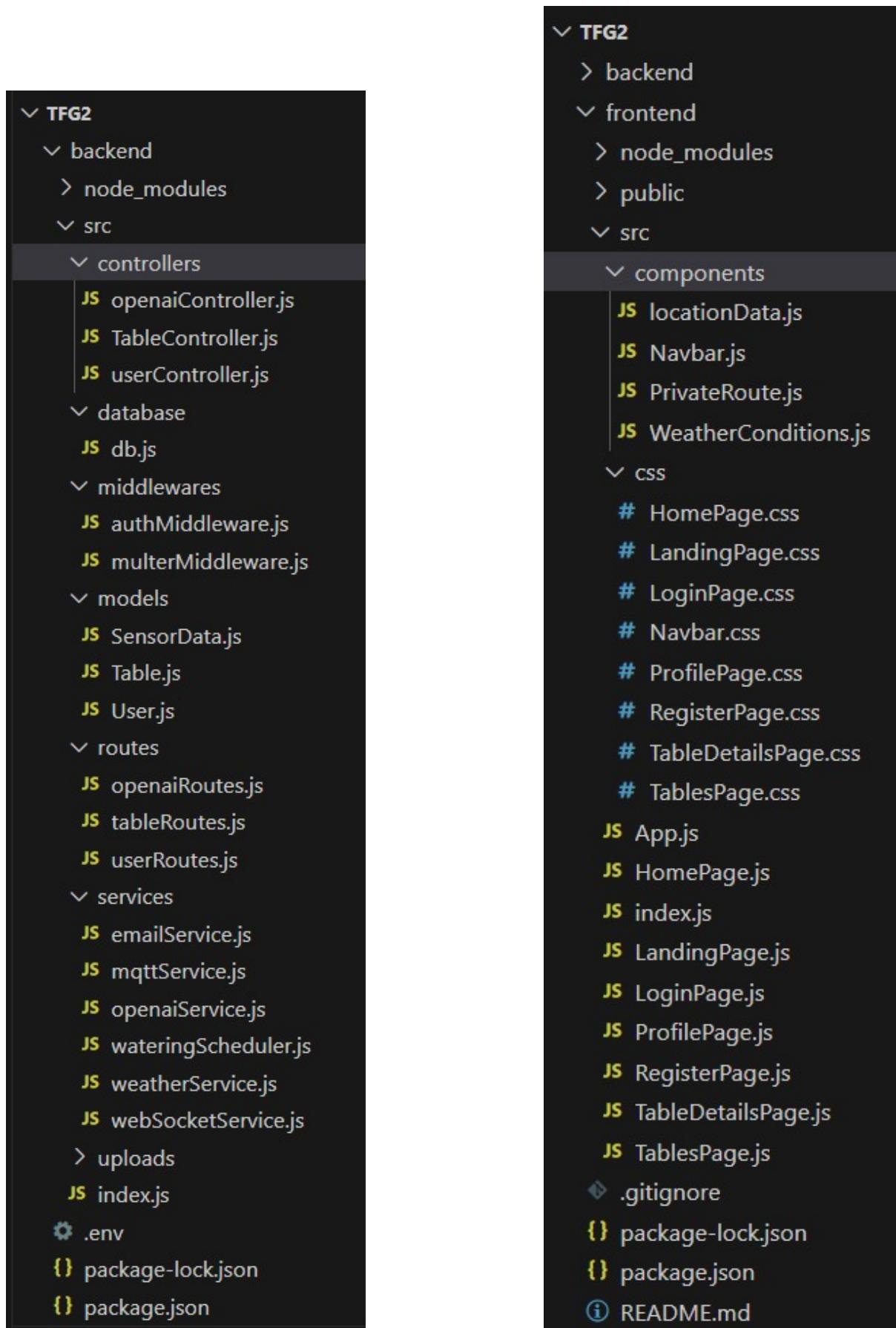


Figura 35: Diagrama d'Arquitectura del Software.

Àrbre de Directori El projecte es troba organitzat de manera clara per a facilitar el desenvolupament, manteniment i extensió del sistema. Aquesta estructura s'ha dissenyat amb una clara separació de responsabilitats, tal com es pot observar en les imatges de l'àrbre de directori (Figura 36a, Figura 36b i Figura 37). L'estructura general es detalla a continuació:

- **Backend:** Conté tota la lògica del servidor desenvolupada amb **Node.js**, la gestió de les operacions amb el hardware i l'API REST (Figura 36a).
 - *controllers/*: Controladors per a operacions relacionades amb els usuaris, les taules amb els sensors i finalment operacions amb el assistent de Inteligència Artificial.
 - *database/*: Arxiu de connexió del backend amb la base de dades de MongoDB.
 - *middlewares/*: Processos intermedis, com l'autenticació i la manipulació d'arxius.
 - *models/*: Definició d'esquemes per a la base de dades MongoDB. Inclou models com el de dades de sensors, taules i usuaris.
 - *routes/*: Rutes per gestionar els mètodes dels usuaris, les taules amb els sensors i finalment el assistent de Inteligència Artificial.
 - *services/*: Serveis per a poder utilitzar les funcionalitats de tota l'aplicació, com per exemple la comunicació MQTT, la obtenció del temps meteorològic o fins i tot la funcionalitat de programació del reg.
 - *uploads/*: Carpeta destinada a emmagatzemar arxius carregats, com imatges de perfil i fotos associades a les taules.
- **Frontend:** Desenvolupat amb **React.js**, el frontend ofereix una interfície gràfica moderna i responsiva (Figura 36b).
 - *public/*: Conté recursos estàtics com el favicon amb imatges i el fitxer HTML principal.
 - *components/*: Inclou els components principals reutilitzables com la barra de navegació (Navbar) i pàgines amb rutes protegides per asegurar la seguretat del usuari.
 - *css/*: Arxius CSS per estilitzar les pàgines i components, cada pàgina té el seu fitxer associat.
- **Hardware:** Aquesta carpeta inclou el codi específic per al microcontrolador Arduino (Figura 37).
 - *arduino-connection.ino*: Conté el firmware desenvolupat per gestionar els sensors i actuadors connectats al microcontrolador.
- **Altres:** Inclou elements auxiliars necessaris per al projecte (Figura 37).
 - *mosquitto-conf/*: Configuració del broker MQTT per a la comunicació amb dispositius físics.



(a) Estructura de l'arbre de directori del Backend.

(b) Estructura de l'arbre de directori del Frontend.

Figura 36: Comparativa entre les estructures del Backend i Frontend.

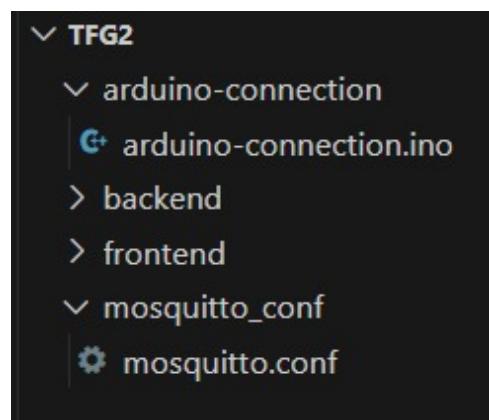


Figura 37: Estructura de l'arbre de directori general amb visualització de carpeta Hardware i Altres configuracions.

3.2.1 Backend

Tecnologies Usades El backend del sistema està desenvolupat amb un conjunt de tecnologies i biblioteques que permeten gestionar la lògica del sistema, la comunicació amb el hardware i el frontend, així com el processament de dades i la seguretat. Les principals tecnologies utilitzades són les següents:

- **Frameworks:**

- **Node.js:** Plataforma de desenvolupament basada en JavaScript, utilitzada per implementar el servidor del backend. La seua arquitectura asíncrona i orientada a esdeveniments permet gestionar múltiples operacions simultàniament.
- **Express.js:** Framework per a Node.js que simplifica la creació de rutes i middleware, utilitzat per implementar l'API RESTful que connecta el frontend amb el sistema.

- **Serveis Auxiliars:**

- **MQTT:** Protocol lleuger de missatgeria, utilitzat per comunicar-se amb l'Arduino. Permet enviar ordres als actuadors (com les vàlvules solenoïdes) i rebre dades dels sensors en temps real.
- **WebSockets:** Implementats amb la biblioteca **ws**, permeten la comunicació bidireccional en temps real entre el backend i el frontend. Això garanteix que l'usuari puga veure les actualitzacions dels sensors i l'estat de les vàlvules immediatament.
- **Nodemailer:** Biblioteca per a l'enviament de correus electrònics. S'utilitza en el projecte per gestionar la recuperació de contrasenyes.
- **Node-Cron:** Permet programar tasques periòdiques, com la programació de l'emplenament i del buidament de les taules.

- **Biblioteques de Suport:**

- **Mongoose:** Biblioteca per a gestionar la base de dades **MongoDB**, incloent la creació d'esquemes i operacions amb les col·leccions.
- **jsonwebtoken (JWT):** Utilitzat per implementar l'autenticació d'usuaris mitjançant tokens segurs.
- **bcryptjs:** Biblioteca per a l'encriptació de contrasenyes, garantint la seguretat de la informació sensible dels usuaris.
- **dotenv:** Utilitzada per carregar variables d'entorn, com la clau necessària per enllaçar-se a la API de OpenAI i poder utilitzar el Asistent de IA.
- **Multer:** Middleware per a la gestió d'arxius, utilitzat per permetre la pujada d'imatges al servidor, com fotos de perfil o imatges de taules.
- **Axios:** Biblioteca HTTP que facilita la comunicació amb serveis externs, com l'API de clima.

- **Entorn de Desenvolupament:**

- **Nodemon:** Utilitzat durant el desenvolupament per reiniciar automàticament el servidor quan es detecten canvis al codi.

Estructura del Backend El backend està organitzat en diverses carpetes que gestionen de manera modular les diferents funcionalitats del sistema. A continuació, es descriu cada carpeta i els documents que conté [Directori Backend](#).

Controllers (backend/src/controllers/) Els controladors contenen la lògica principal de les operacions associades a les rutes definides. A continuació, es descriu el controlador **TableController.js**:

- **TableController.js:** Aquest document gestiona totes les operacions relacionades amb les taules del sistema **RiegAI**. Entre les funcionalitats destacades es troben:
 - **createTable:** Aquest mètode s'encarrega de crear una nova taula. Comprova si ja existeix una taula amb el mateix nom associada a l'usuari i, si no és així, genera un identificador únic de dos díigits. Després guarda la taula a la base de dades MongoDB, afegint informació com el nom, descripció i l'usuari al qual pertany.

```
const createTable = async (req, res) => {
  const { name, description } = req.body;
  const userId = req.userId;

  try {
    const existingTable = await Table.findOne({ name, users: userId });
    if (existingTable) {
      return res.status(400).json({ message: 'Ya existe una mesa con ese nombre.' });
    }

    let tableId;
    let tableExists = true;

    while (tableExists) {
      tableId = Math.floor(10 + Math.random() * 90).toString();
      tableExists = await Table.findOne({ tableId });
    }

    const table = new Table({
      tableId,
      name,
      description,
      users: [userId],
    });

    await table.save();
    res.status(201).json({ message: 'Mesa creada con éxito', table });
  } catch (error) {
    console.error('Error creando la mesa:', error);
    res.status(500).json({ message: 'Error creando la mesa', error });
  }
};
```

Figura 38: Codi del mètode `createTable` al controlador.

- **uploadTableImage:** Aquest mètode permet actualitzar la imatge d'una taula específica. Comprova si la taula existeix i, si és així, guarda la ruta de la nova imatge al camp `tableImage` de la base de dades, vinculant-la amb la taula.

```
const uploadTableImage = async (req, res) => {
  try {
    const { tableId } = req.params;
    const table = await Table.findOne({ tableId });

    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada' });
    }

    table.tableImage = `/uploads/${req.file.filename}`;
    await table.save();

    res.status(200).json({ message: 'Imagen de mesa actualizada con éxito', tableImage: table.tableImage });
  } catch (error) {
    console.error('Error actualizando la imagen de la mesa:', error);
    res.status(500).json({ message: 'Error actualizando la imagen de la mesa', error });
  }
};
```

Figura 39: Codi del mètode `uploadTableImage` al controlador.

- **updateTableName:** Aquest mètode actualitza el nom d’una taula específica per l’usuari autenticat. Verifica que la taula existeixi i pertanyi a l’usuari abans de modificar el seu nom i guardar els canvis a la base de dades.

```
const updateTableName = async (req, res) => {
  const { tableId } = req.params;
  const { name } = req.body;
  const userId = req.userId; // Usuario autenticado

  try {
    // Verificar si la mesa existe y pertenece al usuario usando tableId de dos dígitos
    const table = await Table.findOne({ tableId, users: userId });
    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada o el usuario no tiene acceso' });
    }

    // Actualizar el nombre
    table.name = name;
    await table.save(); // Guardar los cambios

    res.status(200).json({ message: 'Nombre de la mesa actualizado correctamente', table });
  } catch (error) {
    console.error('Error actualizando el nombre de la mesa:', error);
    res.status(500).json({ message: 'Error actualizando el nombre de la mesa', error });
  }
};
```

Figura 40: Codi del mètode updateTableName al controlador.

- **updateTableDescription:** Similar al mètode anterior, aquest s’encarrega d’actualitzar la descripció d’una taula després de verificar que l’usuari autenticat té accés a aquesta taula.

```
const updateTableDescription = async (req, res) => {
  const { tableId } = req.params;
  const { description } = req.body;
  const userId = req.userId; // Usuario autenticado

  try {
    // Verificar si la mesa existe y pertenece al usuario usando tableId
    const table = await Table.findOne({ tableId, users: userId });
    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada o el usuario no tiene acceso' });
    }

    // Actualizar la descripción
    table.description = description;
    await table.save(); // Guardar los cambios

    res.status(200).json({ message: 'Descripción de la mesa actualizada correctamente', table });
  } catch (error) {
    console.error('Error actualizando la descripción de la mesa:', error);
    res.status(500).json({ message: 'Error actualizando la descripción de la mesa', error });
  }
};
```

Figura 41: Codi del mètode updateTableDescription al controlador.

- **deleteTable:** Aquest mètode elimina una taula específica i els seus dades de sensors associats. Comprova que la taula pertanyi a l’usuari autenticat abans d’eliminar-la de la base de dades.

```
const deleteTable = async (req, res) => {
  const { tableId } = req.params;
  const userId = req.userId;

  try {
    // Buscar la mesa usando tableId y asegurarse de que pertenece al usuario
    const table = await Table.findOne({ tableId, users: userId });

    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada o el usuario no tiene acceso' });
    }

    // Eliminar los datos de sensores asociados a la mesa
    await SensorData.deleteMany({ tableId });

    // Eliminar la mesa
    await Table.findOneAndDelete({ tableId, users: userId });

    res.status(200).json({ message: 'Mesa y datos de sensores eliminados con éxito' });
  } catch (error) {
    console.error('Error eliminando mesa o datos de sensores:', error);
    res.status(500).json({ message: 'Error eliminando la mesa o los datos de sensores', error });
  }
};
```

Figura 42: Codi del mètode deleteTable al controlador.

- **saveSensorData:** Permet emmagatzemar dades dels sensors (temperatura, humitat i nivell d'aigua) a la base de dades MongoDB. Crea un nou registre associat al `tableId` de la taula corresponent.

```
const saveSensorData = async (req, res) => {
  const { tableId, temperature, humidity, waterLevel } = req.body; // Los datos de los sensores

  try {
    const sensorData = new SensorData({
      tableId, // Usamos el tableId de dos dígitos
      temperature,
      humidity,
      waterLevel,
    });

    await sensorData.save();
    res.status(201).json({ message: 'Datos del sensor guardados correctamente' });
  } catch (error) {
    res.status(500).json({ message: 'Error al guardar los datos del sensor', error });
  }
};
```

Figura 43: Codi del mètode `saveSensorData` al controlador.

- **getSensorData:** Recupera les dades dels sensors associats a una taula específica. Retorna els últims 100 registres ordenats per data en ordre descendent.

```
const getSensorData = async (req, res) => {
  const { tableId } = req.params;

  try {
    // Recuperar los datos del sensor asociados al tableId de dos dígitos
    const data = await SensorData.find({ tableId }).sort({ date: -1 }).limit(100); // Últimos 100 datos
    res.status(200).json(data);
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener los datos del sensor', error });
  }
};
```

Figura 44: Codi del mètode `getSensorData` al controlador.

- **getUserTables:** Recupera totes les taules associades a l'usuari autenticat. Aquest mètode s'utilitza per mostrar al frontend totes les taules d'un usuari.

```
const getUserTables = async (req, res) => {
  const userId = req.userId;
  try {
    const tables = await Table.find({ users: userId });
    res.status(200).json(tables);
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener las mesas', error });
  }
};
```

Figura 45: Codi del mètode `getUserTables` al controlador.

- **getTableDetails:** Retorna els detalls d'una taula específica identificada pel seu `tableId`. Això inclou informació com el nom, descripció i configuracions específiques.

```
const getTableDetails = async (req, res) => {
  const { tableId } = req.params;

  try {
    // Buscar la mesa usando tableId en lugar de _id
    const table = await Table.findOne({ tableId });
    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada' });
    }

    res.status(200).json(table);
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener los detalles de la mesa', error });
  }
};
```

Figura 46: Codi del mètode `getTableDetails` al controlador.

- **updateWateringSchedule:** Actualitza la planificació del reg per a una taula. Permet establir dies específics i horaris per a omplir o buidar la taula. Els dies es guarden en minúscules i sense duplicats per garantir la consistència de les dades.

```
const updateWateringSchedule = async (req, res) => {
  const { tableId } = req.params;
  const { days, fillTime, drainTime } = req.body; // Extraemos los datos del cuerpo de la petición

  try {
    // Buscar la mesa por su tableId
    const table = await Table.findOne({ tableId });
    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada' });
    }

    // Convertir los días a minúsculas y eliminar duplicados
    const uniqueDays = [...new Set(days.map(day => day.toLowerCase()))];

    // Actualizar el plan de riego
    table.wateringSchedule = {
      days: uniqueDays, // Guardamos los días en minúsculas
      fillTime,
      drainTime,
      updatedAt: new Date(),
    };

    await table.save(); // Guardar los cambios
    res.status(200).json({ message: 'Plan de riego actualizado correctamente', wateringSchedule: table.wateringSchedule });
  } catch (error) {
    console.error('Error al actualizar el plan de riego:', error);
    res.status(500).json({ message: 'Error al actualizar el plan de riego', error });
  }
};
```

Figura 47: Codi del mètode updateWateringSchedule al controlador.

- **verifyTableId:** Comprova si un `tableId` especificat és vàlid i pertany a l’usuari autenticat. Retorna un missatge de confirmació si la taula existeix i està associada correctament.

```
const verifyTableId = async (req, res) => {
  const { tableId } = req.params; // El tableId se obtiene de los parámetros de la URL
  const userId = req.userId; // Extraemos el userId del token JWT proporcionado

  try {
    // Buscar la mesa con el tableId y que además esté asociada al usuario
    const table = await Table.findOne({ tableId, users: userId });

    // Si la mesa no se encuentra o no está asociada al usuario, devolver un error 404
    if (!table) {
      return res.status(404).json({ message: 'Table ID no encontrado o no pertenece al usuario' });
    }

    // Si la mesa existe y está asociada al usuario, devolver un mensaje de éxito
    res.status(200).json({ message: 'Table ID válido' });
  } catch (error) {
    console.error('Error verificando table ID:', error);
    res.status(500).json({ message: 'Error verificando table ID', error });
  }
};
```

Figura 48: Codi del mètode verifyTableId al controlador.

- **fillTable:** Aquest mètode envia una ordre mitjançant MQTT per omplir una taula. Abans de fer-ho, verifica que l'estat actual de la taula permet aquesta acció (no ha d'estar plena ni omplint-se).

```
const fillTable = async (req, res) => {
  const { tableId } = req.params;
  const userId = req.userId;

  try {
    const table = await Table.findOne({ tableId, users: userId });
    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada o el usuario no tiene acceso' });
    }

    if (table.state === 'filling' || table.state === 'full' || table.state === 'draining') {
      return res.status(400).json({ message: 'La mesa ya está llenándose o está llena' });
    }

    // Publicar comando MQTT
    sendFillCommand(tableId);

    res.status(200).json({ message: 'Comando de llenado enviado a la mesa' });
  } catch (error) {
    console.error('Error al llenar la mesa:', error);
    res.status(500).json({ message: 'Error al llenar la mesa', error: error.message || error });
  }
};
```

Figura 49: Codi del mètode fillTable al controlador.

- **drainTable:** Similar al mètode anterior, però per buidar la taula. Envia

una ordre MQTT després de comprovar que l'estat de la taula no és buit ni en procés de buidatge.

```
const drainTable = async (req, res) => {
  const { tableId } = req.params;
  const userid = req.userId;

  try {
    const table = await Table.findOne({ tableId, users: userid });
    if (!table) {
      return res.status(404).json({ message: 'Mesa no encontrada o el usuario no tiene acceso' });
    }

    if (table.state === 'draining' || table.state === 'empty' || table.state === 'filling') {
      return res.status(400).json({ message: 'La mesa ya está vaciándose o está vacía' });
    }

    // Publicar comando MQTT
    sendDrainCommand(tableId);

    res.status(200).json({ message: 'Comando de vaciado enviado a la mesa' });
  } catch (error) {
    console.error('Error al vaciar la mesa:', error);
    res.status(500).json({ message: 'Error al vaciar la mesa', error: error.message || error });
  }
};
```

Figura 50: Codi del mètode `drainTable` al controlador.

- **getUserTablesStats:** Genera estadístiques detallades per a cadascuna de les taules associades a l’usuari autenticat. Inclou informació com el nombre de vegades que s’han omplít i buidat, les dates d’aquestes accions i l’última data de registre de dades dels sensors.

```
const getUserTablesStats = async (req, res) => {
  try {
    const userId = req.userId; // ID del usuario autenticado
    console.log(`User ID: ${userId}`); // Verificar el ID del usuario

    // Buscar todas las mesas del usuario
    const tables = await Table.find({ users: userId });
    console.log(`Tables Found: ${tables.length}`); // Mostrar las mesas encontradas

    if (tables.length === 0) {
      console.log(`No tables found for user`);
      return res.status(404).json({ message: 'No hay mesas asociadas a este usuario' });
    }

    // Generar estadísticas para cada mesa
    const tablesStats = await Promise.all(
      tables.map(async (table) => {
        try {
          console.log(`Processing tableId: ${table.tableId}`);

          // Obtener la última lectura de sensores para la mesa
          const lastSensorData = await SensorData.findOne({ tableId: table.tableId })
            .sort({ date: -1 })
            .select('date')
            .lean();

          console.log(`SensorData for tableId: ${table.tableId}, ${lastSensorData.date}`);
        } catch (err) {
          console.error(`Error processing table: ${table.tableId}, ${err}`);
          throw err;
        }
      })
    );

    res.status(200).json(tablesStats);
  } catch (error) {
    console.error('Error al obtener estadísticas de las mesas:', error);
    res.status(500).json({ message: 'Error al obtener estadísticas de las mesas' });
  }
};
```

Figura 51: Codi del mètode `getUserTablesStats` al controlador.

- **UserController.js:** Responsable de la gestió d’usuaris, com el registre, login, actualització de perfil i gestió de contrasenyes. Proporciona una capa de seguretat mitjançant autenticació basada en tokens JWT.
 - **registerUser:** Aquest mètode registra un nou usuari al sistema. Verifica que el nom d’usuari, el correu electrònic o el número de telèfon no estiguin ja registrats. Si tot és correcte, encripta la contrasenya de l’usuari i crea un

nou registre a la base de dades amb informació com el nom d'usuari, correu electrònic, telèfon i una imatge de perfil predeterminada.

```
const registerUser = async (req, res) => {
  try {
    const { username, password, email, phone, country, province, companyName } = req.body;

    // Verificar si el nombre de usuario, el correo electrónico o el número de teléfono ya existen
    const existingUser = await User.findOne({ $or: [{ username }, { email }, { phone }] });
    if (existingUser) {
      if (existingUser.username === username) {
        return res.status(400).json({ message: 'El nombre de usuario ya existe' });
      } else if (existingUser.email === email) {
        return res.status(400).json({ message: 'El correo electrónico ya está registrado' });
      } else if (existingUser.phone === phone) {
        return res.status(400).json({ message: 'El número de teléfono ya está registrado' });
      }
    }

    // Crear nuevo usuario
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({
      username,
      password: hashedPassword,
      email,
      phone,
      country,
      province,
      companyName,
      profileImage: '/uploads/profileImage.jpg' // Ruta de la imagen por defecto
    });

    await user.save();
    const token = jwt.sign({ userId: user._id }, 'your_jwt_secret', { expiresIn: '1h' });

    // Devolver token y usuario registrado
    res.status(201).json({ token, username: user.username });
  } catch (error) {
    res.status(500).json({ message: 'Error en el servidor', error });
  }
};
```

Figura 52: Codi del mètode `registerUser` al controlador.

- **loginUser:** S'encarrega de l'autenticació d'un usuari existent. Comprova si el nom d'usuari i la contrasenya proporcionats coincideixen amb els registres de la base de dades. Si és així, genera un token JWT que permet la identificació de l'usuari durant la sessió.

```
const loginUser = async (req, res) => {
  try {
    const { username, password } = req.body;
    console.log('Received username:', username);
    console.log('Received password:', password);

    const user = await User.findOne({ username });
    if (!user) {
      console.log('User not found');
      return res.status(400).json({ message: 'Invalid username or password' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      console.log('Password does not match');
      return res.status(400).json({ message: 'Invalid username or password' });
    }

    const token = jwt.sign({ userId: user.id }, 'your_jwt_secret', { expiresIn: '1h' });
    console.log('Login successful, sending token and userId');
    res.status(200).json({ token, userId: user.id });

  } catch (error) {
    console.error('Error logging in:', error);
    res.status(500).json({ message: 'Error logging in', error });
  }
};
```

Figura 53: Codi del mètode `loginUser` al controlador.

- **updatePassword:** Permet que un usuari canvia la seva contrasenya. Comprova que l'usuari estiga autenticat i que la contrasenya actual proporcionada coincideixi amb la guardada. Si tot és correcte, guarda la nova contrasenya després d'encriptar-la.

```

const updatePassword = async (req, res) => {
  try {
    const { currentPassword, newPassword } = req.body;
    const userId = req.userId;

    const user = await User.findById(userId);

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    const isMatch = await bcrypt.compare(currentPassword, user.password);

    if (!isMatch) {
      return res.status(400).json({ message: 'Current password is incorrect' });
    }

    const hashedPassword = await bcrypt.hash(newPassword, 10);

    user.password = hashedPassword;
    await user.save();

    res.status(200).json({ message: 'Password updated successfully' });
  } catch (error) {
    res.status(400).json({ message: 'Error updating password', error });
  }
};

```

Figura 54: Codi del mètode `updatePassword` al controlador.

- **deleteUser:** Elimina un usuari del sistema, incloent totes les seves taules associades i les dades de sensors vinculades a aquestes. Aquest mètode assegura que tota la informació relacionada amb l'usuari sigui eliminada de forma segura.

```

const deleteUser = async (req, res) => {
  try {
    const userId = req.userId;

    // Obtener las mesas asociadas al usuario
    const tables = await Table.find({ users: userId });

    // Eliminar los datos de sensores asociados a cada mesa del usuario
    for (let table of tables) {
      await SensorData.deleteMany({ tableId: table.tableId });
    }

    // Eliminar todas las mesas asociadas al usuario
    await Table.deleteMany({ users: userId });

    // Ahora eliminar el usuario
    const user = await User.findByIdAndDelete(userId);

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    res.status(200).json({ message: 'User, associated tables, and sensor data deleted successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Error deleting user, tables, and sensor data', error });
  }
};

```

Figura 55: Codi del mètode `deleteUser` al controlador.

- **getWeather:** Obtén informació meteorològica basada en la ubicació (país i província) d'un usuari autenticat. Aquestes dades són obtingudes a través del servei d'API climàtica i enviades al frontend.

```

const getWeather = async (req, res) => {
  try {
    const userId = req.userId;

    // Busca al usuari en la base de dades
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'Usuari no encontrado' });
    }

    const { country, province } = user;

    // Obten els dades del clima usando el servici
    const weatherData = await getWeatherByLocation(province, country);

    // Envia els dades al frontend
    res.status(200).json(weatherData);
  } catch (error) {
    console.error('Error al obtener el clima:', error);
    res.status(500).json({ message: 'Error al obtener el clima', error });
  }
};

```

Figura 56: Codi del mètode `getWeather` al controlador.

- **updateEmail:** Actualitza el correu electrònic d'un usuari autenticat. Verifica que el nou correu no estiga ja en ús per un altre usuari abans de guardar el canvi.

```

const updateEmail = async (req, res) => {
  try {
    const { email } = req.body;
    const userId = req.userId;

    // Verificar si el nou correu electrònic ya està en ús
    const existingUser = await User.findOne({ email });
    if (existingUser && existingUser._id.toString() !== userId) {
      return res.status(400).json({ message: 'El correo electrónico ya está en uso' });
    }

    // Actualitzar el correu electrònic si no està en ús
    const updatedUser = await User.findByIdAndUpdate(userId, { email }, { new: true });

    res.status(200).json({ message: 'Correo electrónico actualizado correctamente', user: updatedUser });
  } catch (error) {
    res.status(400).json({ message: 'Error al actualizar el correo electrónico', error });
  }
};

```

Figura 57: Codi del mètode `updateEmail` al controlador.

- **forgotPassword:** Genera un token per a la recuperació de contrasenya i l'envia al correu electrònic de l'usuari. Aquest token és temporal i expira després d'una hora.

```

const forgotPassword = async (req, res) => {
  try {
    const { email } = req.body;
    const user = await User.findOne({ email });

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    const resetToken = crypto.randomBytes(20).toString('hex');
    user.resetPasswordToken = resetToken;
    user.resetPasswordExpires = Date.now() + 3600000; // 1 hour
    await user.save();

    sendResetPasswordEmail(user, resetToken);

    res.status(200).json({ message: 'Reset password email sent' });
  } catch (error) {
    res.status(500).json({ message: 'Error sending reset password email', error });
  }
};

```

Figura 58: Codi del mètode `forgotPassword` al controlador.

- **resetPassword:** Restableix la contrasenya d'un usuari basant-se en un token de recuperació. Verifica que el token sigui vàlid i no hagi expirat abans de guardar la nova contrasenya.

```

const resetPassword = async (req, res) => {
  try {
    const { token, newPassword } = req.body;
    const user = await User.findOne({
      resetPasswordToken: token,
      resetPasswordExpires: { $gt: Date.now() }
    });

    if (!user) {
      return res.status(400).json({ message: 'Invalid or expired token' });
    }

    const hashedPassword = await bcrypt.hash(newPassword, 10);
    user.password = hashedPassword;
    user.resetPasswordToken = undefined;
    user.resetPasswordExpires = undefined;
    await user.save();

    res.status(200).json({ message: 'Password reset successfully' });
  } catch (error) {
    res.status(500).json({ message: 'Error resetting password', error });
  }
};

```

Figura 59: Codi del mètode `resetPassword` al controlador.

- **updateUsername:** Permet que un usuari canvia el seu nom d’usuari. Verifica que el nou nom no estigaa ja en ús per un altre usuari abans de realitzar l’actualització.

```

const updateUsername = async (req, res) => {
  try {
    const { username } = req.body;
    const userId = req.userId;

    // Verificar si el nuevo nombre de usuario ya está en uso
    const existingUser = await User.findOne({ username });
    if (existingUser && existingUser.id.toString() !== userId) {
      return res.status(400).json({ message: 'El nombre de usuario ya está siendo usado' });
    }

    // Actualizar el nombre de usuario si no está en uso
    const updatedUser = await User.findByIdAndUpdate(userId, { username }, { new: true });

    res.status(200).json({ message: 'Nombre de usuario actualizado correctamente', user: updatedUser });
  } catch (error) {
    res.status(400).json({ message: 'Error al actualizar el nombre de usuario', error });
  }
};

```

Figura 60: Codi del mètode `updateUsername` al controlador.

- **validateToken:** Valida el token JWT d’un usuari per assegurar que és vàlid i no ha expirat. Retorna informació de l’usuari sense incloure la contrasenya.

```

const validateToken = async (req, res) => {
  try {
    const userId = req.userId;
    const user = await User.findById(userId).select('-password');

    if (!user) {
      console.log('User not found for token validation'); // Mensaje de depuración
      return res.status(401).json({ message: 'Invalid token' });
    }

    console.log('User found for token validation:', user); // Mensaje de depuración
    res.status(200).json(user);
  } catch (error) {
    console.error('Error validating token:', error); // Mensaje de depuración
    res.status(500).json({ message: 'Error validating token', error });
  }
};

```

Figura 61: Codi del mètode `validateToken` al controlador.

- **getProfile:** Retorna la informació del perfil d’un usuari autenticat, incloent dades com el nom d’usuari, correu electrònic, telèfon, país, província i nom de l’empresa.

```

const getProfile = async (req, res) => {
  try {
    // Obtener el ID del usuario autenticado a partir del token
    const userId = req.userId;

    // Buscar al usuario en la base de datos usando su ID
    const user = await User.findById(userId);

    // Verificar si el usuario existe
    if (!user) {
      return res.status(404).json({ message: 'Usuario no encontrado' });
    }

    // Enviar los datos del perfil del usuario al frontend
    res.status(200).json({
      username: user.username,
      email: user.email,
      phone: user.phone,
      country: user.country,
      province: user.province,
      companyName: user.companyName,
      profileImage: user.profileImage
    });
  } catch (error) {
    // Manejo de errores
    res.status(500).json({ message: 'Error al obtener el perfil del usuario', error });
  }
};

```

Figura 62: Codi del mètode `getProfile` al controlador.

- **uploadProfileImage:** Actualitza la imatge de perfil d'un usuari. Guarda la nova imatge a la base de dades i substitueix la ruta de la imatge anterior.

```

const uploadProfileImage = async (req, res) => {
  try {
    const userId = req.userId;
    const user = await User.findById(userId);

    if (!user) {
      return res.status(404).json({ message: 'Usuario no encontrado' });
    }

    const uploadedFilePath = `src/uploads/${req.file.filename}`;
    console.log("Ruta del archivo subido:", uploadedFilePath);

    // Guardar la nueva imagen
    user.profileImage = `/uploads/${req.file.filename}`;
    await user.save();

    res.status(200).json({ profileImage: user.profileImage });
  } catch (error) {
    console.error("Error al subir la imagen:", error);
    res.status(500).json({ message: 'Error al subir la imagen', error });
  }
};

```

Figura 63: Codi del mètode `uploadProfileImage` al controlador.

- **updatePhone:** Permet que un usuari canvie el seu número de telèfon. Comprova que el nou número no estiga ja en ús per un altre usuari abans de guardar l'actualització.

```

const updatePhone = async (req, res) => {
  try {
    const { phone } = req.body;
    const userId = req.userId;

    // Verificar si el nuevo número de teléfono ya está en uso
    const existingUser = await User.findOne({ phone });
    if (existingUser && existingUser._id.toString() !== userId) {
      return res.status(400).json({ message: 'El número de teléfono ya está en uso' });
    }

    // Actualizar el número de teléfono si no está en uso
    const updatedUser = await User.findByIdAndUpdate(userId, { phone }, { new: true });

    res.status(200).json({ message: 'Número de teléfono actualizado correctamente', user: updatedUser });
  } catch (error) {
    res.status(400).json({ message: 'Error al actualizar el número de teléfono', error });
  }
};

```

Figura 64: Codi del mètode `updatePhone` al controlador.

- **openaiController.js:** Responsable del funcionament per a acridar a la API de OpenAI.

- **handleTextPrompt:** Permet guardar el prompt proporcionat per el usuari i enviar-lo al servei que tenim configurat en la carpeta Services amb la finalitat de que aquest envie el prompt a la API de OpenAI per a obtenir una resposta.

```
const sendPromptToOpenAI = async (prompt) => {
  try {
    const response = await axios.post(
      'https://api.openai.com/v1/chat/completions',
      {
        model: 'gpt-3.5-turbo',
        messages: [{ role: 'user', content: prompt }],
        temperature: 0.2,
        max_tokens: 150,
      },
      {
        headers: {
          Authorization: `Bearer ${apikey}`,
          'Content-Type': 'application/json',
        },
      }
    );
    return response.data;
  } catch (error) {
    if (error.response) {
      console.error(`Error respuesta OpenAI:`, error.response.status, error.response.data);
    } else if (error.request) {
      console.error(`Error en la solicitud:`, error.request);
    } else {
      console.error(`Error en la configuración:`, error.message);
    }
    throw new Error('No se pudo conectar con OpenAI.');
  }
};
```

Figura 65: Codi del mètode handleTextPrompt al controlador.

Rutes API (backend/src/routes/) Les rutes del sistema connecten el frontend amb el backend, proporcionant accés a les funcionalitats principals. A continuació es detallen les rutes agrupades per funcionalitat:

Rutes d'Usuari (*userRoutes.js*) Aquest grup de rutes maneja operacions relacionades amb el controlador d'usuaris *userController.js*, el qual defineix la lògica de cada una de les operacions detallades a continuació. Aquestes rutes estan protegides amb l'*authMiddleware* per verificar l'autenticació de l'usuari en aquelles accions que requereixen permisos [Document de les rutes](#).

- **POST /register:** Registra un nou usuari al sistema. Aquesta ruta és utilitzada per crear un nou compte, enllaçant amb la funció *registerUser* del controlador d'usuaris.
- **POST /login:** Permet als usuaris iniciar sessió al sistema, vinculant-se amb *loginUser* per autenticar i generar un token.
- **PUT /update-username:** Actualitza el nom d'usuari d'un compte existent. La ruta està protegida i utilitza *authMiddleware* per verificar la identitat.
- **PUT /update-email:** Modifica l'adreça de correu electrònic de l'usuari autenticat. Enllaça amb *updateEmail*.
- **PUT /update-password:** Permet a l'usuari canviar la seva contrasenya, validant primer la contrasenya actual amb *updatePassword*.
- **DELETE /delete:** Elimina completament el compte de l'usuari, incloent totes les dades associades (com les taules i els sensors). Implementat amb *deleteUser*.

- **POST /forgot-password:** Inicia el procés de recuperació de contrasenya, enviant un correu amb un enllaç temporal. Associada a *forgotPassword*.
- **POST /reset-password:** Restaura la contrasenya utilitzant un token temporal. Enllaça amb *resetPassword*.
- **GET /weather:** Recupera informació del clima basada en la ubicació de l'usuari (província i país). Relacionada amb *getWeather*.
- **GET /validate-token:** Valida el token d'autenticació proporcionat per l'usuari per assegurar-se que és vàlid. Enllaça amb *validateToken*.
- **GET /profile:** Recupera les dades del perfil de l'usuari autenticat, incloent el nom, correu, telèfon, i imatge de perfil. Implementat amb *getProfile*.
- **POST /upload-profile-image:** Permet pujar una nova imatge de perfil per a l'usuari autenticat. Utilitza *multerMiddleware* per gestionar la càrrega del fitxer i *uploadProfileImage* per processar-ho.
- **PUT /update-phone:** Actualitza el número de telèfon associat al compte de l'usuari. Vinculat a *updatePhone*.

```
// src/routes/userRoutes.js

//Define las rutas relacionadas con los usuarios.
const express = require('express');
const { registerUser, loginUser,
        updatePassword, deleteUser, getWeather,
        updateEmail, forgotPassword, resetPassword,
        validateToken, updateUsername, uploadProfileImage,
        getProfile, updatePhone } = require('../controllers/userController');
const authMiddleware = require('../middlewares/authMiddleware'); // Importa authMiddleware
const upload = require('../middlewares/multerMiddleware'); // Importar el middleware de multer

//router: Es una instancia de express.Router que define rutas específicas para la aplicación.
const router = express.Router();

router.post('/register', registerUser); //Registra un nuevo usuario
router.post('/login', loginUser); //Hacemos login de el usuario registrado previamente
router.put('/update-username', authMiddleware, updateUsername);
router.put('/update-email', authMiddleware, updateEmail);
router.put('/update-password', authMiddleware, updatePassword); //Actualizamos la contraseña del usuario
router.delete('/delete', authMiddleware, deleteUser); // Nueva ruta para eliminar usuario
router.post('/forget-password', forgotPassword); // Nueva ruta para solicitar recuperación de contraseña
router.post('/reset-password', resetPassword); // Nueva ruta para restablecer contraseña
router.get('/weather', authMiddleware, getWeather) // Esta ruta sirve para la temperatura de la zona del usuario
router.get('/validate-token', authMiddleware, validateToken); // Nueva ruta para validar el token
router.get('/profile', authMiddleware, getProfile); // Ruta protegida para obtener el perfil del usuario autenticado
router.post('/upload-profile-image', authMiddleware, upload.single('profileImage'), uploadProfileImage); // Ruta para hacer el cambio de imagen
router.put('/update-phone', authMiddleware, updatePhone); // Ruta para actualizar el número de teléfono

module.exports = router;
```

Figura 66: Imatge del codi de userRoutes.js.

Rutes de Taula (*tableRoutes.js*) Aquest grup de rutes maneja operacions relacionades amb el controlador de taules [TableController.js](#), el qual defineix la lògica de cada una de les accions detallades a continuació. Aquestes rutes estan protegides amb l'*authMiddleware*, assegurant que només els usuaris autenticats puguin interactuar amb les dades de les taules [Document de les rutes](#).

- **POST /create-table:** Crea una nova taula associada a l'usuari autenticat. Aquesta operació és protegida i utilitza *authMiddleware*. Està vinculada a la funció *createTable*.

- **DELETE /delete-table/:tableId:** Elimina una taula especificada pel seu identificador. Inclou la eliminació de dades associades, com els sensors, i està protegida. Associada a *deleteTable*.
- **GET /user-tables:** Obté totes les taules associades a l'usuari autenticat. Està vinculada a la funció *getUserTables*.
- **GET /:tableId:** Recupera els detalls d'una taula específica mitjançant el seu *tableId*. Relacionada amb *getTableDetails*.
- **PUT /:tableId/watering-schedule:** Actualitza la planificació del reg automàtic per a una taula específica. Utilitza la funció *updateWateringSchedule*.
- **PUT /:tableId/update-name:** Actualitza el nom d'una taula específica. Està associada a *updateTableName*.
- **PUT /:tableId/update-description:** Modifica la descripció d'una taula. Relacionada amb *updateTableDescription*.
- **GET /verify-table/:tableId:** Verifica l'existència d'una taula mitjançant el seu *tableId*. Està vinculada a *verifyTableId*.
- **POST /:tableId/upload-image:** Permet pujar una imatge per a una taula específica. Utilitza *multerMiddleware* per gestionar la càrrega i la funció *uploadTableImage* per processar-la.
- **POST /save-sensor-data:** Guarda les dades dels sensors associats a una taula. Està protegida i utilitza *saveSensorData*.
- **GET /sensor-data/:tableId:** Obté les dades històriques dels sensors per a una taula específica. Associada a *getSensorData*.
- **PUT /:tableId/fill:** Envia una ordre per omplir una taula mitjançant el controlador *fillTable*.
- **PUT /:tableId/drain:** Envia una ordre per buidar una taula. Està relacionada amb *drainTable*.
- **GET /user-tables-stats:** Recupera estadístiques generals sobre les taules d'un usuari. Està vinculada a *getUserTablesStats*.

```

const express = require('express');
const { createTable, deleteTable, getUserTables,
    getTableDetails, updateWateringSchedule, updateTableName,
    updateTableDescription, verifyTableId, saveSensorData,
    getSensorData, fillTable, drainTable,
    uploadTableImage, getUserTablesstats } = require('../controllers/TableController');
const authMiddleware = require('../middlewares/authMiddleware'); // Para proteger las rutas
const upload = require('../middlewares/multerMiddleware');
const router = express.Router();

// Ruta para crear una nueva mesa (protegida, solo usuarios logueados)
router.post('/create-table', authMiddleware, createTable);
// Ruta para eliminar una mesa (protegida, solo usuarios logueados)
router.delete('/delete-table/:tableId', authMiddleware, deleteTable);
// Ruta para obtener todas las mesas del usuario
router.get('/user-tables', authMiddleware, getUserTables);
// Ruta para obtener los detalles de la mesa
router.get('/:tableId', authMiddleware, getTableDetails);
// Nueva ruta para actualizar la planificación de riego
router.put('/:tableId/watering-schedule', authMiddleware, updateWateringSchedule);
// Actualizar el nombre de la mesa
router.put('/:tableId/update-name', authMiddleware, updateTableName);
// Actualizar la descripción de la mesa
router.put('/:tableId/update-description', authMiddleware, updateTableDescription);
// Ruta para verificar si el tableId existe
router.get('/verify-table/:tableId', authMiddleware, verifyTableId); // Nueva ruta para la verificación
router.post('/:tableId/upload-image', authMiddleware, upload.single('tableImage'), uploadTableImage); // Nueva ruta para subir imagen
// Ruta para guardar los datos de los sensores
router.post('/save-sensor-data', authMiddleware, saveSensorData); // Protegida
// Ruta para obtener los datos históricos de sensores de una mesa
router.get('/sensor-data/:tableId', authMiddleware, getSensorData); // Protegida
// Ruta para llenar la mesa
router.put('/:tableId/fill', authMiddleware, fillTable);
// Ruta para vaciar la mesa
router.put('/:tableId/drain', authMiddleware, drainTable);
// Nueva ruta para obtener estadísticas
router.get('/user-tables-stats', authMiddleware, getUserTablesstats);

module.exports = router;

```

Figura 67: Imatge del codi de *tableRoutes.js*.

Rutes d'OpenAI (*openaiRoutes.js*) Aquest grup de rutes maneja operacions relacionades amb el controlador d'OpenAI [openaiController.js](#), el qual defineix la lògica per enviar sol·licituds a l'API d'OpenAI i processar respuestes. Aquestes rutes permeten una integració senzilla i eficient amb els serveis d'intel·ligència artificial proporcionats per OpenAI [Document de les rutes](#).

- **POST /send-text:** Permet enviar un text com a entrada a l'API d'OpenAI, processant el resultat mitjançant la funció *handleTextPrompt* al controlador. Aquesta ruta facilita la interacció amb models de llenguatge avançats com *GPT-3.5-turbo*.

```

const express = require('express');
const { handleTextPrompt } = require('../controllers/openaiController');

const router = express.Router();
// Rutas
router.post('/send-text', handleTextPrompt); // Enviar texto

module.exports = router;

```

Figura 68: Imatge del codi de *openaiRoutes.js*.

Rutes al Fitxer *index.js* (backend/src/index.js) El fitxer *index.js* actua com a punt d'entrada principal per al servidor. La seva funció principal és registrar totes les rutes globals del sistema, inicialitzar els serveis necessaris i connectar la base de dades. A continuació es detallen les rutes globals i les seves connexions amb els fitxers de rutes corresponents:

- **/api/users:** Conecta amb les rutes definides a [userRoutes.js](#). Aquestes rutes gestionen operacions relacionades amb usuaris, com registre, inici de sessió, actualització del perfil, entre altres.
- **/api/tables:** Conecta amb les rutes definides a [tableRoutes.js](#). S'encarrega de gestionar operacions sobre les taules, com crear, actualitzar, consultar estadístiques i operar amb les vàlvules.
- **/api/openai:** Conecta amb les rutes definides a [openaiRoutes.js](#). Aquestes rutes permeten la comunicació amb l'API d'OpenAI per generar respostes a sol·licituds d'intel·ligència artificial.
- **/uploads:** Serveix arxius estàtics, com imatges pujades per l'usuari, accessibles a través de *multerMiddleware*.
- **Catch-All Route:** Gestiona qualsevol altra sol·licitud que no coincideixi amb les rutes anteriors, enviant l'aplicació React al client. Això és especialment útil per a aplicacions d'una sola pàgina (SPA) creades amb React.

El fitxer [index.js](#) també inclou la inicialització de serveis importants com:

- **WebSocketService:** Gestiona la comunicació en temps real entre el backend i els clients.
- **MQTTService:** Permet la comunicació amb el hardware a través del protocol MQTT.
- **WateringScheduler:** Programa el reg automàtic mitjançant cron jobs.

```

require('dotenv').config();
const express = require('express');
const cors = require('cors');
const path = require('path');
const connectDB = require('./database/db'); // Importamos la conexión a la base de datos
const userRoutes = require('./routes/userRoutes'); // Importamos las rutas de usuario
const tableRoutes = require('./routes/tableRoutes'); // Importamos las rutas de mesas
const openaiRoutes = require('./routes/openaiRoutes.js'); // Importamos las rutas de mesas
const mqttService = require('./services/mqttService'); // Inicializamos el servicio MQTT
require('./services/webSocketService'); // Servidor WebSocket
require('./services/wateringScheduler'); // Iniciar el cron job

const app = express(); // Inicializamos la aplicación Express
// Middleware para permitir CORS (Permitir solicitudes desde cualquier origen)
app.use(cors());
// Middleware para parsear el cuerpo de las solicitudes HTTP en formato JSON
app.use(express.json());
connectDB();

// **Mover esta linea hacia arriba antes de manejar las rutas del frontend**
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));

// Definir rutas de la API
app.use('/api/users', userRoutes); // Rutas para el manejo de usuarios
app.use('/api/tables', tableRoutes); // Añadir las rutas para las mesas
app.use('/api/openai', openaiRoutes);

// Sirve archivos estáticos de la aplicación React en producción
app.use(express.static(path.join(__dirname, 'frontend/build')));

// Ruta catch-all para manejar rutas del frontend (SPA con React)
app.get('*', (req, res) => {
| res.sendFile(path.join(__dirname, 'frontend/build', 'index.html'));
});

// Puerto donde el servidor escuchará
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
| console.log(`Server is running on port ${PORT}`);
});

```

Figura 69: Definició de les rutes globals i inicialització de serveis al fitxer index.js.

Services (backend/src/services/) Els serveis contenen la lògica auxiliar per a tasques específiques, a continuació es detallen tots.

- **mqttService.js:** Aquest servei gestiona la comunicació entre el backend i el hardware mitjançant el protocol MQTT. Inclou la configuració per connectar-se a un broker MQTT, la lògica per subscriure's a temes rellevants i el processament de missatges MQTT rebuts. També permet enviar comandes de control per al compliment i el buidatge de les taules.
- **webSocketService.js:** Gestiona la comunicació en temps real entre el backend i el frontend mitjançant WebSockets. Permet als clients rebre actualitzacions en temps real sobre l'estat i les estadístiques. També inclou la lògica per gestionar les connexions dels clients i enviar dades dels sensors al frontend.
- **emailService.js:** Aquest servei s'encarrega de l'enviament de correus electrònics utilitzant la llibreria *nodemailer*. Proporciona funcionalitats per enviar correus de recuperació de contrasenya amb enllaços temporals, assegurant que els usuaris puguen restablir les seves credencials de manera segura.
- **openaiService.js:** Gestiona les sol·licituds a l'API de OpenAI per generar respostes intel·ligents basades en *prompts*. Aquest servei utilitza *axios* per fer peticions HTTP i proporciona un mecanisme robust per interactuar amb els models de OpenAI.

- **wateringScheduler.js:** Aquest servei utilitza *cron jobs* per programar tasques automatitzades relacionades amb el reg de les taules. Verifica les hores i els dies definits a la planificació de reg de cada taula i executa automàticament les accions de plenar o buidar segons siga necessari.
- **weatherService.js:** Aquest servei proporciona dades meteorològiques basades en la ubicació de l'usuari. Utilitza l'API de WeatherAPI per obtenir informació com la temperatura, la humitat i la velocitat del vent. És especialment útil per adaptar les operacions del sistema a les condicions climàtiques actuals.

Models (backend/src/models/) Els models defineixen els esquemes de dades per a la base de dades MongoDB. A continuació es descriuen els 3 esquemes que tenim al projecte:

- **User.js:** Aquest model defineix l'estructura dels documents de la col·lecció *users* en MongoDB. Inclou camps com el nom d'usuari (*username*), contrasenya (*password*), correu electrònic (*email*), número de telèfon (*phone*), país (*country*), província (*province*), i nom de l'empresa (*companyName*).

També afegeix funcionalitats per gestionar la recuperació de contrasenyes mitjançant *resetPasswordToken* i *resetPasswordExpires*. Per defecte, assigna una imatge de perfil a cada usuari i inclou marques de temps (*timestamps*) per gestionar la creació i actualització dels documents.

```
// src/models/User.js
const mongoose = require('mongoose');

// userSchema: Define la estructura de los documentos de usuario en la colección users, incluyendo campos y sus tipos de datos.
const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  phone: { type: String, required: true },
  country: { type: String, required: true },
  province: { type: String, required: true },
  companyName: { type: String, required: true },
  // Estas dos variables sirven para poder hacer el cambio de contraseña.
  resetPasswordToken: { type: String },
  resetPasswordExpires: { type: Date },
  // Nuevo campo para la imagen de perfil
  profileImage: { type: String, default: '/uploads/profileImage.jpg' }
}, {
  timestamps: true,
});

// User: Es el modelo que se usará para interactuar con la colección users en MongoDB.
const User = mongoose.model('User', userSchema);

module.exports = User;
```

Figura 70: Codi del document *User.js*.

- **Table.js:** Aquest model representa les taules de reg i defineix camps per emmagatzemar informació com l'identificador únic de la taula (*tableId*), nom (*name*), descripció (*description*) i usuaris amb accés (*users*). També defineix un esquema per a la planificació del reg (*wateringSchedule*), que inclou dies, hores de plenatge i hores de buidatge. A més, gestiona estadístiques com el nombre de plenats (*fillCount*), buidatges (*drainCount*), i les dates de les últimes accions (*lastFilledAt* i *lastDrainedAt*). Aquest model facilita el seguiment i el control de l'estat actual de la taula (*state*).

```

const mongoose = require('mongoose');

// Esquema para la planificación del riego automático
const wateringScheduleSchema = new mongoose.Schema({
  days: [{ type: String, required: true }], // Días de riego (Ej: ["Lunes", "Miércoles"])
  fillTime: { type: String, required: true }, // Hora de llenado en formato HH:mm
  drainTime: { type: String, required: true }, // Hora de vaciado en formato HH:mm
  updatedAt: { type: Date, default: Date.now }, // Fecha de la última actualización
});

// Definimos los estados de la mesa
const tableStateEnum = ['empty', 'filling', 'full', 'draining']; // Estados posibles

// Esquema principal de la mesa
const tableSchema = new mongoose.Schema({
  tableId: { type: String, required: true, unique: true }, // Identificador único de la mesa (2 cifras numéricas)
  name: { type: String, required: true }, // Nombre descriptivo de la mesa (ej: "Invernadero", "Exterior")
  description: { type: String }, // Descripción de la mesa (opcional)
  users: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }], // Usuarios que tienen acceso a esta mesa
  wateringSchedule: wateringScheduleSchema, // Planificación del riego automático
  tableImage: { type: String, default: '/uploads/defaultTableImage.jpg' }, // Imagen de la mesa con valor por defecto
  state: { type: String, enum: tableStateEnum, default: 'empty' }, // Estado de la mesa
  fillCount: { type: Number, default: 0 }, // Total de llenados de la mesa
  drainCount: { type: Number, default: 0 }, // Total de vaciados de la mesa
  lastFilledAt: { type: Date, default: null }, // Fecha de la última vez que se llenó la mesa
  lastDrainedAt: { type: Date, default: null }, // Fecha de la última vez que se vació la mesa
  createdAt: { type: Date, default: Date.now }, // Fecha de creación de la mesa
});

const Table = mongoose.model('Table', tableSchema);

module.exports = Table;

```

Figura 71: Codi del document Table.js.

- **SensorData.js:** Aquest model defineix l'esquema per als registres de dades dels sensors associats a cada taula. Els camps inclouen *tableId* (per identificar la taula), *date* (per registrar la data de les lectures) i les mesures dels sensors, com la temperatura (*temperature*), la humitat (*humidity*) i el nivell d'aigua (*waterLevel*). Aquest model permet emmagatzemar i consultar dades històriques dels sensors per a anàlisis posteriors i monitorització.

```

const mongoose = require('mongoose');

// Esquema para los datos de los sensores
const sensorDataSchema = new mongoose.Schema({
  tableId: { type: String, required: true }, // Referencia a la mesa
  date: { type: Date, default: Date.now }, // Fecha en la que se tomaron las lecturas
  temperature: { type: Number, required: true }, // Temperatura del agua en grados Celsius
  humidity: { type: Number, required: true }, // Humedad del suelo en porcentaje
  waterLevel: { type: Number, required: true }, // Porcentaje de nivel de agua
});

const SensorData = mongoose.model('SensorData', sensorDataSchema);

module.exports = SensorData;

```

Figura 72: Codi del document SensorData.js.

Base de Dades El sistema utilitza **MongoDB**, una base de dades NoSQL orientada a documents, per gestionar de manera eficient les dades del sistema. La connexió amb la base de dades es realitza mitjançant el fitxer *db.js*, que inicialitza i configura la connexió amb el servidor de MongoDB. Aquesta connexió es manté activa durant l'execució del servidor i inclou mecanismes per gestionar errors de connexió de manera eficient.

- **users:** Aquesta col·lecció emmagatzema informació sobre els usuaris del sistema, incloent-hi nom d'usuari, correu electrònic, contrasenya, número de telèfon, ubicació, i configuració personal. També guarda metadades com l'historial de restabliment de contrasenya.

```
_id: ObjectId('674600938893c5aad88e514e')
username: "marc"
password: "$2a$10$YcbY/0OS.vMHkRBuk8UeP0JgWCy2GAPnj2jbASndD01N5.L0IG6a"
email: "marccp012@gmail.com"
phone: "+608659889"
country: "Spain"
province: "Valencia"
companyName: "Viveros Biosca"
profileImage: "/uploads/173298800818-5042442_full-descargar-fondos-pantalla-pc-fondo_"
createdAt: 2024-11-26T17:08:35.638+00:00
updatedAt: 2024-11-30T15:20:00.845+00:00
__v: 0
```

Figura 73: Usuari registrat en la base de dades.

- **tables:** La col·lecció *tables* conté informació sobre cada taula registrada al sistema. Això inclou identificadors únics (*tableId*), configuracions personalitzades, estat actual de la taula (com *empty*, *filling*, *full*), i dades sobre la planificació automàtica del reg (*wateringSchedule*). També guarda estadístiques com el nombre de vegades que la taula ha estat plena o buida.

```
_id: ObjectId('6748a8ead1e749d386e5168a')
tableId: "44"
name: "mesa nueva"
description: "ififiuefhuw"
users: Array (1)
  0: ObjectId('674600938893c5aad88e514e')
tableImage: "/uploads/1732823753172-logofinal.png"
state: "empty"
fillCount: 9
drainCount: 8
lastFilledAt: 2024-11-28T19:38:15.761+00:00
lastDrainedAt: 2024-11-28T19:56:16.819+00:00
createdAt: 2024-11-28T17:31:22.543+00:00
__v: 0
wateringSchedule: Object
  days: Array (empty)
  fillTime: "20:23"
  drainTime: "20:24"
  updatedAt: 2024-11-30T17:18:30.259+00:00
  _id: ObjectId('674b48e639a09bef5938a8dd')
```

Figura 74: Taula registrada en la base de dades.

- **sensorData:** Aquesta col·lecció guarda dades històriques dels sensors associats a cada taula. Inclou lectures de temperatura, humitat del sòl, i nivell d'aigua, registrades en temps real. Les dades s'utilitzen per visualitzar tendències i generar estadístiques.

```
_id: ObjectId('6748a914d1e749d386e51699')
tableId: "44"
temperature: 20.69
humidity: 676
waterLevel: 99.95
date: 2024-11-28T17:32:04.417+00:00
__v: 0
```

Figura 75: Dades dels sensors registrades en la base de dades.

La connexió amb la base de dades es defineix al fitxer [db.js](#):

- **Funció *connectDB*:** Aquesta funció asíncrona utilitza el mòdul *mongoose* per estableir una connexió amb el servidor de MongoDB local (adreça *mongodb://localhost:27017/TFG*). Gestiona automàticament els errors de connexió i mostra missatges a la consola per indicar si la connexió ha estat satisfactòria o si s'ha produït un error.

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/TFG');
    console.log('MongoDB connected successfully');
  } catch (error) {
    console.error('MongoDB connection failed:', error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

Figura 76: Definició de la funció *connectDB* al fitxer *db.js*.

Middlewares (backend/src/middlewares/) Els middlewares són components intermedis que processen les sol·licituds abans d'arribar a les rutes o després de ser gestionades per elles. Al projecte, s'han implementat dos middlewares principals que s'encarreguen de la seguretat i la gestió de fitxers.

- ***authMiddleware.js*:** Aquest middleware assegura que només els usuaris autènticats poden accedir a rutes protegides. Analitza el token JWT (*JSON Web Token*) inclòs a l'encapçalament de la sol·licitud HTTP i valida si és vàlid. Si el token és correcte, extreu l'*userId* del mateix i l'afegeix a la sol·licitud per a utilitzar-lo posteriorment en els controladors. En cas de no proporcionar un token o si aquest no és vàlid, el middleware retorna un error *401 Unauthorized*.

– Procés d'autenticació:

- 1) Verifica si el token existeix a l'encapçalament *Authorization*.
- 2) Divideix l'encapçalament per obtenir el token.
- 3) Utilitza la clau secreta (*your_jwt_secret*) per validar el token.
- 4) Si el token és vàlid, insereix el *userId* al cos de la sol·licitud per a ús posterior.
- 5) En cas d'error o de no proporcionar el token, retorna una resposta d'error.

```

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader) {
    return res.status(401).json({ message: 'No token provided' });
  }

  const token = authHeader.split(' ')[1];
  if (!token) {
    return res.status(401).json({ message: 'Invalid token' });
  }

  try {
    const decoded = jwt.verify(token, 'your_jwt_secret');
    req.userId = decoded.userId;
    console.log(`Decoded userId from token: ${req.userId}`);
    next();
  } catch (error) {
    res.status(401).json({ message: 'Invalid token' });
  }
};

module.exports = authMiddleware;

```

Figura 77: Definició del middleware d'autenticació al fitxer authMiddleware.js.

- **multerMiddleware.js:** Aquest middleware gestiona la pujada d'imatges al servidor. Utilitza la llibreria *Multer* per desar els fitxers de manera segura al directori `src/uploads/`. També inclou filtres per validar el tipus de fitxer (acceptant només imatges amb extensió jpg, jpeg, o png) i limita la mida màxima dels fitxers a 5 MB.

– Funcionalitats principals:

- 1) Configura el directori de destinació per desar les imatges.
- 2) Genera noms únics per als fitxers, basats en la data actual.
- 3) Verifica que el fitxer tingui un format vàlid i no excedeixi la mida màxima.
- 4) Retorna un error si el fitxer no compleix els criteris establerts.

```

const multer = require('multer');

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    console.log("Guardando archivo en la ruta:"); // Debugging para confirmar la ruta de destino
    cb(null, 'src/uploads/'); // Verifica que la ruta sea correcta y que existan los permisos
  },
  filename: (req, file, cb) => {
    console.log("Nombre del archivo guardado:"); // Debugging para confirmar el nombre del archivo
    cb(null, `${Date.now()}-${file.originalname}`);
  }
});

const upload = multer({
  storage: storage,
  filefilter: (req, file, cb) => {
    if (!file.originalname.match(/\.(jpg|jpeg|png)$/)) {
      console.log("Archivo inválido:", file.originalname); // Debugging para archivos no válidos
      return cb(new Error("Please upload a valid image (jpg, jpeg, or png)."));
    }
    cb(null, true);
  },
  limits: { fileSize: 5000000 } // 5 MB
});

module.exports = upload;

```

Figura 78: Definició del middleware per a la pujada d'imatges al fitxer multerMiddleware.js.

Arduino Connection (`arduino-connection/`) Aquesta carpeta conté el firmware desenvolupat específicament per al microcontrolador **Arduino Nano BLE 33 IoT**, que gestiona la interacció amb el hardware del sistema de reg automàtic. Aquest codi és fonamental per establir una comunicació efectiva entre el backend i els sensors i actuadors del projecte.

Descripció General El codi present a la carpeta `arduino-connection` permet coordinar i controlar els sensors i actuadors connectats al sistema, garantint una comunicació fluida amb el backend mitjançant el protocol **MQTT**. També s'encarrega de monitoritzar l'estat de les taules, recollir dades dels sensors i processar les ordres rebudes per executar accions en temps real.

3.2.2 Frontend

Tecnologies Gastades El frontend del sistema ha estat desenvolupat amb l'objectiu de proporcionar una interfície gràfica d'usuari (GUI) intuïtiva i funcional, que permeta als usuaris interactuar de manera efectiva amb el sistema.

- **Framework:** Utilització de **React.js**, un framework modern i escalable, per gestionar la interfície gràfica i proporcionar una experiència d'usuari fluida i interactiva.
- **Llibreries Auxiliars:**
 - **React Select:** Llibreria que proporciona llistes desplegables personalitzables i accessibles per gestionar dades d'entrada.
 - **Axios:** Llibreria per gestionar sol·licituds HTTP de manera eficient, connectant el frontend amb l'API del backend.
 - **React Toastify:** Permet mostrar notificacions visuals personalitzades per millorar l'experiència de l'usuari.
 - **React Slick:** Llibreria per a la creació de carrusels i sliders responsius.

Estructura del Frontend El codi del frontend està estructurat de manera modular per facilitar el manteniment i l'escalabilitat. A continuació, es descriu cada carpeta i els documents que conté [Directori Frontend](#).

Components (frontend/src/components/)

- **locationData.js:** Aquest fitxer conté una estructura de dades que defineix les províncies i regions organitzades per països. És especialment útil per desplegables i seleccions regionals al frontend. Aquestes dades inclouen noms de regions en diversos països com Espanya, Estats Units, Canadà, Brasil, entre altres.
- **Navbar.js:** Aquest component defineix la barra de navegació principal de l'aplicació. Utilitza *React Router* per crear enllaços a diferents seccions del sistema, com ara la pàgina d'inici, les taules i el perfil de l'usuari. Inclou icones de *Material-UI* per fer la interfície més intuïtiva i visualment agradable.
- **PrivateRoute.js:** És un component que actua com a ruta protegida, assegurant que només els usuaris autenticats puguin accedir a certes pàgines. Fa ús de *React Router* i verifica si existeix un token al *localStorage*. En cas contrari, redirigeix l'usuari a la pàgina de *login*.
- **WeatherConditions.js:** Aquest fitxer proporciona una traducció de condicions meteorològiques des d'anglès a espanyol. És útil per mostrar dades climàtiques a l'usuari en el seu idioma natiu. Conté condicions variades, des de *Clear* fins a fenòmens extrems com *Hurricane* o *Tornado*, assegurant una àmplia cobertura.

Pàgines del Frontend (frontend/src/) Les pàgines del frontend són els components clau de la interfície gràfica de l'aplicació. Cada pàgina està dissenyada per oferir funcionalitats específiques i connectar-se amb les rutes del backend. A continuació es detallen:

- **App.js**: És el punt d'entrada principal del frontend. Aquest fitxer defineix les rutes utilitzant **React Router** i connecta les diferents pàgines de l'aplicació. També protegeix algunes rutes utilitzant **PrivateRoute**.
- **HomePage.js**: Aquesta pàgina actua com la pantalla d'inici per als usuaris autenticats. Mostra dades meteorològiques en temps real, un assistent amb OpenAI per generar respostes a consultes de l'usuari, i dóna la benvinguda personalitzada basada en el nom d'usuari.
- **index.js**: Aquest fitxer renderitza el component **App.js** al navegador i actua com a punt d'entrada per a tota l'aplicació frontend.
- **LandingPage.js**: Aquesta pàgina és la primera vista que tenen els usuaris no autenticats. Inclou informació sobre el sistema, testimonis d'usuaris i botons per registrar-se o iniciar sessió.
- **LoginPage.js**: Permet als usuaris iniciar sessió al sistema. Inclou un formulari que valida les credencials amb el backend i redirigeix a la **HomePage.js** si l'autenticació és correcta.
- **RegisterPage.js**: Aquesta pàgina permet als nous usuaris registrar-se al sistema. Ofereix selectors dinàmics per països i províncies utilitzant **react-select**.
- **ProfilePage.js**: Mostra i permet editar les dades del perfil de l'usuari, incloent el nom, el correu electrònic, el telèfon i la imatge de perfil. També ofereix opcions per canviar la contrasenya o eliminar el compte.
- **TablesPage.js**: Aquesta pàgina permet gestionar totes les taules associades a l'usuari, incloent la creació, visualització i navegació als detalls específics de cada taula.
- **TableDetailsPage.js**: Mostra informació detallada sobre una taula seleccionada, com ara dades en temps real dels sensors, planificació de reg automàtic, control de vàlvules i estadístiques històriques.

CSS del Frontend (frontend/src/css/) Els fitxers CSS del frontend defineixen l'estil visual de l'aplicació, assegurant una experiència d'usuari consistent i atractiva. Cada pàgina i component principal té el seu propi fitxer CSS per mantenir l'organització i la modularitat. A continuació es detallen:

- `HomePage.css`: Aquest fitxer proporciona els estils específics per la pàgina d'inici. Inclou classes per a la disposició de la informació meteorològica, la interacció amb l'assistent IA i la presentació de la pàgina de benvinguda personalitzada. També incorpora estils per targetes i seccions destacades.
- `LandingPage.css`: Defineix l'aspecte visual de la pàgina de benvinguda inicial per als usuaris no autenticats. Inclou estils per la secció *Hero*, el carrusel d'imatges, les targetes d'informació i la secció de testimonis. Les animacions de transició per al carrusel també es gestionen aquí.
- `LoginPage.css`: Conté els estils per al formulari d'inici de sessió. Inclou dissenys per la targeta de login, els camps d'entrada, botons i enllaços per registrar-se. Està optimitzat per a una experiència neta i senzilla.
- `Navbar.css`: Gestiona l'aparença de la barra de navegació que és comuna a totes les pàgines. Inclou estils per als enllaços, icones i l'estructura de la barra de navegació, garantint que sigui responsiva i intuitiva.
- `ProfilePage.css`: Aplica els estils necessaris per a la pàgina del perfil d'usuari. Inclou dissenys per a la targeta de perfil, les opcions d'actualització de dades personals, els botons de gestió de perfil i les seccions d'eliminació de compte i tancament de sessió.
- `RegisterPage.css`: Estils dedicats a la pàgina de registre. Inclou dissenys per a les entrades de dades personals, selector dinàmics per a països i províncies, i els botons d'enviament. També gestiona el disseny responsiu del formulari de registre.
- `TableDetailsPage.css`: Proporciona els estils per a la pàgina de detalls d'una taula específica. Inclou dissenys per a targetes d'informació, gràfics de dades de sensors, botons per controlar les vàlvules, i seccions d'actualització i estadístiques de la taula.
- `TablesPage.css`: Defineix l'aparença de la pàgina de gestió de taules. Inclou estils per a la llista de taules creades, el formulari per crear noves taules i l'estructura del grid per mostrar les taules de manera organitzada i atractiva.

Carpeta public (frontend/public/) La carpeta `public` conté recursos estàtics que s'utilitzen en el frontend del projecte. Aquests recursos inclouen el fitxer base `index.html`, el favicon de l'aplicació i diverses imatges que són necessàries per a diferents seccions de la interfície. Aquesta carpeta es manté immutable, ja que no requereix canvis dinàmics durant l'execució de l'aplicació. A continuació es descriu el contingut:

- **index.html:** És el fitxer base que serveix com a punt d'entrada per a l'aplicació React. Aquest fitxer defineix l'estructura bàsica de la pàgina, incloent-hi:
 - **Meta etiquetes:** Proporciona informació sobre la descripció del projecte, les paraules clau, l'autor i altres configuracions útils per a SEO.
 - **Fonts i icones:** Enllaça les fonts de Google Fonts (*Roboto*) i la llibreria d'icones de *Material Icons*.
 - **Estils globals:** Conté estils CSS mínims per assegurar que el contenidor principal (`#root`) cobreixi tota la pantalla i utilitzi la tipografia global definida.
 - **Estructura de la pàgina:** Inclou el div `#root`, on React injectarà dinàmicament tot el contingut de l'aplicació.
- **favicon.ico:** És la icona que es mostra a la pestanya del navegador quan l'aplicació està oberta. Serveix per identificar visualment el projecte i millorar l'experiència de navegació.
- **Imatges estàtiques:** Aquesta carpeta inclou diverses imatges utilitzades en diferents parts de l'aplicació. Aquestes imatges són recursos immutables que ajuden a millorar la interfície gràfica i la usabilitat del projecte. A continuació es llisten:
 - `ahorroAgua.png`: Representa el benefici de l'estalvi d'aigua i s'utilitza en la secció d'informació de beneficis de la pàgina `LandingPage`.
 - `condition.png`: Icôna per mostrar condicions meteorològiques a la `HomePage`.
 - `controliot.png`: Imatge que destaca el control IoT a la secció de beneficis.
 - `counter.png`: Icôna utilitzada per representar estadístiques com el nombre de vegades que s'ha omplert o buidat una taula.
 - `country.png`: S'utilitza en el formulari de registre per indicar la selecció de país.
 - `cuidadoContinuo.png`: Imatge que representa el benefici del monitoratge constant en el sistema de reg.
 - `enterprise.png`: Icôna relacionada amb el nom de l'empresa de l'usuari a la pàgina `ProfilePage`.
 - `gmail.png`: Icôna utilitzada per mostrar l'adreça de correu electrònic de l'usuari.
 - `humidity.png`: Icôna per mostrar el nivell d'humitat en temps real.
 - `logo.png`: El logotip principal de l'aplicació, visible a la `LandingPage`.
 - `placeholder.png`: S'utilitza com a espai reservat per a imatges que no estan disponibles.

- **profileMan.png** i **profileWoman.png**: Imatges genèriques per als perfils d’usuari utilitzades en els testimonis.
- **schedule.png**: Ícone utilitzada per mostrar dades relacionades amb la programació de reg.
- **smartphone.png**: Ícone per mostrar el número de telèfon a la pàgina **ProfilePage**.
- **star.png**: Ícone de valoració que s’utilitza en la secció de testimonis.
- **temperature.png**: Ícone per mostrar la temperatura en temps real.
- **user.png**: Ícone per mostrar el nom d’usuari a la pàgina **ProfilePage**.
- **waterlevel.png**: Ícone que indica el nivell d’aigua de les taules.
- **wind.png**: Ícone per representar la velocitat del vent.

4 Validació

En aquesta secció es demostrarà el correcte funcionament del sistema complet mitjançant evidències visuals, captures de pantalla i imatges que permeten validar el backend, el frontend i els fluxos de treball implementats.

No obstant, el dia de la presentació del projecte es farà la demostració presencial mitjançant una maqueta.

4.1 Validació del Backend

Per validar el backend, es mostrerà l'arrancada del servidor i els logs generats durant el procés. Aquest apartat inclou captures de pantalla que evidencien que tots els serveis s'han iniciat correctament i estan en funcionament.

Mosquitto Broker en funcionament La primera imatge [Imatge Broker](#) mostra el broker MQTT Mosquitto en execució. S'observa que està esperant connexions entrants i escoltant correctament en els ports predeterminats, IPv4 i IPv6, en el port 1883. La configuració s'ha carregat amb èxit des de l'arxiu `mosquitto.conf`, i el servei confirma el seu estat de funcionament actiu amb el missatge *running*. Això indica que el broker està operatiu i llest per gestionar missatges i subsrcipcions.

```
PS C:\Users\marcc\Documents\TFG2\mosquitto_conf> mosquitto -c mosquitto.conf -v
1733349068: mosquitto version 2.0.19 starting
1733349068: Config loaded from mosquitto.conf.
1733349068: Opening ipv6 listen socket on port 1883.
1733349068: Opening ipv4 listen socket on port 1883.
1733349068: mosquitto version 2.0.19 running
```

Figura 79: Terminal del broker Mosquitto.

Arrencada del Backend En aquesta imatge [Imatge backend](#) es visualitza la terminal on s'ha iniciat el backend utilitzant el comanda `npm run dev`. El servidor ha estat inicialitzat correctament amb l'ajuda de `nodemon`, mostrant informació clau com:

- L'adreça IP detectada (192.168.1.39).
- Els serveis actius, incloent el servidor WebSocket al port 8080 i l'API RESTful al port 5000.
- La connexió exitosa al broker MQTT i a la base de dades MongoDB.

Aquestes sortides demostren que el backend està funcionant amb tots els seus serveis engegat correctament.

```
PS C:\Users\marcc\Documents\TFG2\backend> npm run dev
> backend@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
IP local detectada: 192.168.1.39
Servidor WebSocket corriendo en el puerto 8080
Server is running on port 5000
Conectado al broker MQTT
Suscripto correctamente a los tópicos de sensores y estado
MongoDB connected successfully
```

Figura 80: Terminal Backend.

Connexió amb el Broker MQTT Aquí Imatge Broker es mostra la confirmació de connexions entrants al broker MQTT. El backend s'ha connectat com a client (`backend_server`), subscrivint-se amb èxit als temes associats als sensors i a l'estat de les taules. A més, el broker confirma l'establiment correcte de les connexions amb el backend, evidenciant que està rebent i enviant dades segons la configuració definida.

```
PS C:\Users\marcc\Documents\TFG2\mosquitto_conf> mosquitto -c mosquitto.conf -v
1733349068: mosquitto version 2.0.19 starting
1733349068: Config loaded from mosquitto.conf.
1733349068: Opening ipv6 listen socket on port 1883.
1733349068: Opening ipv4 listen socket on port 1883.
1733349068: mosquitto version 2.0.19 running
1733349117: New connection from 192.168.1.39:52216 on port 1883.
1733349117: New client connected from 192.168.1.39:52216 as backend_server (p2,
c1, k60).
1733349117: No will message specified.
1733349117: Sending CONNACK to backend_server (0, 0)
1733349117: Received SUBSCRIBE from backend_server
1733349117:     table/+sensors/data (QoS 0)
1733349117: backend_server 0 table/+sensors/data
1733349117:     table/+status (QoS 0)
1733349117: backend_server 0 table/+status
1733349117: Sending SUBACK to backend_server
```

Figura 81: Evidència connexió backend Broker.

Connexió de l'Arduino Aquesta captura Imatge terminal arduino reflecteix la terminal de l'Arduino, mostrant com aquest s'ha connectat correctament al broker MQTT. Es verifica que s'han completat amb èxit les subscripcions als temes relacionats amb les comandes de la taula (`table/44/commands`) i que el sistema està preparat per rebre instruccions i enviar dades dels sensors.

```

Conectado al broker MQTT
Verificando tableId con el backend...
Código de estado: 200
Respuesta: {"message":"Table ID válido"}
tableId es válido.
Fallo al suscribirse al tópico de comandos: table/44/commands
Reconectado al broker MQTT
Suscripción exitosa al tópico de comandos: table/44/commands
Temperatura del agua: 85.00 °C
Humedad del suelo: 676
Porcentaje de llenado: 99.95%

```

Figura 82: Terminal de Arduino.

Flux de dades entre Arduino i Backend A la cinquena imatge [Imatge terminals Broker i Backend](#), es visualitzen les terminals del broker MQTT i del backend. S'observa que l'Arduino està enviant dades de sensors (temperatura, humitat i nivell d'aigua) al broker MQTT, i aquest les reenvia al backend. Els logs del backend confirmen que aquestes dades s'han rebut correctament i que s'han emmagatzemat a la base de dades amb els valors esperats.

<pre> 1733349777: Received PINGREQ from backend server 1733349813: Sending PINGRESP to backend server 1733349813: New connection from 192.168.1.37:58837 on port 1883. 1733349813: New client connected from 192.168.1.37:58837 as ArduinoClient (p2, c1, k1, 5). 1733349813: No will message specified. 1733349813: Sending CONNACK to ArduinoClient (0, 0) 1733349813: Client ArduinoClient closed its connection. 1733349813: New connection from 192.168.1.37:56150 on port 1883. 1733349813: New client connected from 192.168.1.37:56150 as ArduinoClient (p2, c1, k1, 5). 1733349813: Received SUBSCRIBE from ArduinoClient 1733349813: Received PUBSUBSCRIBE from ArduinoClient 1733349813: table/44/commands (QoS 0) 1733349813: ArduinoClient 0 table/44/commands 1733349813: Sending SUBACK to ArduinoClient 1733349813: Received PUBLISH from ArduinoClient (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349813: Sending PUBLISH to backend_server (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349823: Received PUBLISH from ArduinoClient (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349823: Received PUBLISH from ArduinoClient (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349833: Sending PUBLISH to backend_server (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349833: Received PUBLISH from ArduinoClient (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349833: Sending PUBLISH to backend_server (d0, q0, r0, m0, 'table/44/sensors/data', ... (55 bytes)) 1733349837: Received PINGREQ from backend server </pre>	<pre> Datos de sensores guardados correctamente en la base de datos. Cron job ejecutado: 4/12/2024, 23:04:00 Hora actual: 23:04, Día actual: miércoles Mesas encontradas para llenado (0): Mesas encontradas para vaciado (0): Mensaje recibido en el tópico table/44/sensors/data: {"temperature":22.5 0,"humidity":682,"waterlevel":99.95} Datos de sensores recibidos: { temperature: 22.5, humidity: 682, waterlevel: 99.95 } Datos de sensores guardados correctamente en la base de datos. Mensaje recibido en el tópico table/44/sensors/data: {"temperature":22.5 0,"humidity":682,"waterlevel":99.95} Datos de sensores recibidos: { temperature: 22.5, humidity: 682, waterlevel: 99.95 } Datos de sensores guardados correctamente en la base de datos. Mensaje recibido en el tópico table/44/sensors/data: {"temperature":22.5 0,"humidity":685,"waterlevel":99.97} Datos de sensores recibidos: { temperature: 22.5, humidity: 685, waterlevel: 99.97 } Datos de sensores guardados correctamente en la base de datos. Mensaje recibido en el tópico table/44/sensors/data: {"temperature":22.5 0,"humidity":693,"waterlevel":99.97} Datos de sensores recibidos: { temperature: 22.5, humidity: 693, waterlevel: 99.97 } Datos de sensores guardados correctamente en la base de datos. </pre>
---	---

Figura 83: Terminals Broker i Backend.

Almacenament de dades a MongoDB En aquesta imatge [Imatge BDD](#) es mostra el registre d'una col·lecció a MongoDB on s'han guardat les lectures dels sensors associades a la taula `tableId: "44"`. Els valors registrats, com la temperatura (22.5 °C), humitat (682) i nivell d'aigua (99.95%), són consistents amb els valors enviats per l'Arduino i processats pel backend. Això confirma que la integració amb la base de dades funciona sense errors.

```

_id: ObjectId('6750d26ab140007005d4e887')


```

Figura 84: Esquema del backend.

4.2 Validació del Frontend

Aquest apartat demostra que la interfície gràfica del sistema està operativa i respon correctament. S'inclouen captures de pantalla que mostren l'aplicació arrencada i navegant per les pàgines principals.

Arrencada del Frontend La primera captura mostra el procés d'arrencada del frontend mitjançant la comanda `npm start`. En aquesta terminal es pot observar que la compilació es realitza amb èxit, indicant que el servidor està actiu i disponible localment en `http://localhost:3000` i accessible a la xarxa a través de l'adreça IP `http://192.168.1.39:3000`. Aquest missatge confirma que l'entorn de desenvolupament està configurat correctament i llest per a ser utilitzat.

```

Compiled successfully!

You can now view frontend in the browser.

  Local:          http://localhost:3000
  On Your Network: http://192.168.1.39:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully

```

Figura 85: Captura de la terminal Frontend

Landing Page Les captures dos i tres mostren la interfície principal del sistema, coneguda com la *Landing Page*.

En la segona imatge, es visualitza el capçal amb el títol *Benvingut a RiegAI*, acompanyat d'un breu eslògan que destaca l'automatització del reg de plantes mitjançant tecnologia IoT. També es presenten dos botons principals: *Registra't* i *Inicia Sessió*, que redirigeixen l'usuari a les pàgines corresponents.

En la tercera imatge, es presenta una descripció més detallada sobre el sistema, titulada *Què és RiegAI?*. Es destaquen tres característiques principals:

- **Estalvi d'Aigua:** Optimització de l'ús de l'aigua per a reduir el malbaratament.
- **Control IoT:** Administració del sistema des de qualsevol lloc mitjançant l'aplicació web.
- **Cura Contínua:** Monitoratge 24/7 de les plantes gràcies a sensors intel·ligents.

A més, s'inclouen testimonis d'usuaris satisfets en la part inferior, la qual cosa reforça la confiança en el sistema.

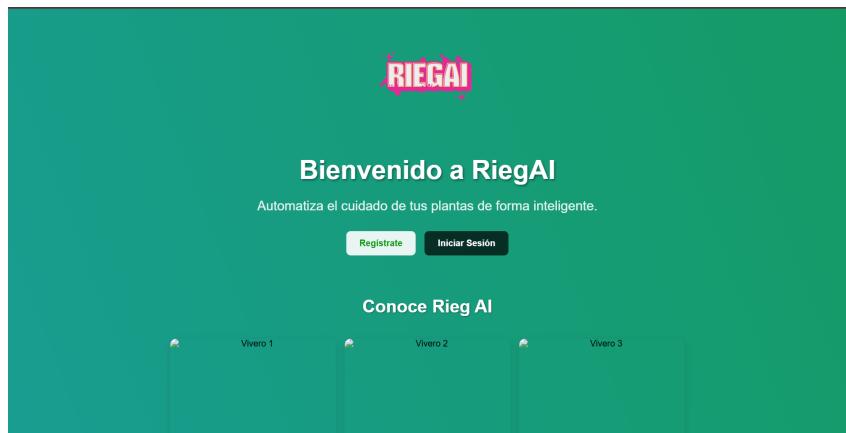


Figura 86: Landing Page 1/2



Figura 87: Landing Page 2/2

Formulari de Registre La quarta imatge mostra el formulari de registre del sistema. Aquest formulari inclou els següents camps obligatoris:

- **Nom d'Usuari:** Per a identificar l'usuari en el sistema.
- **Contrasenya:** Clau d'accés encriptada per a la seguretat de l'usuari.
- **Email:** Adreça de correu electrònic de l'usuari.
- **Telèfon:** Número de contacte.
- **País:** Selecció del país de residència mitjançant un menú desplegable.
- **Nom de la Companyia:** Informació opcional sobre l'empresa de l'usuari.

En la part inferior, s'inclou un botó *Registra't* i un enllaç per a redirigir a la pàgina d'inici de sessió en cas que l'usuari ja tinga un compte.

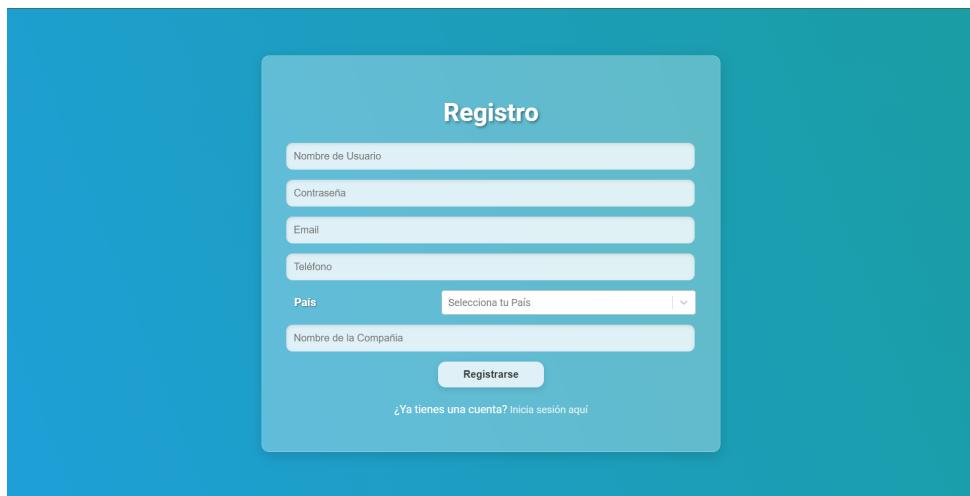


Figura 88: Register Page

Formulari de Login La cinquena captura correspon al formulari d'inici de sessió (*Login*). Aquest formulari és més senzill i sol·licita únicament:

- **Nom d'Usuari.**
- **Contrasenya.**

Inclou un botó per a iniciar sessió i un enllaç per a redirigir al formulari de registre en cas que l'usuari no tinga un compte. Aquest disseny minimalist facilita un accés ràpid i segur al sistema.

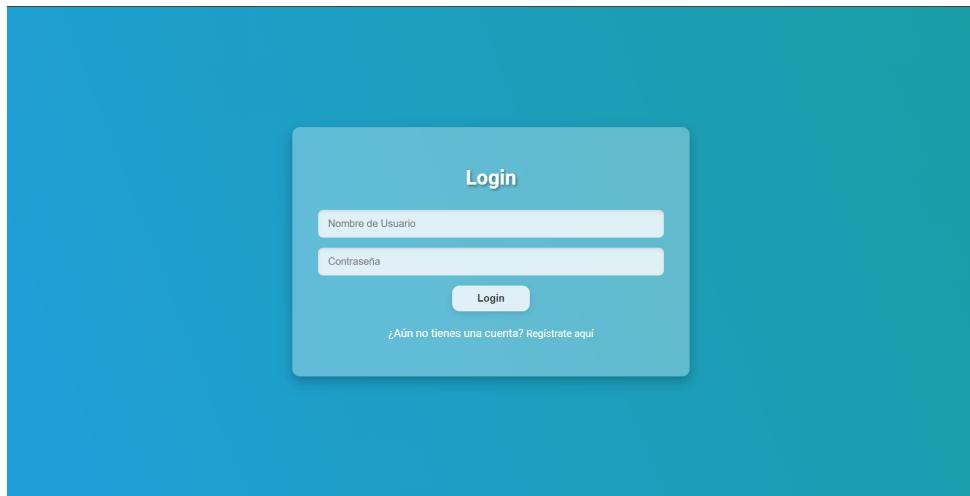


Figura 89: Login Page

Home Page En la sisena imatge s'observa la *Home Page*, dissenyada per a donar la benvinguda a l'usuari autenticat. En aquest cas, es personalitza amb el missatge *Benvingut, Marc*. La pàgina inclou un panell informatiu sobre les condicions meteorològiques actuals, com ara:

- Temperatura.
- Humitat.
- Condició climàtica.
- Velocitat del vent.

També s'inclou un *Assistent IA* per a interactuar amb el sistema d'intel·ligència artificial integrat, permetent a l'usuari realitzar consultes o sol·licitar assistència.



Figura 90: Home Page

Gestió de Taules La setena imatge mostra la pàgina *TablesPage*, que permet a l'usuari gestionar les taules de reg automàtic. En aquesta pàgina es troben dues seccions principals:

- **Crear una Nova Taula:** Un formulari senzill que sol·licita el nom i la descripció de la taula.
- **Llista de Taules:** Un panell que mostra les taules creades per l'usuari, identificades amb el seu nom i una imatge representativa.

Aquest disseny intuïtiu permet a l'usuari afegir i visualitzar taules de manera ràpida i eficient.

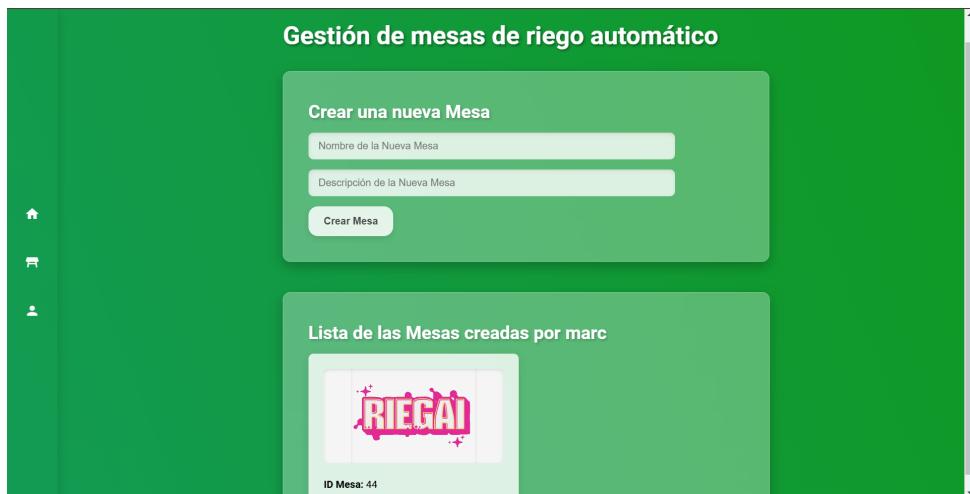


Figura 91: Tables Page

Detalls de la Taula i Configuració La pàgina *TableDetailsPage.js* proporciona una interície detallada per gestionar cadascuna de les taules del sistema. En la primera captura, es mostren els detalls bàsics de la taula, incloent el seu ID (44), el nom personalitzat ("mesa nueva") i una descripció ("ifiwuefhuw"). A més, s'inclou l'opció de pujar una imatge associada a la taula, afegint un toc personalitzat a l'experiència d'usuari. La segona captura destaca la secció de *Control de Vàlvules*, on es mostra l'estat actual de la taula (per exemple, "Buida") i es proporcionen botons per omplir o buidar la taula segons sigui necessari. Aquesta funcionalitat permet una gestió directa del flux d'aigua. En la tercera imatge, es poden veure les dades dels sensors en temps real, incloent la temperatura de l'aigua (23.56 °C), el nivell d'humitat ("Sec") i el nivell d'aigua (99.95%). Això assegura que l'usuari pugui monitorar en tot moment l'estat de les plantes. Finalment, la quarta captura mostra la secció de *Planificació de Reg*, on l'usuari pot seleccionar els dies i horaris per omplir i buidar la taula, adaptant el sistema a les necessitats específiques de les plantes.



Figura 92: Pàgina de Detalls de la Taula

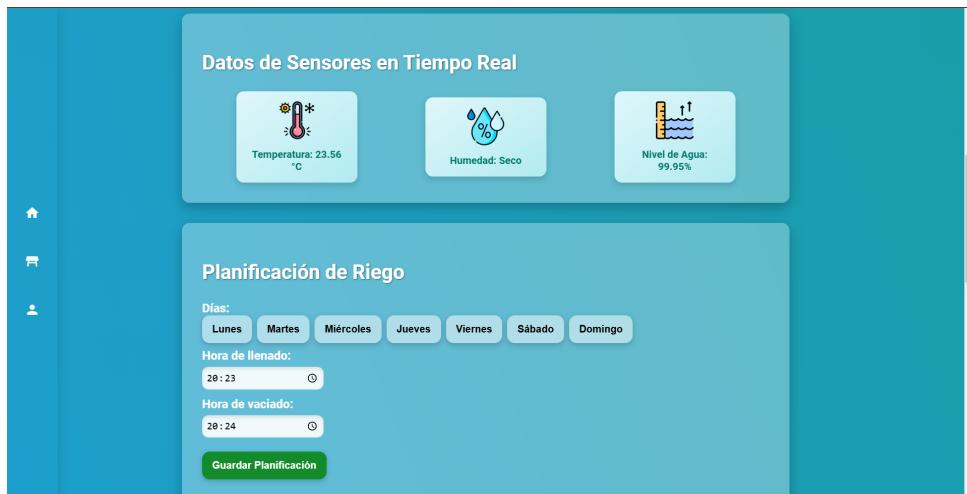


Figura 93: Pàgina de Detalls de la Taula



Figura 94: Pàgina de Detalls de la Taula



Figura 95: Pàgina de Details de la Taula

Perfil de l'Usuari i Ajustos La pàgina ProfilePage.js permet gestionar les dades personals de l'usuari de manera eficient. En la cinquena captura, es mostra la secció principal del perfil, amb informació com el nom ("marc"), el correu electrònic, el número de telèfon, el país, la província i la companyia associada. També s'inclou la possibilitat de pujar una nova imatge de perfil, oferint una experiència més personalitzada. La sisena imatge mostra un formulari complet per actualitzar les dades personals, com el nom, el correu electrònic, el telèfon i la contrasenya. A més, es presenten opcions per accions crítiques com eliminar el compte o tancar la sessió. Aquesta funcionalitat garanteix que l'usuari tingui un control total sobre les seves dades i pugui modificar-les segons sigui necessari.

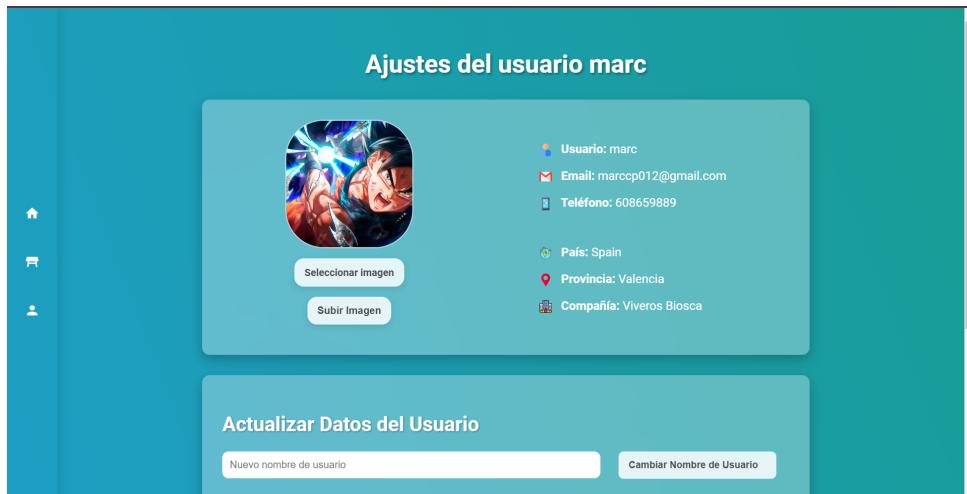


Figura 96: Settings Page



Figura 97: Settings Page

4.3 Demostració de Fluxos de Treball

En aquest apartat es presenten exemples pràctics de com funciona el sistema en diferents fluxos de treball. Les proves inclouen captures de pantalla per validar funcionalitats específiques, mostrant totes les pantalles implicades en el procés.

4.3.1 Flux 1: Registre i Login d'un Usuari

El primer flux mostra el procés de registre i posterior inici de sessió d'un nou usuari. Aquest flux valida que el sistema permet la creació de comptes i l'accés segur al sistema.

Procés de Registre En la Figura 98, es pot observar el formulari de registre completat per a un usuari nou anomenat *adria*. En aquest formulari, l'usuari introduceix la informació requerida: nom d'usuari, contrasenya, correu electrònic, número de telèfon, país, província i nom de la companyia. Després de prémer el botó *Registrar-se*, el sistema valida les dades i crea correctament el nou compte d'usuari.

Figura 98: Formulari de registre d'un nou usuari.

Procés de Login Un cop completat el registre, l'usuari pot iniciar sessió utilitzant les seves credencials. La Figura 99 mostra el formulari de login on l'usuari *adria* introduceix el seu nom d'usuari i contrasenya. Després de premer el botó *Login*, el sistema verifica les credencials i accedeix correctament a la interfície principal.

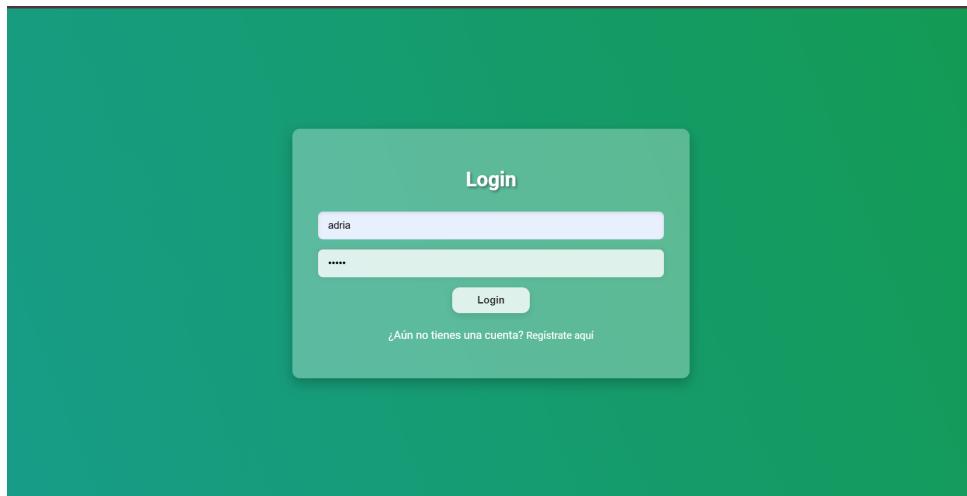


Figura 99: Formulari de login per accedir al sistema.

Interfície d'inici Després d'iniciar sessió, l'usuari accedeix a la pàgina principal, tal com es mostra a la Figura 100. Aquesta pàgina dona la benvinguda a l'usuari amb el seu nom i mostra informació meteorològica en temps real, com ara la temperatura, la humitat, les condicions del cel i la velocitat del vent. A més, inclou un assistent IA per a la interacció de l'usuari.



Figura 100: Pàgina principal després de l'inici de sessió.

4.3.2 Flux 2: Plenatge d'una Taula

El procés de plenatge d'una taula es desenvolupa de manera clara i intuïtiva mitjançant l'aplicació. Les següents imatges il·lustren l'evolució d'aquest procés, començant amb l'estat inicial de la taula i acabant amb la confirmació del plenatge complet.

En aquesta primera imatge, es mostra l'estat inicial de la taula, que es troba en “Buit”. Es poden visualitzar detalls com el nom, la descripció i l'ID de la taula. A més, l'usuari pot seleccionar o pujar una imatge associada a la taula.

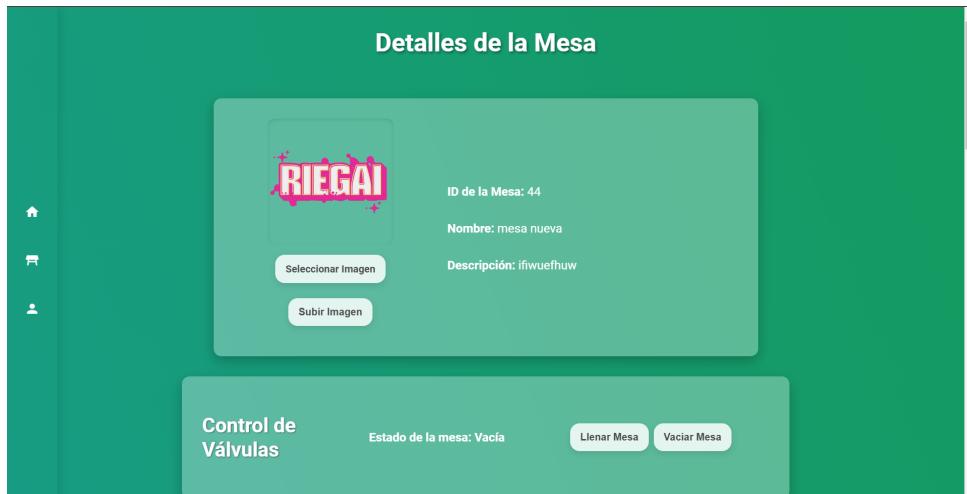


Figura 101: Detalls de la taula en estat inicial.

Quan es pressiona el botó “Llenar Mesa”, el sistema mostra un missatge de confirmació en temps real que indica “Llenando la mesa...”. A més, l'estat de la taula canvia a “Llenándose”, tal com es pot veure en aquesta segona imatge.

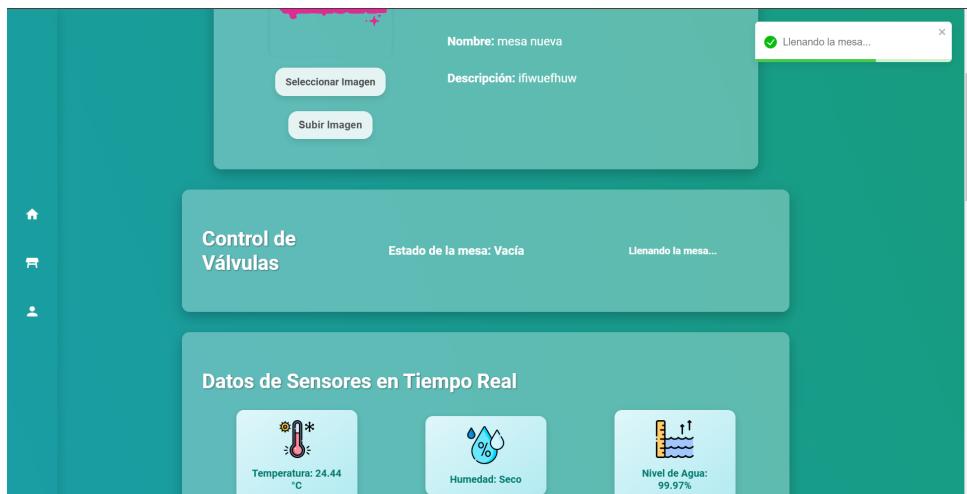


Figura 102: Estat actualitzat durant el procés de plenatge.

En aquesta imatge es manté l'estat de “Llenándose”, mostrant com el sistema actualitza en temps real les dades proporcionades pels sensors, incloent temperatura, humitat i nivell d'aigua.

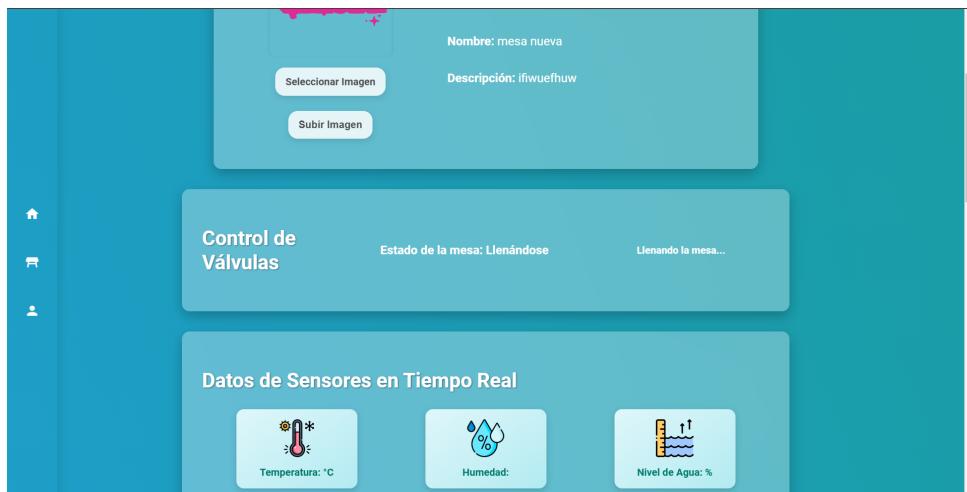


Figura 103: Inici del procés de plenatge.

Finalment, una vegada completat el procés, l'estat de la taula es marca com “Plena”. Això confirma que el plenatge ha estat satisfactori i que la vàlvula ha finalitzat correctament la seva operació.

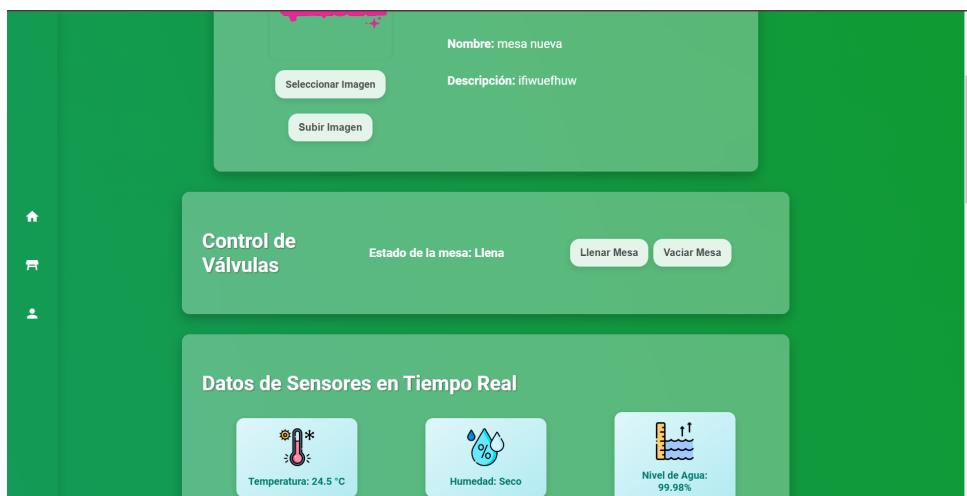


Figura 104: Estat final de la taula després de completar el plenatge.

En aquesta última imatge es pot observar la terminal de l’Arduino, que detalla el procés tècnic realitzat. L’Arduino rep el missatge MQTT amb el tema associat a la taula i la instrucció ‘fillTable’. Executa la funció de plenatge, activa la vàlvula corresponent i publica l’estat actualitzat, passant de ”filling” (plenatge) a ”full” (plena). Això assegura que el plenatge ha estat complet i sincronitzat amb l’aplicació.

```
Mensaje recibido en el Arduino:  
Tópico: table/44/commands  
Mensaje: fillTable  
Comando de llenado recibido. Ejecutando llenarMesa().  
Llenando la mesa...  
Publicando en MQTT:  
table/44/status: filling  
Válvula de llenado cerrada después de 5 segundos  
Publicando en MQTT:  
table/44/status: full
```

Figura 105: Registre de la terminal de l'Arduino.

5 Conclusions

5.1 Conclusions

Aquest projecte ha assolit l'automatització efectiva del reg de taules de cultiu mitjançant tecnologies IoT. S'han integrat funcionalitats clau com el control de vàlvules, la monitorització en temps real i l'emmagatzematge de dades a MongoDB, permetent als usuaris gestionar les taules de manera remota i eficient.

La comunicació fiable entre hardware i software, garantida pels protocols MQTT i Web-Socket, assegura temps de resposta ràpids i una sincronització òptima. A més, la interfície intuitiva i la personalització de la planificació de reg proporcionen una experiència d'usuari senzilla i completa, destacant l'eficiència i sostenibilitat del sistema.

Els resultats confirmen que el sistema és robust i capaç de gestionar diverses taules simultàniament, aportant dades útils per optimitzar recursos com l'aigua.

5.2 Treball Futur

Malgrat els èxits aconseguits, hi ha marge per a millors significatives:

- **Aprenentatge automàtic:** Analitzar dades històriques dels sensors per predir necessitats de reg i automatitzar encara més el sistema.
- **Integració mòbil:** Implementar notificacions push per alertar d'incidències com l'escassetat d'aigua o errors als sensors.
- **Nous sensors:** Afegir sensors de pH o conductivitat per proporcionar informació més detallada del cultiu.
- **Energia sostenible:** Utilitzar energia solar per alimentar el hardware i fer el sistema autosuficient.
- **Codi obert:** Publicar el projecte per fomentar la col·laboració i adaptació en altres aplicacions d'agricultura intel·ligent.

6 Bibliografía

¿Qué es MQTT? - Explicación del protocolo MQTT [[En línia]. Disponible en: [\[https://aws.amazon.com/es/what-is/mqtt/\]](https://aws.amazon.com/es/what-is/mqtt/). [Últim accés: desembre 2024].]. (2024). Amazon Web Services (AWS). (Vid. pág. 23).

¿Qué es MQTT? El protocolo más utilizado para IoT [[En línia]. Disponible en: [\[https://pandorafms.com/es/it-topics/que-es-mqtt/\]](https://pandorafms.com/es/it-topics/que-es-mqtt/). [Últim accés: desembre 2024].]. (2024). Pandora FMS. (Vid. pág. 23).

Aggregation Framework in MongoDB [[En línia]. Disponible en: [\[https://www.mongodb.com/docs/manual/aggregation/\]](https://www.mongodb.com/docs/manual/aggregation/). [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).

Arduino Documentation [[En línia]. Disponible en: [\[https://docs.arduino.cc/\]](https://docs.arduino.cc/). [Últim accés: desembre 2024].]. (2024). Arduino. (Vid. pág. 34).

Arduino Libraries [[En línia]. Disponible en: [\[https://www.arduino.cc/en/reference/libraries\]](https://www.arduino.cc/en/reference/libraries). [Últim accés: desembre 2024].]. (2024). Arduino. (Vid. pág. 34).

Arduino Nano 33 IoT Datasheet [[Online]. Available: [\[https://docs.arduino.cc/resources/datasheets/ABX00027-datasheet.pdf\]](https://docs.arduino.cc/resources/datasheets/ABX00027-datasheet.pdf). [Accessed: Dec. 2024]]. (2024). Arduino. (Vid. págs. 7, 8).

Arduino Nano 33 IoT Documentation [[Online]. Available: [\[https://docs.arduino.cc/hardware/nano-33-iot\]](https://docs.arduino.cc/hardware/nano-33-iot). [Accessed: Dec. 2024]]. (2024). Arduino. (Vid. págs. 7, 8).

Arduino Software [[En línia]. Disponible en: [\[https://www.arduino.cc/en/software\]](https://www.arduino.cc/en/software). [Últim accés: desembre 2024].]. (2024). Arduino. (Vid. pág. 34).

Axios: Promise based HTTP client for the browser and Node.js [[En línia]. Disponible en: [\[https://axios-http.com/\]](https://axios-http.com/). [Últim accés: desembre 2024].]. (2024). Axios. (Vid. pág. 28).

Batlle, R. (2024). Sistemes de Reg Automàtic. (Vid. pág. 3).

Building Queries in MongoDB Compass [[En línia]. Disponible en: [\[https://www.mongodb.com/docs/compass/current/\]](https://www.mongodb.com/docs/compass/current/). [Últim accés: desembre 2024].]. (2024). MongoDB Inc. (Vid. pág. 38).

Capacitive Soil Moisture Sensor v1.2: Características i funcionament [Disponible en: <https://naylampmechatronics.com>. Últim accés: desembre 2024].]. (2024). Naylamp Mechatronics. (Vid. pág. 9).

CitySens. (2024). Sistemes de Reg Automàtic CitySens. (Vid. pág. 3).

Datasheet Oficial del SRD-05VDC-SL-C [Disponible en: <https://songlerelay.com/srd-05vdc-sl-c-datasheet.pdf>. Últim accés: desembre de 2024].]. (2024). Songle Relay Co. (Vid. pág. 14).

Datasheet Tècnica de Vàlvules Solenoides DC 12V i 4A [Disponible en: <https://solenoidcomponents.com/solenoid-dc12v-4a-datasheet.pdf>. Últim accés: desembre de 2024].]. (2024). Solenoid Components Co. (Vid. pág. 12).

- Debugging in Visual Studio Code* [[En línia]. Disponible en: <https://code.visualstudio.com/docs/editor/debugging>. [Últim accés: desembre 2024].]. (2024). Microsoft. (Vid. pág. 36).
- Descripció tècnica del Capacitive Soil Moisture Sensor v1.2* [Disponible en: <https://ssdselect.com/sensors>. Últim accés: desembre 2024].]. (2024). SSDSELECT. (Vid. pág. 9).
- DS18B20 Datasheet* [Disponible en: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Últim accés: desembre 2024].]. (2024). Maxim Integrated. (Vid. pág. 10).
- Eclipse Mosquitto Documentation* [[En línia]. Disponible en: <https://mosquitto.org/documentation/>. [Últim accés: desembre 2024].]. (2024). Eclipse Foundation. (Vid. pág. 35).
- Express.js Documentation* [[En línia]. Disponible en: <https://expressjs.com/en/4x/api.html>. [Últim accés: desembre 2023].]. (2023). Express.js. (Vid. pág. 18).
- Extensions Marketplace for VS Code* [[En línia]. Disponible en: <https://marketplace.visualstudio.com/vscode>. [Últim accés: desembre 2024].]. (2024). Microsoft. (Vid. pág. 36).
- Fette, I., & Melnikov, A. (2011). *RFC 6455: The WebSocket Protocol* [[En línia]. Disponible en: <https://tools.ietf.org/html/rfc6455>. [Últim accés: desembre 2024].]. (Vid. pág. 24).
- Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [Tesis doctoral, University of California, Irvine] [[En línia]. Disponible en: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm > [Últim accés: desembre 2024].]. (Vid. pág. 19).
- FreeCodeCamp. (2024). *React-Toastify Guide* [[En línia]. Disponible en: <https://www.freecodecamp.org/news/react-toastify-tutorial/>. [Últim accés: desembre 2024].]. (Vid. pág. 33).
- Group, L. (2024). Reg Automàtic. (Vid. pág. 2).
- Guia d'Integració en Projectes IoT* [Disponible en: <https://iotapplications.com/solenoid-integration-guide.pdf>. Últim accés: desembre de 2024].]. (2024). IoT Applications Co. (Vid. pág. 13).
- Guia d'ús del Mòdul SRD-05VDC-SL-C* [Disponible en: <https://iotguides.com/rele-srd05vdc-usage.pdf>. Últim accés: desembre de 2024].]. (2024). IoT Automation Guides. (Vid. pág. 14).
- A guide to React Router for beginners* [[En línia]. Disponible en: <https://blog.logrocket.com/react-router-guide-beginners/>. [Últim accés: desembre 2024].]. (2024). LogRocket. (Vid. pág. 32).
- HC-SR04 Ultrasonic Sensor Module User Guide* [Disponible en: <https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf>. Últim accés: desembre de 2024].]. (2024). Handson Technology. (Vid. pág. 12).
- i Jardins Catalunya, P. (2024). Regs Automàtics per Jardins i Espais Verds. (Vid. pág. 3).

Implementing React-Toastify [[En línia]. Disponible en: [¡https://blog.logrocket.com/react-toastify-notifications/¡.](https://blog.logrocket.com/react-toastify-notifications/) [Últim accés: desembre 2024].]. (2024). LogRocket Blog. (Vid. pág. 33).

Index Management in MongoDB Compass [[En línia]. Disponible en: [¡https://www.mongodb.com/docs/compass/current/indexes/¡.](https://www.mongodb.com/docs/compass/current/indexes/) [Últim accés: desembre 2024].]. (2024). MongoDB Inc. (Vid. pág. 38).

Indexes in MongoDB [[En línia]. Disponible en: [¡https://www.mongodb.com/docs/manual/indexes/¡.](https://www.mongodb.com/docs/manual/indexes/) [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).

JSON Web Token: Secure Token-Based Authentication [[En línia]. Disponible en: [¡https://jwt.io/¡.](https://jwt.io/) [Últim accés: desembre 2024].]. (2024). jwt.io. (Vid. pág. 28).

MongoDB Compass Documentation [[En línia]. Disponible en: [¡https://www.mongodb.com/docs/compass/¡.](https://www.mongodb.com/docs/compass/) [Últim accés: desembre 2024].]. (2024). MongoDB Inc. (Vid. pág. 38).

MongoDB Documentation [[En línia]. Disponible en: [¡https://www.mongodb.com/docs/¡.](https://www.mongodb.com/docs/) [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).

MQTT: The Standard for IoT Messaging [[En línia]. Disponible en: [¡https://mqtt.org/¡.](https://mqtt.org/) [Últim accés: desembre 2024].]. (2024). MQTT.org. (Vid. págs. 23, 35).

Multer: Middleware for handling multipart/form-data [[En línia]. Disponible en: [¡https://github.com/expressjs/multer/¡.](https://github.com/expressjs/multer/) [Últim accés: desembre 2024].]. (2024). Express.js Middleware. (Vid. pág. 28).

Node.js v20.5.1 Documentation [[En línia]. Disponible en: [¡https://nodejs.org/dist/latest-v20.x/docs/api/¡.](https://nodejs.org/dist/latest-v20.x/docs/api/) [Últim accés: desembre 2023].]. (2023). OpenJS Foundation. (Vid. pág. 17).

Nodemailer: Send emails with Node.js [[En línia]. Disponible en: [¡https://nodemailer.com/¡.](https://nodemailer.com/) [Últim accés: desembre 2024].]. (2024). Nodemailer. (Vid. pág. 28).

OpenAI. (2024a). *Chat Completions API* [Última visita: Desembre 2024]. (Vid. pág. 26).

OpenAI. (2024b). *GPT-3.5 and GPT-4 Models Documentation* [Última visita: Desembre 2024]. (Vid. pág. 26).

OpenAI. (2024c). *OpenAI API Documentation* [Última visita: Desembre 2024]. (Vid. pág. 26).

OpenAI. (2024d). *OpenAI API Overview* [Última visita: Desembre 2024]. (Vid. pág. 26).

Postman Documentation [[En línia]. Disponible en: [¡https://learning.postman.com/¡.](https://learning.postman.com/) [Últim accés: desembre 2024].]. (2024). Postman Inc. (Vid. pág. 37).

Postman Product Overview [[En línia]. Disponible en: [¡https://www.postman.com/¡.](https://www.postman.com/) [Últim accés: desembre 2024].]. (2024). Postman Inc. (Vid. pág. 37).

Querying Data in MongoDB [[En línia]. Disponible en: [¡https://www.mongodb.com/docs/manual/tutorial/query-documents/¡.](https://www.mongodb.com/docs/manual/tutorial/query-documents/) [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).

- React Docs - React Hooks Reference* [[En línia]. Disponible en: <https://es.react.dev/reference/react>. [Últim accés: desembre 2024].]. (2024). Meta (Facebook). (Vid. pág. 30).
- React Docs - Start Learning React* [[En línia]. Disponible en: <https://es.react.dev/learn>. [Últim accés: desembre 2024].]. (2024). Meta (Facebook). (Vid. pág. 30).
- React Router Tutorial for Beginners* [[En línia]. Disponible en: <https://www.freecodecamp.org/news/react-router-tutorial/>. [Últim accés: desembre 2024].]. (2024). freeCodeCamp. (Vid. pág. 32).
- React Router: Declarative routing for React.js* [[En línia]. Disponible en: <https://reactrouter.com/>. [Últim accés: desembre 2024].]. (2024). React Training. (Vid. pág. 32).
- React-Toastify Documentation* [[En línia]. Disponible en: <https://fkhadra.github.io/react-toastify/>. [Últim accés: desembre 2024].]. (2024). React-Toastify. (Vid. pág. 33).
- React-Toastify on npm* [[En línia]. Disponible en: <https://www.npmjs.com/package/react-toastify>. [Últim accés: desembre 2024].]. (2024). npm, Inc. (Vid. pág. 33).
- React: A JavaScript library for building user interfaces* [[En línia]. Disponible en: <https://es.react.dev/>. [Últim accés: desembre 2024].]. (2024). Meta (Facebook). (Vid. pág. 30).
- Regaber, M. (2024). Solucions de Reg Professional. (Vid. pág. 2).
- Replication in MongoDB* [[En línia]. Disponible en: <https://www.mongodb.com/docs/manual/replication>. [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).
- Schema Analysis in MongoDB Compass* [[En línia]. Disponible en: <https://www.mongodb.com/docs/compass/current/schema-analyzer/>. [Últim accés: desembre 2024].]. (2024). MongoDB Inc. (Vid. pág. 38).
- Sharding in MongoDB* [[En línia]. Disponible en: <https://www.mongodb.com/docs/manual/sharding>. [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).
- Transactions in MongoDB* [[En línia]. Disponible en: <https://www.mongodb.com/docs/manual/transactions>. [Últim accés: desembre 2024].]. (2024). MongoDB, Inc. (Vid. pág. 21).
- Transformador de 12V i 2A: Full tècnica* [Disponible en: <https://ejemplo-transformador-12v.com>. Últim accés: desembre de 2024].]. (2024). Electronics Power Co. (Vid. pág. 15).
- Transformador de 5V i 2A: Full tècnica oficial* [Disponible en: <https://ejemplo-transformador-5v.com>. Últim accés: desembre de 2024].]. (2024). MicroPower Supplies. (Vid. pág. 16).
- Tutorials, R. N. (2024). *Guia d'ús del sensor DS18B20* [Disponible en: <https://randomnerdtutorials.com/ds18b20-temperature-sensor-arduino/>. Últim accés: desembre 2024].]. (Vid. págs. 10, 11).

Ultrasonic Ranging Module HC-SR04 Datasheet [Disponible en: <https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>. Últim accés: desembre de 2024]. (2024). SparkFun Electronics. (Vid. pág. 11).

Understanding RESTful API Architecture [[En línia]. Disponible en: <https://apidesign.com/restful-api-diagram>. [Últim accés: desembre 2024]]. (2024). API Design Co. (Vid. pág. 19).

Visual Studio Code Documentation [[En línia]. Disponible en: <https://code.visualstudio.com/docs>. [Últim accés: desembre 2024]]. (2024). Microsoft. (Vid. pág. 36).

WebSockets [[En línia]. Disponible en: <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>. [Últim accés: desembre 2024]]. (2024). MDN Web Docs. (Vid. pág. 24).

What is Postman? [[En línia]. Disponible en: <https://www.postman.com/product/what-is-postman/>. [Últim accés: desembre 2024]]. (2024). Postman Inc. (Vid. pág. 37).

Annex II - Relació del TFG amb els ODS

Tabla 3: Relació del treball amb els ODS

Objectius de Desenvolupament Sostenible	Alt	Mitjà	Baix	No procedeix
ODS 1. Fi de la pobresa.				X
ODS 2. Fam zero.				X
ODS 3. Salut i benestar.		X		
ODS 4. Educació de qualitat.			X	
ODS 5. Igualtat de gènere.				X
ODS 6. Aigua neta i sanejament.	X			
ODS 7. Energia assequible i no contaminant.				X
ODS 8. Treball digne i creixement econòmic.				X
ODS 9. Indústria, innovació i infraestructures.	X			
ODS 10. Reducció de les desigualtats.				X
ODS 11. Ciutats i comunitats sostenibles.		X		
ODS 12. Producció i consum responsables.		X		
ODS 13. Acció pel clima.		X		
ODS 14. Vida submarina.				X
ODS 15. Vida d'ecosistemes terrestres.		X		
ODS 16. Pau, justícia i institucions sòlides.				X
ODS 17. Aliances per aconseguir els objectius.				X

Explicació dels ODS marcats:

- **ODS 3 - Salut i benestar:** El projecte RiegAI contribueix a la millora del benestar en les pràctiques agrícoles reduint l'esforç físic dels operaris i assegurant condicions òptimes per al cultiu de plantes.
- **ODS 6 - Aigua neta i sanejament:** L'ús eficient i optimitzat de l'aigua garanteix la sostenibilitat en el reg de cultius, evitant el malbaratament d'aquest recurs essencial.
- **ODS 9 - Indústria, innovació i infraestructures:** La integració d'IoT, intel·ligència artificial i tecnologies modernes com Node.js i React.js fomenta la innovació tecnològica en el sector agrícola.
- **ODS 11 - Ciutats i comunitats sostenibles:** Amb un sistema de reg automàtic, s'optimitzen els recursos, promouent la sostenibilitat en entorns urbans i rurals.
- **ODS 12 - Producció i consum responsables:** El sistema optimitza l'ús de recursos com l'aigua, fomentant una gestió responsable dels cultius.
- **ODS 13 - Acció pel clima:** La reducció de l'ús excessiu d'aigua i l'optimització del reg ajuda a mitigar l'impacte del canvi climàtic a través de pràctiques agrícoles més sostenibles.
- **ODS 15 - Vida d'ecosistemes terrestres:** L'eficiència en el reg ajuda a mantenir ecosistemes saludables i a evitar la degradació del sòl.