Lluís Gallart Cid

Gestió Autònoma sobre el Regadiu d'una Plantació

TREBALL DE FI DE GRAU

dirigit pel Prof. David Gámez Alari

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2025

Resum.

Aquest projecte té com a objectiu el desenvolupament d'un sistema de reg intel·ligent que combini un sistema central amb diversos dispositius interconnectats mitjançant una xarxa. La finalitat principal és optimitzar l'ús de l'aigua en entorns agrícoles, activant el reg de manera automàtica en funció de les condicions ambientals i de les prediccions meteorològiques.

Cada dispositiu de la xarxa està equipat amb sensors que mesuren diversos paràmetres, com ara la temperatura, la humitat del sòl, la humitat ambiental i la lluminositat. Aquests dispositius transmeten la informació a través de la xarxa fins al node arrel, que actua com a pont amb el sistema central. Aquest rep les dades dels dispositius i consulta dues fonts externes d'informació meteorològica per obtenir les prediccions del temps. A més, incorpora un algorisme que avalua la fiabilitat de cada font en funció de l'experiència acumulada i de la precisió històrica. Amb tota aquesta informació, el sistema central determina automàticament si cal activar el reg.

Les tasques desenvolupades inclouen el disseny i la configuració d'una xarxa de comunicació entre diversos dispositius de recollida de dades ambientals, la integració dels sensors, el desenvolupament d'una interfície gràfica per a l'usuari i la creació d'algoritmes per a la presa de decisions. També s'ha garantit que el sistema sigui fiable i permeti la consulta de dades en temps real.

Els resultats obtinguts demostren que el sistema funciona correctament en condicions reals, permetent un control eficient del reg a partir de dades verificades. La solució presentada constitueix una proposta viable i accessible per millorar la sostenibilitat i l'eficiència en sistemes de reg automatitzats.

Resumen.

Este proyecto tiene como objetivo el desarrollo de un sistema de riego inteligente que combine un sistema central con varios dispositivos interconectados mediante una red. El propósito principal es optimizar el uso del agua en entornos agrícolas, activando el riego de manera automática en función de las condiciones ambientales y de las predicciones meteorológicas.

Cada dispositivo de la red está equipado con sensores que miden diversos parámetros, como la temperatura, la humedad del suelo, la humedad ambiental y la luminosidad. Estos dispositivos transmiten la información a través de la red hasta el nodo raíz, que actúa como puente con el sistema central. Este último recibe los datos de los dispositivos y consulta dos fuentes externas de información meteorológica para obtener las predicciones del

tiempo. Además, incorpora un algoritmo que evalúa la fiabilidad de cada fuente en función de la experiencia acumulada y de la precisión histórica. Con toda esta información, el sistema central determina automáticamente si es necesario activar el riego.

Las tareas desarrolladas incluyen el diseño y la configuración de una red de comunicación entre varios dispositivos de recogida de datos ambientales, la integración de los sensores, el desarrollo de una interfaz gráfica para el usuario y la creación de algoritmos para la toma de decisiones. También se ha garantizado que el sistema sea fiable y permita la consulta de datos en tiempo real.

Los resultados obtenidos demuestran que el sistema funciona correctamente en condiciones reales, permitiendo un control eficiente del riego a partir de datos verificados. La solución presentada constituye una propuesta viable y accesible para mejorar la sostenibilidad y la eficiencia en sistemas de riego automatizados.

Abstract.

This project aims to develop a smart irrigation system that combines a central system with several devices interconnected through a network. The main goal is to optimize the use of water in agricultural environments, activating irrigation automatically according to environmental conditions and weather forecasts.

Each device in the network is equipped with sensors that measure parameters such as temperature, soil moisture, ambient humidity and luminosity. These devices transmit the information through a network to the root node, which acts as a bridge with the central system. The central system receives the data from the devices and consults two external sources of meteorological information to obtain weather forecasts. On the other hand, an algorithm evaluates the reliability of each external source, based on the accumulated experience of the system with respect to the accuracy of each source, the central system processes all the data to determine the irrigation status.

The tasks implemented include the design and configuration of a communication network between several environmental data collection devices, the integration of sensors, the development of a graphical user interface and the creation of algorithms for decision-making. In addition, it

guarantees that the system is reliable and that it allows the consultation of data in real time.

The results obtained demonstrate that the system works correctly in real conditions, allowing efficient irrigation control based on the data obtained and contrasted. The solution presented represents a viable and accessible proposal to improve sustainability and efficiency in automatic irrigation systems.

Llista d’abreviatures

ADC: Analog-to-Digital Convert.

API: Application Programming Interface.

GND: Ground.

GPIO: General Purpose Input/Outpu.

IDE: Integrated Development Environment.

IoT: Internet of Things

JSON: JavaScript Object Notation.

LDR: Light Dependent Resistor.

SoC: System on Chip.

TCP/IP: Transmission Control Protocol/Internet Protocol.

Wi-Fi: Wireless Fidelity.

Índex

Índex de taules

Índex de figures

Índex de codi

Índex de formules

1 INTRODUCCIÓ

1.1 Descripció General del Projecte

L'agricultura ha estat sempre una activitat essencial per a la humanitat, tant per garantir l'alimentació com per al desenvolupament econòmic de moltes regions. En l'actual context de canvi climàtic i escassetat d'aigua, aquesta activitat es veu especialment condicionada per la disponibilitat i gestió eficient de l'aigua. Davant d'aquest repte, es fa cada vegada més necessari trobar solucions que permetin fer un ús més eficient i responsable d'aquest recurs, especialment en els sistemes de reg. A més, tenint en compte la necessitat de reduir l'esforç manual i millorar l'optimització, automatitzar aquests processos esdevé un objectiu clau per al sector agrícola.

Aquest treball té com a objectiu el disseny i implementació d'un sistema de reg intel·ligent basat en tecnologia ***IoT***, que hem anomenat **AutoReg**. El sistema es fonamenta en l'ús de dispositius interconnectats mitjançant una xarxa . Cadascun d'aquests dispositius incorpora sensors ambientals i del sòl, capaços de mesurar paràmetres com la temperatura, la humitat del sòl, la humitat ambiental i la lluminositat. Les dades recollides es transmeten a la xarxa fins arriba a node arrel, que s'encarrega d'agrupar la informació i enviar-la a un sistema central, aquest ha de tenir més capacitat d'emmagatzematge i computació que un dispositiu encastat.

Aquest sistema central és l'encarregat de processar les dades i prendre decisions sobre l'activació del reg, segons unes condicions configurades prèviament per l'usuari. També disposa de connexió a Internet, fet que li permet consultar les prediccions meteorològiques. A més, incorpora un algoritme capaç d'analitzar en temps real la fiabilitat de les fonts consultades, a partir de l'avaluació de les seves taxes d'encerts i errors, per tal de determinar si cal tenir-les en compte durant la presa de decisions. Aquest aspecte és especialment rellevant quan es preveu pluja, ja que aquest factor pot condicionar la necessitat de regar o no.

En conjunt, **AutoReg** representa una proposta pràctica, escalable i adaptable a diferents entorns agrícoles, amb l'objectiu d'optimitzar el consum d'aigua, reduir el malbaratament i contribuir a una agricultura més sostenible i intel·ligent.

1.2 Reptes principals del projecte

*1.2.1 **Recol·lecció de dades***

Un dels principals reptes d'aquest projecte és la capacitat de captar i recollir tot tipus de dades ambientals, amb l'objectiu d'analitzar-les posteriorment.

És imprescindible garantir una recollida eficient i constant d'aquestes dades, ja que constitueixen l'eix central sobre el qual es construeix tot el sistema. Gràcies a aquesta informació, és possible entendre l'estat de l'entorn i prendre decisions adequades en funció de les condicions detectades.

El sistema ha de ser capaç de captar diversos tipus de dades ambientals, la qual cosa suposa un nou repte: cal assegurar que cada component mesura correctament els paràmetres assignats i que està integrat de manera eficient dins de l'arquitectura del sistema. A més, la recollida de dades s'ha de dur a terme en temps real per garantir la màxima precisió i capacitat de resposta.

D'altra banda, és necessari disposar d'un dispositiu que actuï com a element de connexió entre els diferents sistemes de captació de dades i el sistema central.

*1.2.2 **Sistema per analitzar les dades***

Després de la recollida de dades ambientals, es presenta un nou repte de gran rellevància: el tractament i l'anàlisi d'aquesta informació. És fonamental que les dades obtingudes estiguin centralitzades, processades i interpretades de manera coherent i automatitzada.

En aquest context, és necessari desenvolupar un sistema central amb capacitat de gestionar un gran volum d'informació i, alhora, per prendre decisions intel·ligents basades en les dades recollides.

*1.2.3 **Comunicació entre dispositius***

Un dels reptes fonamentals d'aquest projecte és garantir la comunicació eficient entre diversos dispositius, ja que es preveu la presència simultània de múltiples sistemes de recollida de dades. Per aquest motiu, és necessari implementar un sistema que permeti la comunicació entre aquests dispositius i que, alhora, sigui escalable. Cada plantació presenta unes característiques i dimensions diferents, fet que implica que, segons el cas, caldrà incorporar un nombre variable de dispositius de captació de dades ambientals.

El projecte no pot basar-se en un únic tipus de comunicació. En primer lloc, cal establir un canal de comunicació entre els dispositius encarregats de recollir la informació. Un cop recollides, aquestes dades han de ser transmeses al sistema central, tal com es descriu a l'apartat **1.2.2**. Per tant, és imprescindible desenvolupar un sistema de comunicació que actuï com a pont entre els dispositius distribuïts al camp i el nucli central encarregat del processament i la presa de decisions.

*1.2.4 **Prediccions meteorològiques***

Una altra dificultat important del projecte és la gestió de les prediccions meteorològiques. Dins del sistema central s'ha d'integrar un mecanisme capaç d'obtenir dades actualitzades sobre el temps provinents de fonts externes. Aquestes prediccions són molt útils per anticipar situacions de pluja i, per tant, evitar el reg quan no sigui necessari. Si es preveu pluja l'endemà, es pot suspendre el reg de manera preventiva i, així, estalviar aigua.

Ara bé, no només cal obtenir aquestes dades, sinó assegurar que siguin fiables. Les previsions del temps no són sempre encertades, i per això convé disposar d'un sistema que tingui en compte la precisió de les fonts consultades. Es proposa, per exemple, fer servir informació comparativa o experiències passades per valorar la qualitat i la confiança de cada predicció.

2 Objectius del Projecte

2.1 Objectius Generals

L'objectiu general d'aquest projecte es el desenvolupament d'un sistema de reg intel·ligent i autònom, que permeti millorar la gestió de l'aigua en entorns agrícoles optimitzant l'ús. Es tracta d'un sistema capaç de recollir informació de l'entorn i actuar de manera automàtica, tenint en compte les condicions ambientals i altres factors que poden influir en el reg.

D'altra banda, un altre objectiu es aconseguir que el sistema funcioni de forma autònoma, requerint una intervenció mínima per part de l'usuari. Aquest només hauria d'introduir uns paràmetres inicials i, a partir d'aquests, el sistema ha de ser capaç d'analitzar l'entorn i decidir quan cal activar el reg.

2.2 Objectius Específics

*2.2.1 **Recollida fiable de les dades ambientals***

Un dels objectius específics mes essencials es una recollida precisa i continua de les dades ambientals per a la presa de decisions del reg. Aquestes dades han de incloure tant la temperatura, humitat ambiental, humitat del sòl i lluminositat. Per fer aquesta recollida es necessitarà un dispositiu capaç de contenir diversos sensors que puguin obtenir totes les dades ambientals necessàries. Per altra banda, aquest dispositiu ha de poder enviar aquestes dades a un sistema que les analitzi per a la presa de decisions del reg.

*2.2.2 **Sistema central per al anàlisis de dades autònoma***

Un cop assolit l'objectiu de la recollida de dades ambientals, és fa necessari disposar d'un sistema capaç de donar significat a aquesta informació i actuar en conseqüència. L'objectiu és desenvolupar un sistema central que pugui gestionar grans volums de dades i interpretar-les de manera autònoma tenint en compte les condicions ambientals del moment, sense necessitat d'intervenció per part de l'usuari final.

*2.2.3 **Establiment de les comunicacions***

Tal com està plantejat, requereix diversos dispositius capaços de recollir dades de l'entorn de manera coordinada. Per aquest motiu, un dels aspectes clau del projecte és la implementació d'un sistema de comunicacions que permeti la connexió eficient entre tots aquests dispositius.

Aquesta comunicació ha de complir una sèrie de requisits: ha de transmetre la informació en temps real, ha de ser estable i escalable. Això significa que ha de poder adaptar-se fàcilment al nombre de dispositius que calgui incorporar, en funció de les dimensions i les necessitats específiques de cada entorn.

A més, cal implementar un sistema de comunicació específic que permeti als dispositius enviar les dades recollides cap al sistema central, que serà l'encarregat d'analitzar tota la informació i prendre les decisions corresponents.

2.2.4 Integració de les prediccions meteorològiques

Un altre aspecte important és la capacitat d'incorporar informació externa, com les prediccions meteorològiques. Aquestes són un element clau per a la presa de decisions automàtiques, ja que permeten al sistema anticipar-se a situacions com la pluja, evitant regs innecessaris i optimitzant l'ús dels recursos hídrics. Per assolir aquest objectiu, cal garantir que les dades meteorològiques es consultin de forma automàtica i contínua.

D'altra banda, és necessari implementar un algoritme capaç d'avaluar la fiabilitat de la informació rebuda. En cas que les prediccions presentin un elevat marge d'error i el sistema no activi el reg quan caldria, es podrien generar pèrdues significatives en la plantació.

.

2.2.5 Facilitar l'ús del sistema

Finalment, cal garantir que el sistema sigui fàcil d'utilitzar per qualsevol persona, independentment del seu nivell de coneixement tècnic. Per aconseguir-ho, és necessari implementar una interfície funcional, intuïtu, visual i accessible, que permeti entendre el funcionament del sistema i intervenir-hi en cas que sigui necessari.

Aquest aspecte és essencial per assegurar la utilitat pràctica de la solució. Un sistema pot ser molt eficient des del punt de vista tècnic, però si resulta complicat d'utilitzar, perd gran part del seu valor per als usuaris finals.

3 Requisits

3.1 Requisits funcionals

3.1.1 Recollida fiable de les dades ambientals

Per a la recollida de dades ambientals, s'ha establert com a requisit la utilització d'un dispositiu anomenat ***ESP32***[1], que s'explicarà amb més detall a l'apartat corresponent **(5.2.1.4)**. Al llarg del projecte, aquests dispositius es referiran com a nodes encastats.

Després d'una anàlisi previ, s'ha determinat que les dades més rellevants per al control del reg són la temperatura, la humitat del sòl, la humitat ambiental i la lluminositat. Un dels criteris principals del projecte és desenvolupar un programa específic per als nodes encastats, que permeti obtenir les dades de tots els sensors integrats

3.1.2 Sistema central per al anàlisis de dades autònoma

Després de la recollida de totes les dades ambientals provinents dels diferents nodes encastats, es fa necessari implementar un sistema central amb capacitat per gestionar un gran volum d'informació i analitzar-la de manera centralitzada i automatitzada. No només cal disposar d'un sistema que rebi totes les dades, sinó que aquest també ha de ser capaç de processar-les i interpretar-les de forma coherent, sense requerir la intervenció de l'usuari.

Un cop el sistema central ha rebut i analitzat les dades, ha de ser capaç d'assumir la responsabilitat de prendre decisions sobre el reg, tenint en compte les condicions ambientals del moment.

*3.1.3 **Establiment de les comunicacions***

Com s'ha indicat en els objectius amb la comunicació, un dels requisits essencials del projecte és la creació de dos sistemes de comunicació diferenciats.

El primer sistema té com a finalitat la interconnexió de tots els nodes encastats. Per aconseguir-ho, s'ha optat per una xarxa de tipus **Mesh**, que permet compartir informació entre diversos nodes de manera eficient i robusta.

A més, aquesta xarxa ofereix un alt grau d'escalabilitat, ja que pot adaptar-se fàcilment a les dimensions i necessitats de cada plantació. Aquest tipus d'arquitectura facilita la connexió i desconnexió de nodes encastats sense requerir configuracions addicionals, fet que contribueix a la flexibilitat del sistema.

El segon sistema de comunicació s'encarrega d'establir un pont entre la xarxa **Mesh** i el sistema central. Per fer-ho possible, s'utilitza un node encastat que actua com a node arrel. Aquest node té la funció de connectar-se simultàniament a la xarxa **Mesh**, on rep les dades dels altres nodes, i a la xarxa local del sistema central mitjançant un punt d'accés creat expressament.

El node arrel és responsable de recollir totes les dades transmeses pels nodes fills, és a dir, els nodes encastats que formen part de la xarxa **Mesh** i que envien les dades provinents dels sensors. Posteriorment, aquestes dades es transmeten al sistema central, on seran analitzades per tal de prendre decisions automatitzades sobre el reg.

*3.1.4 **Integració de les prediccions meteorològiques***

El sistema central ha de ser capaç d'integrar informació procedent de fonts externes per tal de millorar la presa de decisions de manera automàtica. En aquest projecte s'han utilitzat dues ***APIs*** meteorològiques (***OpenWeather***[2] i ***WeatherAPI***[3]) per obtenir les prediccions, amb especial atenció a la probabilitat de pluja.

Les dades d'aquestes fonts s'han d'actualitzar de manera periòdica, i cal validar la fiabilitat per garantir que les decisions es prenguin amb criteris fiables. Per dur a terme aquesta validació, s'utilitza un algoritme d'aprenentatge basat en taxes d'encerts. El sistema aprèn a partir de l'experiència de la següent manera: primer, s'extreuen les dades de predicció; l'endemà, el sistema comprova si la predicció ha estat encertada. En funció del resultat, s'assigna una puntuació positiva o negativa a cada font. Aquesta puntuació es tradueix en una taxa d'encerts, que representa el grau de fiabilitat atribuït a cada una de les ***APIs***.

*3.1.5 **Facilitar l'ús del sistema***

El sistema ha d'incloure una interfície gràfica clara i intuïtiva, que permeti una interacció senzilla amb l'usuari. Aquesta ha d'oferir la possibilitat de configurar els valors del llindar per a cadascun dels paràmetres monitoritzats pels sensors. Igualment, ha de permetre definir la ciutat, amb l'objectiu d'adaptar les consultes meteorològiques a l'entorn concret on s'instal·li el sistema.

A més, la interfície ha d'incloure una secció dedicada a la visualització de tots els nodes connectats, juntament amb els valors actuals registrats per cada sensor. Aquesta funcionalitat resulta especialment útil per identificar possibles anomalies, com ara un mal funcionament d'algun node, i per conèixer l'origen específic de les dades obtingudes.

Finalment, es considera essencial disposar d'una pàgina principal visualment clara i atractiva, que permeti conèixer de forma immediata si el sistema ha activat el reg.

3.2 Requisits no funcionals

*3.2.5 **Eficiència***

El sistema ha de poder transmetre i processar les dades del sensor en temps real, amb un retard màxim de 3 segons entre la captura de les dades i la seva visualització.

La xarxa **Mesh** ha de permetre la connexió de múltiples nodes encastats sense afectar al rendiment.

L'algoritme sobre la decisió del reg ha de prendre una decisió en menys d'un segon després d'analitzar les dades recollides.

*3.2.6 **Escalabilitat***

La xarxa ***Mesh*** ha de ser capaç de poder incorporar nous nodes encastats sense la necessitat de tornar a configurar tot el sistema.

*3.2.7 **Fiabilitat***

El sistema ha de poder enviar la informació de forma continua fins i tot si es desconnecta algun node, per això la xarxa ***Mesh*** ha de ser capaç de resoldre Aquest problema sense inconvenients.

*3.2.8 **Disponibilitat***

El sistema ha d'estar operatiu les 24 hores del dia. En cas de pèrdua de senyal de qualsevol node el sistema ha de seguir funcionant sense cap problema.

*3.2.9 **Usabilitat***

La interfície gràfica del usuari ha de ser intuïtiva i fàcil d'utilitzar, fins i tot per a persones sense cap coneixement tècnic.

Els paràmetres de configuració han de ser intuïtius i en cas de fallada s'ha d'explicar de forma senzilla quina dada ha introduït de forma incorrecta.

*3.2.10 **Compatibilitat***

El sistema he de poder ser compatibles en varis models ESP32, sol hem de tenir en compte que aquest nou dispositiu sigui compatible amb les xarxes ***ESP_MESH***.

L'aplicació ha de ser compatible en varis sistemes operatius.

4 Anàlisis dels requisits funcionals

4.1 Diagrama de classes

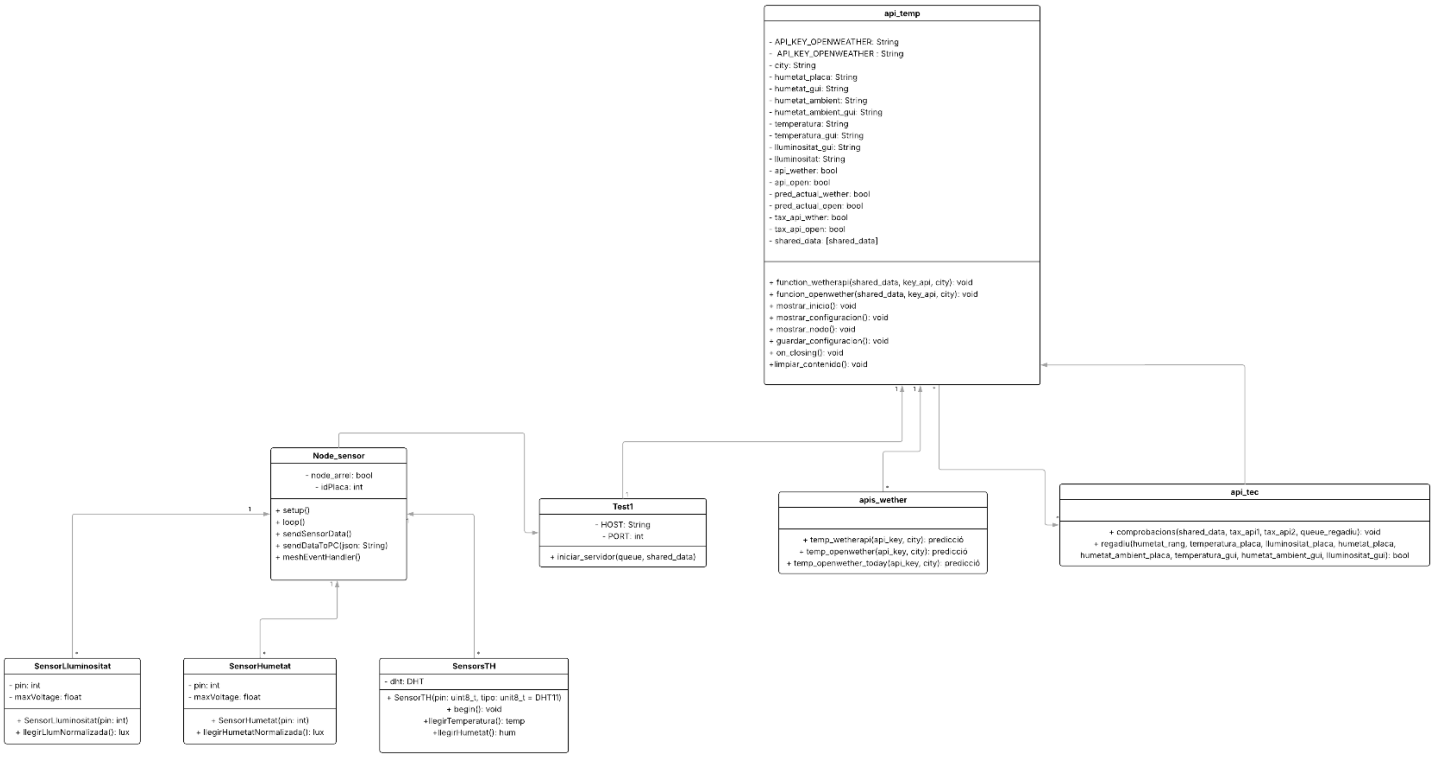

Figura 1. Diagrama de classes

Per a la creació d'aquest diagrama de classe s'ha utilitzat l'aplicació web ***LucidChart*** *[4]*.

4.1.1 Explicació general del diagrama de classes

Aquest diagrama es divideix en dues àrees funcionals:

El sistema dels nodes ESP32: es basa en un estructura modular formada per diverses subclasses (***SensorLluminositat***, ***SensorHumitat*** i ***SensorTH***). Cadascuna d'aquestes té la funció de capturar les dades provinents dels sensors corresponents i transformar-les en valors entenedors per a l'usuari final.
Totes aquestes subclasses s'integren dins d'una classe principal anomenada node sensor, encarregada d'agrupar i estructurar la informació recollida. Aquesta classe genera un fitxer en format JSON amb els valors obtinguts i segons el tipus de node, actua de la manera següent:

- Si es tracta d'un node fill, envia el fitxer ***JSON*** [5]generat a través de la xarxa ***Mesh***.
- Si és el node arrel, envia tant el seu propi fitxer ***JSON*** [5]com els rebuts dels altres nodes a l'ordinador central que s'encarrega del seu procesament.

El sistema centra es la part del sistema que es basa en una arquitectura modular formada per una classe principal i diverses subclasses, cadascuna amb funcions específiques relacionades amb la lògica de decisió i la gestió de dades meteorològiques mitjançant ***APIs*** externes:

- Classe principal:
  - **Api_temp**: És l'aplicació principal on esta la interfície gràfica i la crida als subfitxers.
- Subclasses:
  - **ConnexioNodes**: s'encarrega d'establir el punt d'accés Wi-Fi al qual es connecta el node arrel. Aquesta classe rep els fitxers en format ***JSON*** [5]generats pels nodes i els transmet a la classe principal per al seu posterior processament.
  - **Api_Weather**: té com a funció realitzar consultes a dues ***APIs*** meteorològiques per obtenir la previsió del temps del dia en curs i del següent. Aquestes dades són essencials per anticipar condicions que poden afectar el reg.
  - **Api_Tec**: implementa l'algoritme de decisió. Aquesta classe rep com a paràmetres les dades ambientals obtingudes pels sensors, les prediccions meteorològiques i les respectives taxes d'encert de les fonts consultades. Amb tota aquesta informació, determina automàticament si cal activar el sistema de reg.

4.2 Casos d'ús

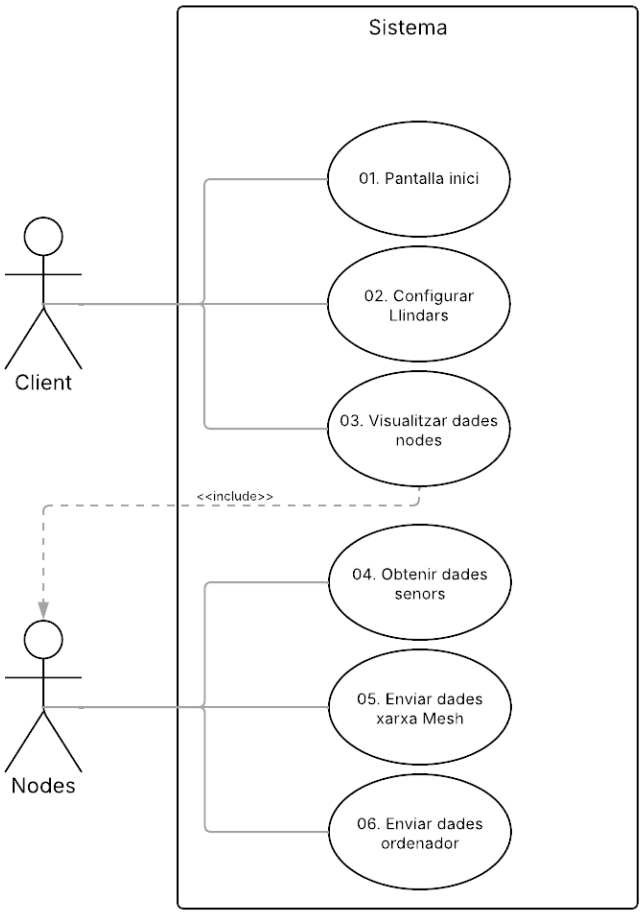

Figura 2. Diagrama de caos d'us

Per a la creació d'aquest diagrama de casos d'us s'ha utilitzat l'aplicació web ***LucidChart****[4]*.

4.3.1 Cdu 1 Pantalla Inici

Resum de la funcionalitat: Mostrar el logotip de l'aplicació i poder veure el resultat del reg.

Paràmetres d'entrada: Cap.

Paràmetres de sortida: Cap.

Actors: Client.

Precondició: Cap.

Postcondició: Cap.

Procés normal principal:

1. En el moment d'obrir l'aplicació, es mostra una finestra principal amb el logotip en una mida destacada. A continuació, de manera gràfica i clara, s'hi visualitza la decisió automàtica del sistema pel que fa a l'activació o no del reg.

2. L'usuari disposa d'un menú de navegació que dona accés a la resta de seccions de l'aplicació.

4.3.2 Cdu 2 Configuració llindars.

Resum de la funcionalitat: Configuració del client per a la presa de decisions.

Paràmetres d'entrada: Temperatura, lluminositat, humitat ambient, humitat del sòl.

Paràmetres de sortida: Cap.

Actors: Client.

Precondició: Cap.

Postcondició: Cap.

Procés normal principal:

1. L'aplicació mostrarà una pàgina de configuració.

2. El client podrà introduir els llindars que siguin convenients per a la seva funció.

3. D'altra banda, l'usuari ha d'introduir la ciutat on es troba ubicada la plantació.

4. Finalment, la pàgina incorpora un botó per desar la configuració. En cas que es detecti algun error durant el procés, l'aplicació mostra un missatge clar i immediat per tal d'informar l'usuari i facilitar-ne la correcció.

4.3.3 Cdu 3 Visualitzar dades node

Resum de la funcionalitat: És mostra en temps real les dades dels sensors de cada node.

Paràmetres d'entrada: Cap.

Paràmetres de sortida: Nombre de nodes, id_node i valors dels sensors de cada node.

Actors: Client.

Precondició: Els nodes han d'estar actius i connectats a l'ordinador.

Postcondició: Cap.

Procés normal principal:

1. L'aplicació mostra una llista de nodes actius, on s'indiquen tant la identificació de cada dispositiu com els valors que està recollint en temps real.

2. L'usuari podrà consultar en temps real totes les dades recollides pels dispositius ***ESP32*** [1].

*4.3.4 **Cdu 4 Obtenir dades sensor***

Resum de la funcionalitat: Els dispositius ***ESP32*** [1]recullen les dades dels seus propis sensors.

Paràmetres d'entrada: Sensors de la placa.

Paràmetres de sortida: Temperatura, lluminositat, humitat ambient, humitat del sòl.

Actors: Node.

Precondició: El node ha de estar connectat.

Postcondició: Cap.

Procés normal principal:

1. El node ha d'estar correctament connectat per tal de poder activar els sensors.
2. Un cop activats, els sensors recullen la informació i la transmeten en forma de senyals elèctriques.
3. Els sensors recullen la informació i l'envien en forma de senyals.
4. Aquestes senyals són interpretades pel dispositiu, que les transforma en valors comprensius per a l'usuari final.

*4.3.5 **Cdu 5 Enviar dades xarxa Mesh***

Resum de la funcionalitat: Un cop recollida tota la informació, els nodes han de ser capaços de transformar les dades obtingudes en un fitxer en format ***JSON****[5]*. Posteriorment, el fitxer es distribueix a través de la xarxa ***Mesh***, assegurant-ne la transmissió al node arrel o al sistema central, segons correspongui.

Paràmetres d'entrada: Red mesh.

Paràmetres de sortida: Fitxer ***JSON****[5]*.

Actors: Node.

Precondició: Ha d'estar habilitada la xarxa ***Mesh*** per poder enviar la informació.

Postcondició: El node ha de poder recollir tota la informació dels sensors.

Procés normal principal:

1. El dispositiu ha de generar el seu propi fitxer ***JSON*** [5]amb les dades recollides.

2. Aquest dispositiu es connectarà a una xarxa ***Mesh*** prèviament configurada.

3. Un cop establerta la connexió correctament, enviarà les dades a través de la xarxa fins que arribin al node arrel, que s'encarregarà de centralitzar la informació.

*4.3.6 **Cdu 6 Enviar dades ordenador***

Resum de la funcionalitat: El node arrel rep tant les seves pròpies dades com les que li arriben dels altres nodes de la xarxa. Un cop recopilada tota la informació, es responsable d'enviar-la a l'ordinador central, on serà processada de manera centralitzada i automàtica.

Paràmetres d'entrada: Red ***Mesh***.

Paràmetres de sortida: Punt accés ordinador.

Actors: Node.

Precondició: Ha d'estar habilitada la xarxa ***Mesh*** per poder enviar la informació.

Postcondició: El punt d'accés de l'ordinador ha d'estar operatiu per garantir la recepció correcta de les dades.

Procés normal principal:

1. El node arrel observa dins de la xarxa ***Mesh*** i rep la informació.

2. El node arrel no enviarà les seves dades dins d'aquesta xarxa.

3. En Aquest moment intenta connectar-se a la xarxa de l'ordinador.

4. Un cop establerta la connexió, procedeix a enviar totes les dades disponibles, tant les seves pròpies com les rebudes dels altres nodes.

5 Disseny dels nodes encastats

Els nodes encastats constitueixen una part essencial del sistema, que capten les dades sòl i del ambient que permeten prendre decisions intel·ligents sobre el reg. Aquest sistema de nodes encastats ha estat dissenyat per garantir la fiabilitat, l'escalabilitat i l'eficiència de l'ús de l'aigua, utilitzant una comunicació fluida amb la resta de la xarxa ***Mesh***.

Aquest disseny a tingut en compte diversos factors claus: la selecció i integració dels sensors, la distribució física del maquinari, la configuració dels pins i la seva capacitat per el enviament de dades a una xarxa ***Mesh***.

Cada node està format per un conjunt de sensors que mesuren quatre tipus de dades: humitat del sòl, humitat ambiental, lluminositat i temperatura.

Aquest conjunt de nodes ha estat desenvolupat dins del projecte amb l'objectiu de complir tant criteris tècnics com pràctics, assegurant una captació eficient de dades i una transmissió fiable, que són elements fonamentals per a l'èxit del sistema de reg automàtic.

5.1 Arquitectura dels nodes encastats

5.1.1 Llenguatges utilitzats

Per a la implementació del node, s'ha dut a terme un estudi per determinar quin és el llenguatge de programació més adequat per adaptar-se a les necessitats del desenvolupament. Paral·lelament, també s'han analitzat els entorns de desenvolupament més òptims per dur a terme la programació de manera eficient i estructurada.

5.1.1.5 C++

En la implementació del node, es va realitzar un estudi per determinar quin seria el llenguatge de programació més adequat, considerant diverses opcions com Assembly, CircuitPython i C++[6]. Després d'un procés de recerca i comparació, es va concloure que el llenguatge més utilitzat en el disseny de sistemes amb microcontroladors és el C++[6].

Aquest llenguatge presenta diversos avantatges que s'ajusten a les necessitats específiques del projecte. Es destaca, principalment, la seva orientació a objectes, que facilita la definició de classes i contribueix a una millor modularitat i reutilització del codi. Aquest enfocament resulta especialment pertinent en aquest cas, ja que es requereixen diverses classes per a la captació de dades, les quals poden ser reutilitzades en diferents parts del programa principal.

A més, el llenguatge C++[6] ofereix les avantatges d'accedir a llenguatge de baix nivell sobre el maquinari, cosa que altres llenguatges de més alt nivell no permeten, o bé ho fan amb una eficiència inferior. Aquest característica és especialment rellevant en aquest projecte, ja que cal accedir directament als ***GPIOs*** de ***l'ESP32*** [1] per obtenir les dades dels sensors de manera fiable i controlada.

Un altre aspecte important és que els dispositius encastats tenen recursos limitats, tant en memòria com en capacitat de processament. Per aquest motiu, és fonamental que el codi sigui ràpid i eficient, no només pel que fa a l'ús de recursos, sinó també en termes de consum

energètic. En aquest sentit, C++ [6] representa un punt d'equilibri ideal: no és un llenguatge d'alt nivell que pugui afectar el rendiment, però tampoc és un llenguatge de baix nivell, que tot i ser més eficient, resulta menys pràctic i més complex a l'hora de treballar-hi.

*5.1.2 **Entorn de desenvolupament utilitzat***

El desenvolupament d'aquest projecte combina dispositius encastats amb una aplicació, i per tant, requereix l'ús d'eines adequades i d'entorns de treball que permetin desenvolupar aquesta aplicació de la manera més eficient possible. En aquest cas, és essencial seleccionar entorns que s'adaptin a les necessitats tècniques tant del maquinari com del programari. Així mateix, han de facilitar la integració de llibreries externes, gestió de dependències i la portabilitat entre sistemes.

Per tal de garantir un flux de treball òptim, s'han escollit entorns que permetin treballar còmodament tant en la programació dels nodes ***ESP32****[1]* com en el desenvolupament de l'aplicació destinada a l'ordinador.

5.1.2.5 PlatformIo*[7]*

Quan es va iniciar el projecte, una de les primeres necessitats identificades va ser la d'utilitzar un entorn de treball òptim per poder programar un sistema encastat de manera més còmoda. Per aquest motiu, es va realitzar un petit estudi comparatiu entre diversos entorns de desenvolupament per a dispositius encastats, com ara *Arduino IDE, Visual Studio code[8]* amb *Arduino Extension, Thonny IDE* e.t.c

Després d'aquesta anàlisi, es va optar per utilitzar ***PlatformIO*** , principalment perquè és compatible amb *Visual Studio Code[8],* l'eina principal utilitzada per a tot el desenvolupament del projecte. Aquesta eina ens dona la facilitat de poder estar integrat dins d'aquest entorn de programació.

A més, ens dona la facilitat de gestiona les dependències de forma eficient ja que permet instal·lar, actualitzar i gestionar les llibreries directament des del fitxer "***platform.ini****"*. Aquest sistema agilitza el procés d'importació de llibreries, redueix els errors habituals de compatibilitat i facilita la portabilitat del projecte entre diferents equips amb el mateix entorn configurat.

A diferencia d'altres entorns, aquest ens permet tenir una estructura de fitxers organitzada, amb una separació clara entre el fitxer principal i els fitxers secundaris. Aquest enfocament modular és fonamental per al desenvolupament d'aquest projecte, ja que permet diferenciar clarament el nucli funcional del sistema dels components encarregats de la lectura dels sensors.

Un dels grans avantatges que ofereix aquest entorn, especialment quan s'integra amb *Visual Studio Code[8]* és la possibilitat d'automatitzar processos essencials com la compilació del programa, la càrrega del codi a la placa ***ESP32****[1]* i la visualització en temps real de les dades generades pel node mitjançant la terminal integrada. Aquesta funcionalitat

resulta molt útil durant les fases de desenvolupament i prova, ja que permet controlar amb precisió el comportament dels dispositius.

D'altra banda, es va tenir la necessitat de poder gestionar i visualitzar múltiples entorns de manera simultània. Ja que el sistema es basa en la comunicació entre múltiples plaques ***ESP32*** [1] a través d'una xarxa ***Mesh***, era imprescindible disposar d'un entorn que permetés observar el funcionament de diversos nodes al mateix temps i detectar possibles anomalies o comportaments inesperats.

5.1.2.6 Visual Studio Code

Una de les prioritats era escollir un entorn de treball que permetés gestionar de manera còmoda, ordenada i eficient, un programa amb múltiples fitxers i components, tant per la part de sistemes encastats, com en la part del sistema central. Per aquest motiu, es va estudiar quin entorn complia millor aquests requisits bàsics.

En primer lloc, l'editor de codi és altament extensible permetent la integració d'extensions com ***PlaformIO****[7]* o ***Python****[9]*, cobrint així totes les necessitats del projecte dins d'un únic entorn. Aquesta capacitat d'integració va ser un factor clau en la decisió, ja que l'objectiu era poder treballar amb un sol *IDE* tant per al sistema de nodes encastats com per la part central. A més, ***Visual Studio Code****[8]* ofereix funcionalitats molt importants com pot ser ressaltar la sintaxi, la detecció d'errors, la disponibilitat d'extensions pràctiques i una navegació àgil entre fitxers, aspectes que contribueixen a agilitzar notablement el desenvolupament del projecte.

D'altra banda, es va optar per utilitzar el control de versions amb ***GIT*** per gestionar de manera eficient l'evolució del projecte. Aquesta eina facilita el seguiment dels canvis realitzats i permet recuperar versions anteriors en cas que es produeixi algun error. Per aquest motiu, l'ús d'extensions compatibles amb ***Visual Studio Code****[8]* ha resultat especialment útil, ja que permeten integrar aquest control de versions de forma senzilla i intuïtiva dins de l'entorn de treball.

Un altre aspecte destacable és la terminal integrada, que facilita l'execució de comandes necessàries per compilar el projecte, monitoritzar el port sèrie o gestionar l'execució de l'aplicació desenvolupada en ***Python****[9]*, tot sense sortir de l'editor. Aquesta funcionalitat és especialment útil per observar en temps real el comportament del node ***ESP32****[1]*, així com per provar i ajustar el codi de forma dinàmica i àgil.

Finalment, l'ús d'aquest entorn ha permès treballar de manera transversal en diverses plataformes, obrint el mateix projecte en diferents sistemes operatius sense problemes de compatibilitat. Aquesta característica ha afavorit la portabilitat i flexibilitat del desenvolupament al llarg de tot el projecte.

5.2 Dispositius externs.

Donada la finalitat principal de l'aplicació, capturar les condicions ambientals d'una plantació i prendre decisions automàtiques sobre el reg, ha estat necessari incorporar diversos dispositius externs a l'ordinador. Per aquest projecte s'han utilitzat sensors ambientals i una placa ***ESP32*** [1]que actua com intermediari entre els sensors i el sistema central.

5.2.1 Sensors

Per a la selecció dels sensors s'han tingut en compte diversos criteris. En primer lloc, han de poder donar valors fiables i que estiguin ben calibrats. A més, ja que aquest projecte pretén ser una solució accessible per a entorns agrícoles, els dispositius escollits havien de tenir un cost assequible per evitar un increment excessiu en el cost total del sistema.

D'altra banda, s'ha de tenir en compte una variable important: el dispositiu ***ESP32****[1]*, només pot treballar amb senyals de 3,3 volts. Per aquest motiu, tots els sensors seleccionats són compatibles amb aquest requisit elèctric.

5.2.1.1 Sensor d'humitat del sòl

Després de realitzar un breu estudi sobre diversos sensors d'humitat disponibles al mercat, es va optar pel model ***SparkFun Soil Moisture Sensor****[10]*. Aquest sensor destaca per la seva simplicitat de funcionament i la seva eficàcia en entorns agrícoles.

El dispositiu incorpora dues plaques metàl·liques de grans dimensions que actuen com a sondes del sensor. Aquestes funcionen conjuntament com una resistència variable: com més humitat hi ha al sòl, més gran és la conductivitat entre les plaques, cosa que provoca una disminució de la resistència i, en conseqüència, una senyal de sortida més elevada. Aquesta característica permet una lectura directa i efectiva del nivell d'humitat present al sòl.

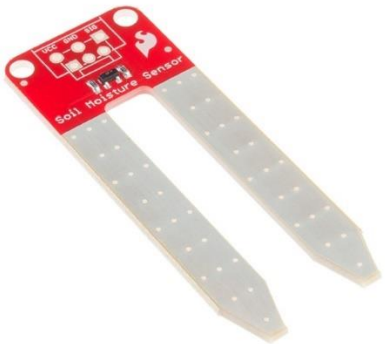

Figura 3. Sensor d'humitat del sòl

5.2.1.2 Sensor de lluminositat

S'ha incorporat la mesura de la quantitat de llum solar que incideix sobre la plantació. Amb aquesta finalitat, s'ha seleccionat un sensor que fos econòmic i capaç de detectar una intensitat mínima de lúmens. El dispositiu escollit ha estat el ***TEMT6000****[11]***.**

Aquest sensor destaca per la seva simplicitat i funcionalitat. El seu funcionament es basa en un transistor de llum: com més intensitat lumínica rep, més gran és el voltatge analògic que genera en el pin de senyal. Aquesta resposta proporcional facilita la integració i el tractament de dades dins del sistema.

Una de les característiques destacables d'aquest dispositiu és el seu rang de detecció, que se situa entre 10 i 1000 lúmens, suficient per monitoritzar les condicions de llum en una explotació agrícola. A més, opera amb tensions d'entre 3 i 5 volts, fet que el fa compatible amb el microcontrolador ESP32, el qual treballa amb un màxim de 3,3 volts.

Figura 4. Sensor lluminositat TEMT600

5.2.1.3 Sensor de temperatura i humitat ambiental.

Una variable molt importat per aquest projecte és la lectura tant de la humitat ambiental com de la temperatura. Per fer-ho, s'ha realitzat un estudi comparatiu de diversos sensors disponibles al mercat, amb l'objectiu de trobar una solució eficient que permetés obtenir dues mesures de manera simultània. El dispositiu escollit va ser el ***DHT11,*** un sensor compacte que integra dues funcionalitats.

Aquest tipus de sensor té tres avantatges principals. En primer lloc, ofereix una lectura amb un nivell de precisió molt bona, aquest fet és especialment important ja que aquesta aplicació intenta assegurar la millor fiabilitat possible. En segon lloc, es tracta d'un sensor que permet mesurar dos valors ambientals al mateix temps (sensor d'humitat ambient i de temperatura), reduint tant la complexitat com l'espai ocupat al maquinari. Finalment, disposa de llibreries que faciliten la seva integració, proporcionant mètodes que retornen la informació en formats comprensibles per a l'usuari final.

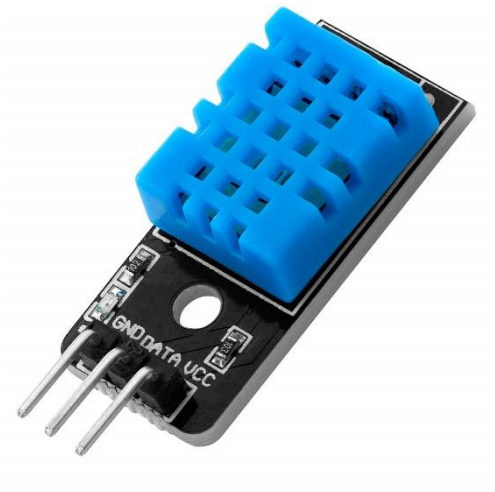

Figura 5. Sensor d'humitat ambiental i temperatura

5.2.1.4 Placa ESP32

Per tal de desenvolupar un sistema de reg automàtic basat en nodes distribuïts, un dels elements clau del projecte és l'elecció del microcontrolador més adient. Després d'analitzar diverses alternatives disponibles al mercat, es va optar per utilitzar la placa ***ESP32***[1], ja que ofereix una combinació òptima entre potència, funcionalitat i cost.

Aquest microcontrolador té un sistema de 32 bits per "***Espressif System"***, que destaca per la seva connectivitat integrada, equipat amb un sistema en chip "***SoC" Wi-Fi*** de 2,4 Ghz, una característica especialment rellevant per aquest projecte, ja que es requeria la creació d'una xarxa ***Mesh*** per a la comunicació entre nodes, així com la transmissió de dades al sistema central.

A més. disposa d'una gran varietat de ports ***GPIO***, juntament amb interfícies analògiques i digitals, que permeten la connexió directa de diversos sensors (de temperatura, humitat del sòl, humitat ambiental i lluminositat) sense moltes complicacions. Això facilita la integració de components externs.

Un altre aspecte destacable és que és placa econòmica i àmpliament suportada per la comunitat, amb gran quantitat d'informació. Aquesta accessibilitat ha facilitat i millorat el desenvolupament i la resolució dels problemes.

L'ús de ***l'ESP32****[1]* ha permès satisfer tots els requisits tècnics del projecte (connectivitat, potencia, integració de sensors i escalabilitat) amb una ferramenta accessible,

potent i ben suportada, el que fa que sigui una molt bona opció per al sistema embegut implementat en aquest projecte.

Figura 6. ESP32-S2

5.3 Disseny de l'estructura del sistema encastat

Un dels aspectes més fonamentals del desenvolupament ha estat el disseny de les connexions entre els sensors i la placa ***ESP32****[1].* Aquesta tasca no implica assignar un pin a cada sensor, sinó entendre les característiques tècniques de cada component, les capacitats d'aquest dispositiu encastat i les implicacions que pot tenir cada decisió amb el rendiment global del sistema.

En aquesta secció es descriu i es justifica com s'han dissenyat les connexions i quins criteris s'han tingut en compte.

5.3.1 Esquema de pins ESP32

L'esquema de connexions de la placa ***ESP32****[1]*, representat en la **Figura [7]**, mostra de manera gràfica la distribució dels pins ***GPIO***, així com les seves funcions alternatives i limitacions. Aquest diagrama ha estat un element fonamental per al disseny correcte del dispositiu, ja que la selecció dels pins no pot fer-se de manera arbitrària, sinó que ha d'estar guiada per criteris tècnics específics.

Un dels principals factors a considerar és el tipus de senyal que emet cada sensor, ja sigui digital o analògica, això determinarà quins pins poden utilitzar-se. Tot i que l'***ESP32****[1]* disposa de molts ***GPIOs***, però no tots adequats per a la tasca a realitzar. Alguns d'aquests pins tenen funcions reservades, d'altres estan compartits amb altres perifèrics interns, i n'hi ha que poden ser vulnerables en certs modes d'arrancada.

Per tal d'evitar disfuncions en el sistema o comportaments no desitjats en el futur, s'ha realitzat una selecció acurada dels pins, prioritzant aquells que ofereixen estabilitat, compatibilitat amb els sensors seleccionats i seguretat durant tot el cicle d'execució del dispositiu.

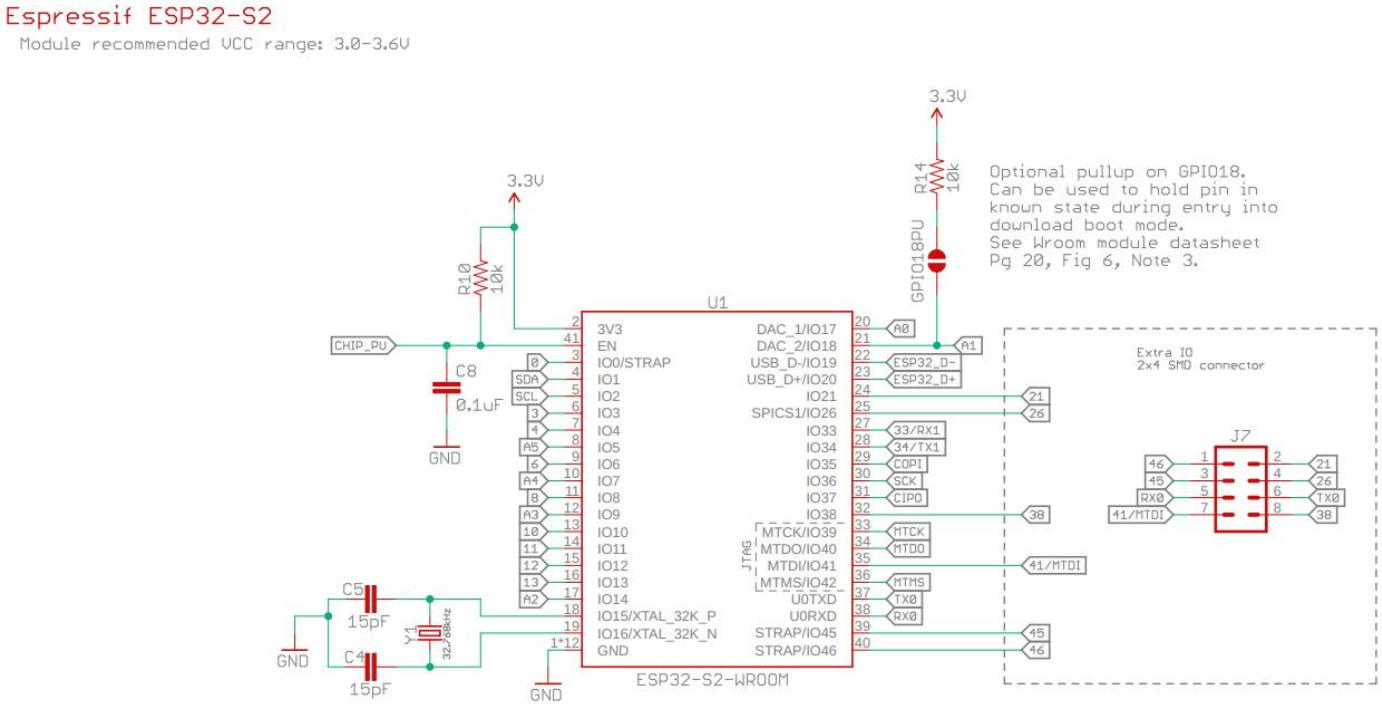

Figura 7. Esquema ESP32-S2

Aquest esquema es pot trobar a la següent referència. [12]

5.3.2 Esquema de pins del sensor d'humitat del sòl

Aquest sensor proporciona una sortida analògica que varia en funció de la quantitat de la quantitat d'aigua present al sòl. Funciona mitjançant una resistència que actua com a divisor de tensió, per la qual cosa és necessari un convertidor analògic a digital (**ADC**) per poder interpretar la lectura. En aquest projecte, s'ha assignat el pin A1 per aquest sensor. aquesta elecció no ha estat aleatòria, no es recomana usar la tecnologia d'**ADC2** si està el ***Wi-Fi*** actiu, ja que pot provocar interferències o errors en la lectura. Per aquest motiu, s'ha evitat connectar aquest sensor a pins que utilitzin **ADC2**.

Tal com es mostra a la **Figura [8],** l'esquema de connexió del sensor és simple, perquè només disposa de tres pins. El primer pin correspon a l'entrada d'alimentació, que en aquest cas és de **3,3 volts**, valor subministrat per la pròpia placa ***ESP32****[1]*. El segon pin es connecta a terra (**GND**) i el tercer és la sortida de senyal, que transmet les dades a la ***ESP32****[1]*.

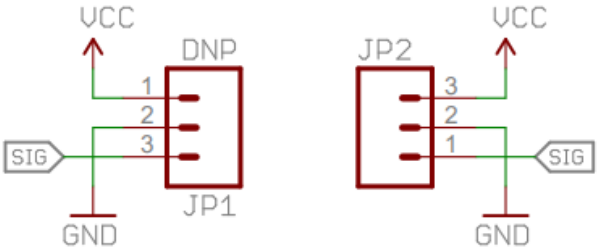

Figura 8. Esquema sensor humitat del sòl

Aquest esquema es pot trobar a la següent referència. [13]

5.3.3 Esquema de pins del sensor de lluminositat

El present sensor, esta basat amb ***LDR,*** també ofereix una sortida analògica proporcional a la quantitat de llum ambiental. Igual que el sensor d'humitat del sòl, requereix un pin amb capacitat ***ADC1***. En aquest cas s'ha assignat el pin A0. Aquesta ubicació assegura una lectura neta i estable que s'ha adaptat en lúmens per a l'adaptació a l'algoritme automàtic del reg.[11]

Figura 9. Esquema sensor lluminositat

5.3.4 Esquema de pins del sensor de temperatura i humitat ambiental

Aquet sensor ofereix 4 pins en concret, un d'aquets pins no te ninguna utilitat ja que no es funcional, per altra banda hi ha els tres pins, el primer es el que rep els volts utilitzats. Aquet es connectarà al dispositiu **ESP32**[1] el **GPIO 2,** per altra banda hi ha el pin 2 que envia la senyal al dispositiu que rep les dades, aquesta senyal, en aquest cas envia tant les dades de temperatura com d'humitat. Es connecta al pin **A2.** Per últim, esta el pin **GND**, aquet es la senyal de sòl.

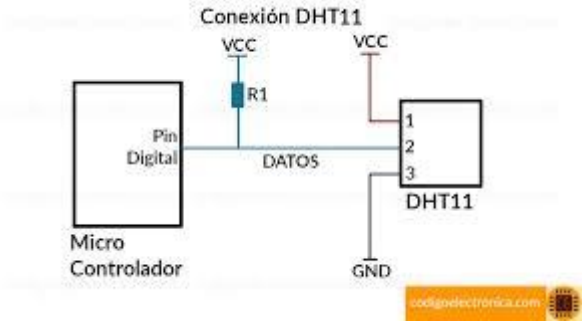

Figura 10. Esquema DHT11

6 Implementació dels nodes encastats

Un cop ja definit tot el disseny estructural i funcional dels nodes encastats, es va procedir a la implementació del codi encarregat de controlar el maquinari i garantir el funcionament correcte del sistema. En aquesta etapa es veurà la transformació de la planificació teòrica fins a l'execució real.

En dit procés s'ha implementat un codi centrat en l'optimització i en una estructura modular. Tal com s'il·lustra a la **Figura [9]**, es mostra l'organització dels fitxers del projecte. D'una banda, s'identifica el fitxer **"main.cpp",** que conté el codi principal i és responsable de l'execució global del sistema. Aquest fitxer també s'encarrega de cridar les funcions auxiliars definides en altres arxius. D'altra banda, es destaca una carpeta anomenada **sensors,** on es troben els fitxers corresponents a cada sensor. Cadascun d'aquests arxius gestiona la lectura de dades d'un sensor concret i en transforma la senyal en valors comprensibles per a l'usuari final.

Cal remarcar que no es mostrarà la totalitat del codi font desenvolupat, sinó que s'han seleccionat i explicat les parts més rellevants del programa, especialment aquelles que constitueixen els aspectes clau del projecte. Cada fragment presentat inclou una explicació de les decisions tècniques preses en la seva implementació.

Paral·lelament, tal com s'ha exposat a l'apartat dedicat al **Disseny dels nodes encastats,** tot el codi presentat ha estat desenvolupat en llenguatge C++, mitjançant els entorns de programació **PlatformIO**[7] i **Visual Studio Code**[8].

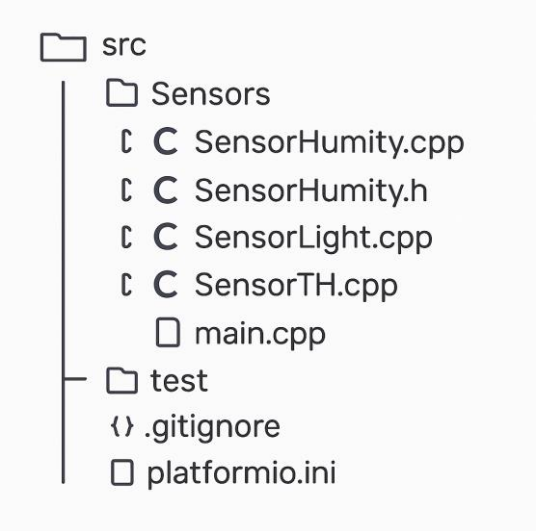

Figura 11. Esquema node encastat

6.1 Implementació programa principal

Per facilitar la comprensió del funcionament intern del sistema, en el següent apartat s'exposen diverses seccions representatives del codi font del programa principal. L'objectiu és il·lustrar, de manera estructurada i entenedora, com s'han dissenyat els nodes encastats i quina lògica de programació s'hi ha aplicat.

En primer lloc, es mostren les variables globals i les constants del sistema. S'hi poden identificar els pins assignats a cada sensor, així com l'identificador de node. Aquest identificador és fonamental per a la posterior associació de les dades amb el node corresponent dins del sistema central, fet que permet representar-ne l'origen a la interfície gràfica de manera clara.

També s'inclou una constant anomenada **MESH_ID** definida com un vector de 6 bytes, que conté l'identificador de la xarxa **Mesh**. Per tal que un node pugui formar part d'aquesta xarxa, ha de fer ús d'aquest mateix identificador.

D'altra banda, la constant **node_arrel** s'utilitza per diferenciar els nodes fills del node arrel dins de l'arquitectura de comunicació. Finalment, es defineixen diverses constants addicionals destinades a la configuració i connexió amb el sistema central, les quals es detallaran més endavant.

```
// Constants sobre els pins de cada sensor
#define SENSOR_LUZ_PIN A0
#define SENSOR_HUMEDAD_PIN A1
#define SENSOR_TH_PIN 14

#define ID_PLACA 2
//Identifiacdor de la xarxa mesh
const uint8_t MESH_ID[6] = {0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC};
// Configuració de la xarxa mesh
#define MESH_CHANNEL 6
#define MESH_MAX_LAYER 10
// Configuracio de punt d'acces per enviar la informacio al portatil
const char* WIFI_SSID = "Lluis";               // SSID del punt d'acces
const char* WIFI_PASSWORD = "lluisgallart";    // Contrasenya del punt d'acces
const char* SERVER_IP = "192.168.137.1";       // IP per defecte al moment de crear el punt d'acces d'un portatil
const uint16_t SERVER_PORT = 12345;            // Port per on es fara la comunicació

/*En aquert cas tindre dos tipus de node, el node "arrel" i el node fill, per poder-ho destingit usarem aquesta variable
aixo ho fem ja que el node arrel es el encarregat de enviar la informacio al portatil i els nodes fills se envien la informacio fins arriba al node arrel*/
const bool node_arrel = true;
```

Codi 1. Variables globals del node encastat

A continuació, s'explica una part rellevant de la funció principal del programa, concretament la funció "**setup".** En primer lloc, s'estableix una connexió sèrie entre el dispositiu **ESP32** [1]i el sistema central. Paral·lelament, es procedeix a la inicialització dels sensors encarregats de mesurar la humitat ambiental i la temperatura.

Tot seguit, el sistema intenta establir una connexió Wi-Fi, configurant-se com un node client dins de la xarxa. Aquesta connexió es du a terme utilitzant les dades definides prèviament a la secció de constants del programa. Cal destacar que aquest procediment s'executa únicament si el node en qüestió actua com a node arrel; en cas contrari, és a dir, si es tracta d'un node fill, aquest procés no es realitza.

```
115  void setup() {
116      // Inicialitzem la connexio serie entre el ESP32 i el ordenador a la velocitat de 115200 bauds
117      Serial.begin(115200);
118      // Inicialitzem el de temperatura i humitat
119      sensorTH.begin();
120      if(node_arrel) {
121          // Configurem el Wifi ESP32 com a client
122          WiFi.mode(WIFI_STA);
123          WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
124          Serial.println("🌐 Conectando al punto de acceso del portátil...");
125          while (WiFi.status() != WL_CONNECTED) {
126              delay(500);
127              Serial.print(".");
128          }
129          Serial.println("\n✅ Conectado al punto de acceso");
130          Serial.println("📶 IP local: " + WiFi.localIP().toString());
131      }
```

Codi 2. Primera part de la funció principal.

Una part fonamental d'aquest sistema es la creació i connexió a la xarxa ***Mesh***, en el **Codi [3]** es pot veure la configuració principal d'aquest sistema.

Per explicar la creació i funcionament d'aquesta xarxa es farà punt per punt, en primer lloc es crear una estructura de configuració de la xarxa **Mesh** per defecte, que ens donarà la els paràmetres necessaris per configurar aquesta mateixa xarxa.

Per altra banda, després d'obtenir els paràmetres de configuració necessaris, s'especifica quin serà el canal de comunicació, en aquest cas serà el canal 6, per a que funcioni correctament tots els nodes han de estar connectats al mateix canal. A continuació hem de introduir el identificador de la xarxa dins de la configuració, per això es copia aquest identificador directament al paràmetre necessari. Una altra implementació ha estat crear un visualitzador de esdeveniments, actualment no hi farem explicacions molt concretes ja que més endavant s'explicarà com funciona exactament.

Ja per últim, en cas de ser un node arrel hem de poder accedir tant a la xarxa **Mesh** com al punt d'accés de l'aplicació central, per això el que es fa es introduir les dades de connexió d'aquest accés directament a la configuració de la xarxa **Mesh.**

```
    mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
    // Configuracio del canal WIFI
    cfg.channel = MESH_CHANNEL;
    // Es un identificador unic per poder accedir a la xarxa mesh, tots els nodes han de tenir el mateix mesh_id
    memcpy(&cfg.mesh_id.addr, MESH_ID, 6);
    // Notifica del que esta pasant amb el node sobre la xarxa mesh
    esp_event_handler_register(MESH_EVENT, ESP_EVENT_ANY_ID, &visualitzadorEvents, NULL);
    // Configuracio del punt d'acces
    strncpy((char *)cfg.router.ssid, WIFI_SSID, sizeof(cfg.router.ssid));
    strncpy((char *)cfg.router.password, WIFI_PASSWORD, sizeof(cfg.router.password));
    // Inici de la xarxa mesh
    esp_mesh_init();
    esp_mesh_set_config(&cfg);
    esp_mesh_start();

    Serial.println("Xarxa mesh iniciada");
}

void loop() {
    sendSensorData();
    delay(2500);
}
```

Codi 3. Segona part de la funció principal.

Com s'ha comentat en l'apartat anterior, s'ha creat un visualitzador d'esdeveniments amb l'objectiu d'informar, sobre el que esta passant dins del node pel que fa la gestió de la xarxa **Mesh**, el funcionament es el següent: al moment de crear aquesta xarxa es crea un element que te un identificador, aquest identificador el que fa es informar el estat actual del node, després en aquest identificador podem saber exactament el que esta passant i mostrar-ho per pantalla.

```
// Funcio de esteveniments, ens comenta que esta passant pel nostre node, si s'ha connectat a un node pare, o si ha trobat alguna connexio o algun metode
// per poder enviar les dades
void visualitzadorEvents(void* arg, esp_event_base_t event_base, int32_t event_id, void* event_data) {
    switch (event_id) {
        case MESH_EVENT_PARENT_CONNECTED:
            Serial.println("Connectat al node pare");
            break;
        case MESH_EVENT_ROOT_ADDRESS:
            Serial.println("Node arrel detectat");
            break;
        case MESH_EVENT_CHILD_CONNECTED:
            Serial.println("Node fill connectat");
            break;
        case MESH_EVENT_CHILD_DISCONNECTED:
            Serial.println("Node fill desconnectat");
            break;
        default:
            break;
    }
}
```

Codi 4. Visualitzador d'esdeveniments

En el **Codi [3]** al final de tot podem veure la funció **loop**, el que te aquesta funció es que fa de bucle infinit on s'executa reiteradament el codi de dins, com es pot observar tenim una funció anomenada "**enviamentDades**" aquesta funció es la que s'explicarà a continuació. S'ha dividit en dos parts essencials, en aquesta part es pot veure en el **codi [5]**, el que es fa es crear una variable on se introduirà un ***JSON*** *[5]*, en la variable es on s'introduiran totes les dades dels sensors en format de text per després enviar-ho o be al sistema central o a la xarxa **Mesh**.

```
/*Aquesta funcio te dos objectius
    1. En primer lloc tranformar totes les dades dels sensors en un fitxer JSON
    2. Comprovar si aquet node és arrel o fill
    3. En cas de ser node arrel enviar el JSON al portatil
    4. En cas de se node fill esta encorregar de enviar les dades a un altre node fill/arrel */
void enviamentDades() {
    // Creacio del fitxer JSON
    String json = "{";
    json += "\"id_placa\": " + String(ID_PLACA) + ",";
    json += "\"lux\": " + String(sensorLuz.llegirLlumNormalizada(), 2) + ",";
    json += "\"humetat_sol\": " + String(sensorHumedad.llegirHumetatNormalitzada(), 2) + ",";
    json += "\"temperatura\": " + String(sensorTH.llegirTemperatura(), 2) + ",";
    json += "\"humetat_ambient\": " + String(sensorTH.llegirHumetat(), 2);
    json += "}";
```

Codi 5. Primera part: Funció enviament de dades

En la segona part de la funció, es presenta la logística emprada per a l'enviament de dades, detallant els aspectes considerats per gestionar tant la recepció com la transmissió de la informació.

En primer lloc, s'estableix un condicional per determinar si el node actual és l'arrel o no. En cas que sigui un node arrel, primer s'envien les dades recollides pels seus sensors i, posteriorment, es transmeten les dades rebudes dins de la xarxa Mesh. Tant la transmissió com la recepció d'informació estan desglossades en altres funcions que es descriuen detalladament més endavant.

D'altra banda, si el node no és arrel, primer es defineix l'estructura a través de la qual s'enviaran les dades. Posteriorment, aquesta estructura es vincula al punter de la variable on es troba el ***JSON****[5]***,** la qual cosa permet introduir la informació necessària. Així mateix, es determina el tipus de transmissió i el protocol que s'utilitzarà. Finalment, les dades són enviades mitjançant una funció de la llibreria **"esp_mesh.h".**

```
    // Comprovacio del tipus de node
    if (node_arrel) {
        Serial.println("Node arrel -> enviant al portatil...");
        // Enviem les dades al portatil
        enviament(json);
        // Enviem les dades rebudes de la xarxa mesh al portatil
        enviarDadesMesh();
    } else {
        // En cas de ser node fill enviem les dades al node arrel
        Serial.println("Node fill enviant inforamcio a la xarxa mesh...");
        // Definicio de la estructura on empaquetarem tant les dades que volem enviar com els protocols que hem de seguir
        mesh_data_t data;
        // punter a les dades que volem enviar
        data.data = (uint8_t*)json.c_str();
        // Definim la mida de les dades
        data.size = json.length();
        // Definim el protocol a seguir, en aquest cas JSON
        data.proto = MESH_PROTO_JSON;
        // Definim el tipus de transmissio, en aquet cas P2P (se envien les dades de punt a punt)
        data.tos = MESH_TOS_P2P;
        //Enviament de les dades a la xarxa mesh
        esp_err_t err = esp_mesh_send(NULL, &data, MESH_DATA_P2P, NULL, 0);
        // Notificacio del resultat de l'enviament
        if (err != ESP_OK) {
            Serial.println("Error al enviar al node arrel " + String(err));
        } else {
            Serial.println("Dades enviades al node arrel " + json);
        }
    }
}
```

Codi 6. Segona part: Funció enviament de dades

Com hem pogut observar en el **Codi [6],** hi ha una funció anomenada "**enviarDadesMesh"**, la qual té com a finalitat principal la recepció de totes les dades en la xarxa **Mesh** i afegir-les a un vector que permet emmagatzemar fins a 256 bytes d'informació. Aquest està definit en el tipus "unit8_t", ja que és l'estàndard per tractament de dades en binari en C++[6].

A més, la funció incorpora un condicional que determina si s'ha rebut alguna informació de la xarxa. En cas afirmatiu, les dades es desen dins del buffer en format text i s'envien directament al sistema central.

```
void enviarDadesMesh(){
    mesh_addr_t from;
    mesh_data_t data;
    int flag = 0;
    uint8_t dadesRebudes[256];

    // Observem si hi ha dades rebudes de la xarxa mesh, en cas de haber dades rebudes,
    // les llegim, les posem al buffer i les enviem al portatil
    if (esp_mesh_recv(&from, &data, 0, &flag, NULL, 0) == ESP_OK) {
        String dadesRebudes = String((char*)data.data);
        Serial.println("Dades rebudes de la mesh: " + dadesRebudes);
        enviament(dadesRebudes);
    }

}
```

Codi 7. Funció enviar Dades Mesh

L'última funció dins del fitxer principal és **"enviament".** Aquesta la podem veure tant en el **Codi[6],** com en el **Codi[7]** . La seva finalitat és obtenir els fitxer ***JSON*** *[5]* i enviar-los al sistema central.

```
// Funcio encarregada de enviar les dades dels sensors en format JSON al portatil
void enviament(const String& json) {
    WiFiClient client;
    // En primer lloc hem de comprobar quie la conexio ha estat establerta
    if (client.connect(SERVER_IP, SERVER_PORT)) {
        // En el moment de estar establerta enviarem el JSON al portatil
        client.println(json);
        // Parem la connexio per evitar que es quedi oberta
        client.stop();
        Serial.println("Datos enviados al PC: " + json);
    } else {
        Serial.println("Error al conectar con el servidor.");
    }
}
```

Codi 8. Funció enviament

6.2 Implementació dels sensors

En aquest apartat s'explica la logística de la recol·lecció de dades dels sensors i la transformació utilitzada en cadascun d'ells. Un punt important és que, com es pot observar en la **Figura [15]**, cada sensor té dos fitxer. El primer que acaba amb la extensió ".h" correspon als fitxers destinats a la declaració de la interfície. En aquest cas no es comentaran ja que no tenen la lògica per a l'obtenció i transformació de les dades. Per contra, s'analitzaran tots els fitxers amb la extensió ".**cpp**".

El primer sensor que es descriu és el de mesura d'humitat del sòl. Com es pot observar al **Codi [9]** , hi ha dues funcions principals. La primera serveix per configurar el dispositiu encastat per a la recollida de les dades. Inicialment, s'assigna el pin al qual estarà connectat el sensor. Posteriorment, s'especifica la transformació del valor analògic a digital amb un rang de 13 bits, la qual cosa implica que el valor màxim que pot captar el sistema és de 2^13. Finalment, es configura el sistema permetent que pugui llegir voltatges de 3,3 volts sense cap problema.

La segona funció té com a objectiu la transformació del valor recollit pel sensor en un valor comprensible per a l'usuari final. Per a aquesta transformació, s'ha utilitzat la següent fórmula:[14]

$$\text{Valor Normalitzat} = (\text{Valor actual} - \text{mínim} / \text{màxim} - \text{mínim}) * 100 \quad (1)$$

L'objectiu d'aquesta és la transformació d'un valor analògic a un percentatge, ja que la quantitat d'humitat present al sòl es calcula en percentatges.

```
SensorHumetat::SensorHumetat(int pin) {
    this->pin = pin;                // Assignar el pin
    analogReadResolution(13);       // Es tranforma de analogic a digital amb un rang maxim de 13 bits
    analogSetAttenuation(ADC_11db);  // Fem que el ADC pugui llegir fins a 3.3V correctament
}

float SensorHumetat::llegirHumetatNormalitzada() {
    int rawValue = analogRead(pin);  // Lectura del ADC
    rawValue = constrain(rawValue, 80, 500);
    float humedadPorcentaje = float(rawValue - 80) / float(500 - 80) * 100.0;

    return humedadPorcentaje;  // Normalizar a rango 1 (máxima luz) a 0 (oscuridad)
}
```

Codi 9. Lectura humitat del sòl

Per fer la lectura sobre la quantitat de llum que arriba al sensor, s'utilitza el fitxer anomenat "**SensorLight.cpp**" (s'observa a la **Figura[9]**). En aquest cas, no s'explicarà en detall la funció anomenada "**SensorLluminositat**", ja que té la mateixa funcionalitat que en el **Codi[9].** Per altra banda, la funció "**llegirLlumNormalitzada**" recull el valor analògic del sensor de lluminositat i el transforma en un valor comprensible per a l'usuari.

Per transformar aquest valor s'han utilitzat dues formules principals. En primer lloc s'ha fet la conversió de la senyal obtinguda a voltatge, per fer-ho es divideix en valor obtingut per 8191 (aquest valor es el màxim que es pot obtenir, ja que sol s'agafen 13 bits) i després el resultat es multiplica per 3,3 que es el voltatge màxim enviat al pin. Aquest resultat dona el voltatge real.

$$\text{Voltatge} = (\text{valor actual} / \text{valor màxim}) * \text{voltatge enviat} \quad (2)$$

Després, es multiplica nombre màxim de lúmens que pot recollir aquest sensor (segons les especificacions el valor màxim són 1000 lúmens) i es divideix pel voltatge màxim que es pot enviar per un pin i retorna els lúmens. [15]

Lúmens = (Voltatge * 1000) / voltatge enviat (3)

```cpp
SensorLluminositat::SensorLluminositat(int pin) {
    this->pin = pin;
    analogReadResolution(13);  // Configuracio del ADC a 13 bits
    analogSetAttenuation(ADC_11db);
}

float SensorLluminositat::llegirLlumNormalizada() {
    int rawValue = analogRead(pin);  // Lectura del ADC
    //Formula per passar la senyal a "lumenes"
    float voltage = (rawValue / 8191.0) * 3.3;
    float lux = (voltage * 1000) / 3.3;
    return lux; // Retornem els lumens
}

```

Codi 10. Lectura lluminositat

L'últim fitxer per a la lectura de condicions ambientals és "**SensorTH.cpp**". Aquest fitxer recopila tant la temperatura com la humitat ambient. En aquest cas, la lectura i transformació és completament diferent als sensors anteriors, ja que aquest sensor té una llibreria anomenada "**DHT**" la qual disposa de dues funcions molt importants.

En la funció "**llegirTemperatura**" s'utilitza un mètode proporcionat per la llibreria anomenat "**readTemperature**", que retorna directament un valor comprensible de temperatura per l'usuari final. Per verificar que aquest valor és correcte s'utilitza la funció "**isnan**", ja que en cas de que sigui un valor no vàlid, retornarà -1000.

Per altra banda, hi ha la funció anomenada **"llegirHumitat".** En aquesta funció també s'utilitza la llibreria comentada anteriorment però amb el mètode "**readHumidity**", ja que retorna directament un valor comprensible. Com en la funció anterior s'utilitza "**isnan**" per validar el valor, i en cas de valor incorrecte, retorna -1.

```cpp
// Funcio que retorna la temperatura en graus Celsius
float SensorTH::llegirTemperatura() {
  float temp = dht.readTemperature();
  if (isnan(temp)) return -1000.0;  // valor de error
  return temp;
}
// Funcio que et retorna el persentatge d'humitat ambiental
float SensorTH::llegirHumetat() {
  float hum = dht.readHumidity();
  if (isnan(hum)) return -1.0;
  return hum;
}
```

Codi 11. Lectura d'humitat ambient i temperatura

7 Disseny de l'aplicació

L'aplicació del sistema central és la peça central de tot el sistema, ja que s'encarrega de recol·lectar, interpretar i analitzar totes les dades rebudes pels nodes encastats. A més, gestiona les prediccions de pluja i és on esta desenvolupat l'algoritme tant per a la taxa d'encerts de les prediccions com per a la decisió automàtica del reg.

D'altra banda, aquesta aplicació conté la interfície gràfica de l'usuari, on es podrà veure tant la decisió escollida per l'algoritme com la configuració necessària per afegir els llindars personalitzats. Finalment, es podran observar les dades recollides, desglossades per cada node encastat.

7.1 Arquitectura de l'aplicació

7.1.1 Llenguatge utilitzat

Per a la realització de l'aplicació central s'ha optat pel llenguatge de programació ***Python****[9]***.** Aquesta decisió ha estat motivada per una combinació de factors tècnics i pràctics que s'ajusten perfectament a les necessitats del projecte.

*7.1.1.5 **Python**[9]*

Per al desenvolupament del programa destinat a l'ordinador, responsable de recollir tota la informació, cridar ***APIs***, executar algoritmes i generar una interfície gràfic, es van considerar diverses opcions de llenguatge de programació, com C++, Java o JavaScript. Tanmateix, després d'analitzar els objectius del projecte i valorar quin llenguatge s'adaptava millor a les necessitats, es va optar per ***Python*** per diversos motius.

En primer lloc, ***Python*** [9] presenta una sintaxi senzilla i fàcil d'entendre, que permet escriure codi de manera ràpida, clara i eficient. A més, disposa d'un conjunt molt ampli de llibreries potents per a la gestió de xarxes i la connexió amb ***APIs***. Concretament, ofereix suport per a la comunicació TCP/IP amb els dispositius ***ESP32****[1]*, per a la consulta de prediccions meteorològiques a través ***d'APIs*** externes i per al tractament eficient de fitxers en format ***JSON***[5], que són els que contenen les dades enviades pels nodes a través de la xarxa.

Un altre aspecte rellevant és la facilitat de creació d'interfícies gràfiques mitjançant la llibreria ***Tkinter****[16]*, inclosa de manera nativa en ***Python*** [9]. Aquesta eina permet implementar aplicacions d'escriptori multi plataforma sense necessitat d'instal·lar components addicionals ni fer configuracions complexes. En aquest projecte, ha resultat una eina molt útil per dissenyar una interfície intuïtiva, que pugui ser utilitzada per usuaris sense coneixements tècnics avançats. A més, el fet que l'aplicació pugui executar-se en diferents sistemes operatius (com Windows, Linux o macOS) és un avantatge significatiu a l'hora de garantir-ne l'accessibilitat i la flexibilitat.

7.1.2 Entorn de desenvolupament utilitzat

L'entorn de desenvolupament utilitzat per a la creació del sistema central ha estat el ***Visual Studio Code****[8]***.** En aquest apartat no s'exposa de manera detallada la justificació del ús, ja que es troba explicada a l'apartat (**5.1.2.6).**

7.2 Disseny de la interfície gràfica

Per al disseny gràfic d'aquesta aplicació s'ha optat dividir-ho en tres parts fonamentals. En primer lloc, es troba la pàgina principal, des de la qual es pot visualitzar tant la interfície principal del programa com la variable objectiu: el resultat de la decisió automàtica sobre l'activació del reg.

D'altra banda, s'ha incorporat un menú lateral que permet accedir a totes les pàgines disponibles. La segona pàgina correspon a l'apartat de configuració, dedicat íntegrament a establir els paràmetres bàsics del sistema. Aquesta secció es descriu amb més detall en un apartat posterior.

Finalment, està la pàgina de nodes on es mostren tant el nombre de nodes actius com les dades que aquests estan recollint en temps real.

*7.2.1 **Pàgina principal***

En executar l'aplicació, es mostra la **Figura[12],** en la qual es pot observar, en primer lloc, el logotip de l'aplicació. Aquest s'ha dissenyat amb un estil minimalista, amb l'objectiu de no sobrecarregar visualment la pantalla amb informació innecessària. Un dels principals propòsits del projecte és oferir una eina d'ús senzill per als agricultors, per la qual cosa es prioritza una interfície clara i accessible.

D'altra banda, es pot observar que sota el logotip, es presenta una variable que indica l'estat de la vàlvula de reg. Aquesta informació es representa mitjançant un cercle amb dos possibles estats: si l'algoritme de reg automàtic ha determinat que s'ha d'activar el reg, el cercle apareix de color verd, en cas contrari, és de color vermell.

A més, s'ha optat per afegir un menú lateral prou intuïtiu, en el qual podem visualitzar els tres botons corresponents a les pàgines esmentades anteriorment.

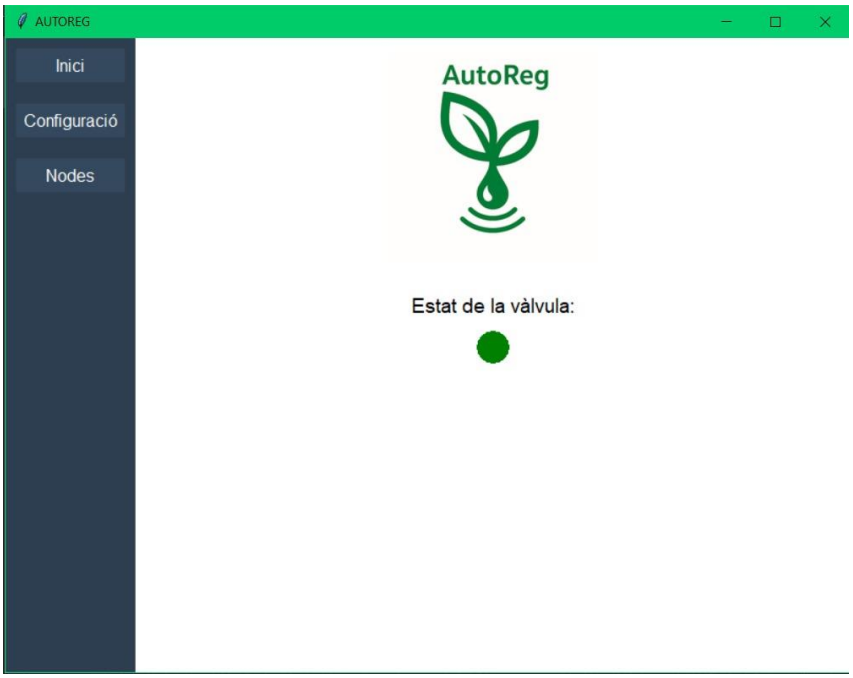

Figura 12. Pàgina principal

*7.2.2 **Pàgina de configuració***

En seleccionar l'opció de configuració, es mostrà la pàgina representada a la **Figura [12]**. Aquest secció conté tota la configuració que l'usuari ha d'introduir perquè l'aplicació funcioni correctament. S'hi poden distingir cinc apartats que cal emplenar els quatre primers corresponen als llindars que utilitza el sistema per prendre la decisió automàtica sobre l'activació del reg, mentre que el cinquè camp fa referència a la ciutat on es troba instal·lat el sistema, informació necessària per a la configuració de les **APIs** meteorològiques

A més, en aquesta pàgina s'han afegit algunes excepcions, ja que en cas de que l'usuari introdueixi valors vàlids com lletres dins dels valors numèrics o que se introdueixi valors incorrectes en la ciutat es mostri un missatge d'error clar i explicatiu. Aquest avís permet a l'usuari comprendre l'error i corregir les dades introduïdes de manera adequada.

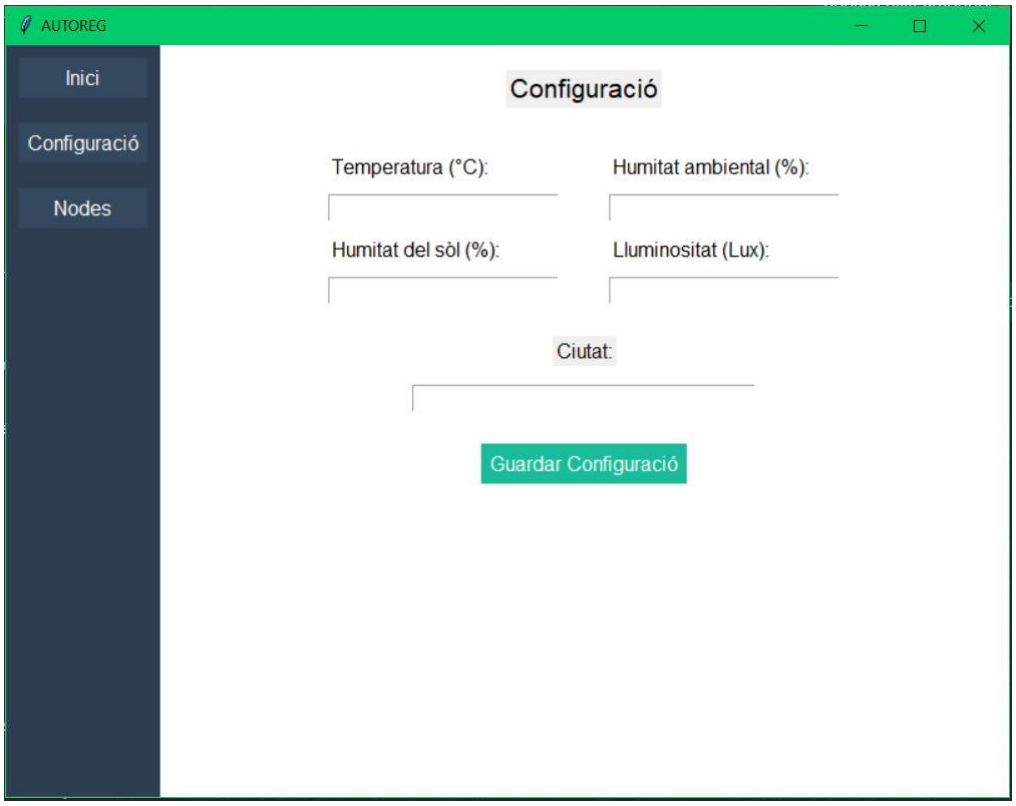

Figura 13. Pàgina de configuració

*7.2.3 **Pàgina dels nodes***

Finalment, l'última opció disponible és la finestra amb les dades dels nodes. Aquesta és la secció que mostra més informació a la pantalla i, per tant, pot resultar la més aclaparadora per a l'usuari final. No obstant això, aquesta finestra compleix diverses funcions rellevants: en primer lloc, permet visualitzar el nombre total de nodes afegits al sistema; en segon lloc, mostra en temps real les dades recollides per cada node; i, finalment, facilita la detecció de possibles incidències, com ara la desconnexió d'un node per falta d'alimentació elèctrica o qualsevol altre tipus de fallada en la transmissió de dades cap al node central.

D'altra banda, s'ha dissenyat amb l'objectiu de ser el més clara i entenedora possible per a l'usuari. Per aquest motiu, es mostren únicament els nodes configurats i les seves dades corresponents, sense cap informació addicional que pugui dificultar-ne la interpretació.

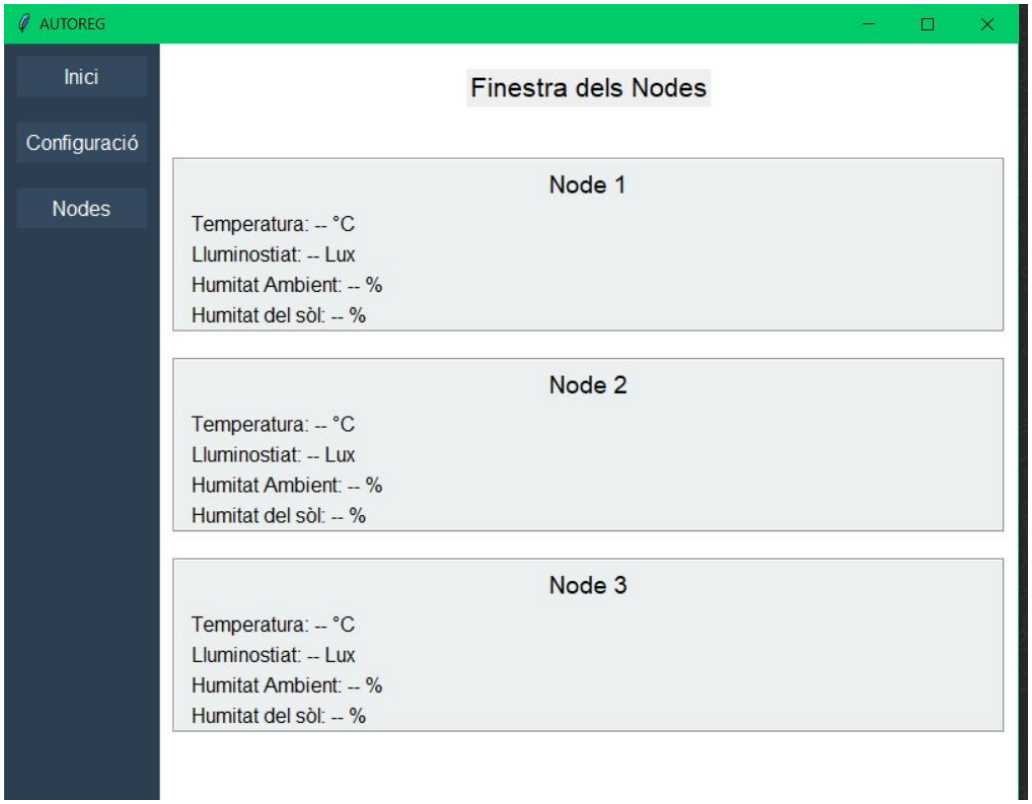

Figura 14. Pàgina dels nodes

8 Implementació de l'aplicació

Un cop definit el disseny i l'estructura funcional del sistema central, s'ha procedit a la implementació del codi que permet rebre, processar i analitzar tota la informació dels nodes encastats. Aquesta fase representa la transformació de la base teòrica exposada en els apartats anteriors en una aplicació funcional real, on es posa en pràctica la integració de la comunicació entre dispositius, la consulta de serveis externs, l'algoritme utilitzat per avaluar la fiabilitat d'aquests serveis externs i, finalment, el procediment que determina la decisió automàtica sobre l'activació del reg.

Durant tot aquest procés, s'ha procurat fer un codi eficient i modular. Com es pot observar a la **Figura [15],** l'estructura del projecte s`ha organitzat en diversos fitxers. El fitxer principal, anomenat "**api_temp.py**"**,** constitueix el nucli del projecte: conté la interfície gràfica, l'algoritme de càlcul de la taxa d'encert de les **APIs** meteorològiques i gestiona la crida als fitxers secundaris. Aquests fitxers auxiliars es descriuran detalladament més endavant, amb l'explicació del seu funcionament específic.

D'altra banda, igual que en el punt **6** , dedicat a la implementació dels nodes encastats, no es mostrarà la totalitat del codi, ja que l'objectiu es poder presentar les parts importants del codi, el seu funcionament i les decisions adoptades.

Per últim, tot el codi es veurà programat en ***Python***[9] i l'entor utilitzat ha estat ***Visual Studio Code****[8]*.

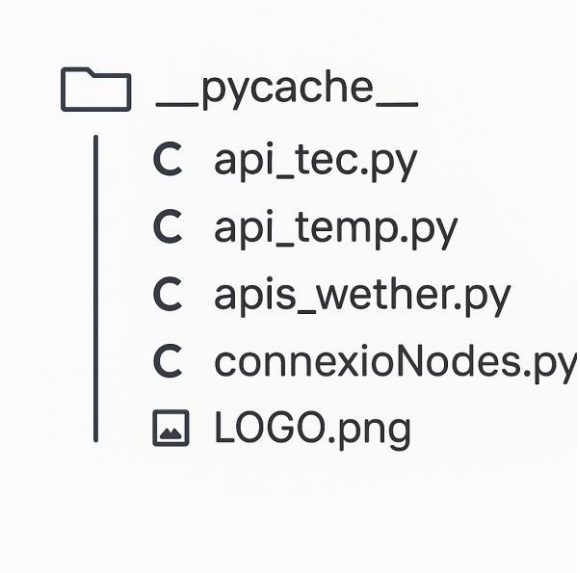

Figura 15. Estructura del sistema central

8.1 Fitxer principal

El primer element que es pot observar en el **Codi [12]** són les variables globals. En aquest cas, destaca la variable "**city**", que emmagatzema la ciutat per a la qual l'usuari vol consultar la predicció meteorològica. D'altra banda, es poden distingir la resta de variables en dues categories: les variables normals o les que acaben amb el sufix "**gui**". Les primeres corresponen a les dades recollides pels nodes encastats, mentre que les segones fan referència als llindars introduïts per l'usuari a través de la interfície gràfica.

Un altre aspecte és la presència de dues cues, que actuen com a ponts de comunicació, ja que en diversos casos s'han utilitzat "***threads"*** cridant a funcions que estan en altres fitxers i s'ha optat per emprar cues per transmetre la informació del "***thread"*** al programa principal.

```
# Constants de configuració per escollir si s'ha de regar o no
city = ""
id_placa = 1
humetat_placa = 0
humetat_gui = 0
humetat_ambient = 0
humetat_ambient_gui = 0
temperatura = 0
temperatura_gui = 0
lluminositat_gui = 0
lluminositat = 0
valvula = True

num_nodes = 3

# Variables globals que ens dirà si les apis ens indiquen que demaà ploura o no
api_wether = False
api_open = False
# Variable que ens comenta si la predicció ha sigut certa o no
pred_actual_wether = False
pred_actual_open = False
# Variable per executar sol una vegada el thread del servidor
count = 0
# Cues per agafar les dades dels threads
queue_esp32 = Queue()
queue_reg = Queue()

# Creació del event per aturar els threads
stop_event = threading.Event()
```

Codi 12. Variables globals en el sistema central

Un punt important, són les variables compartides, ja que en diversos fitxers es necessari accedir a dades actualitzades en temps real al moment, En aquests casos, no és suficient passar les dades com a paràmetres, especialment quan es treballa amb fils d'execució dins de bucles. Si una variable és passada per valor en el moment de la crida, qualsevol modificació posterior en el programa principal no es reflectiria dins del fil corresponent. Per aquest motiu, s'ha optat per utilitzar variables compartides.

```
55  # Variables compartides entre els threads que són mutables
56  shared_data1 = {"tax_api_wether": tax_api_wether,
57                  "event_close": stop_event,}
58  shared_data2 = {"tax_api_open": tax_api_open}
59  shared_data3 = {"api_wether": api_wether,
60                  "api_open": api_open,
61                  "humetat_gui": humetat_gui,
62                  "humetat_placa": humetat_placa,
63                  "humetat_ambient": humetat_ambient,
64                  "temperatura": temperatura,
65                  "temperatura_gui": temperatura_gui,
66                  "humetat_ambient_gui": humetat_ambient_gui,
67                  "lluminositat_gui": lluminositat_gui,
68                  "lluminositat": lluminositat,}
69
70
```

Codi 13. Variables compartides

Després d'haver presentat algunes de les variables globals més importants per a l'aplicació, es pot observar al **Codi [14]** el punt d'inici de l'execució del programa. A primera vista, es constata que el codi principal no és especialment extens ni complex, ja que la major part de les funcions d'aquest programa estan creades en funcions o en fitxers auxiliars.

A mes, en aquest codi es pot observar la creació del ***Tkinter****[16]* . En primer lloc, s'especifica el títol i les dimensions de la finestra. Després aquesta finestra es divideix en dues parts principals: d'una banda, el menú lateral, anomenat "**frame_menu**", que mostra en tot moment les diferents opcions disponibles del sistema (la finestra d'inici, la de configuració i la de visualització dels nodes); i, de l'altra, la secció de contingut, anomenada **"frame_contingut"**, que actualitza el seu contingut depenent de l'opció del menú escollida.

```python
# Bucle principal on esta tota la configuracio de la finestra
root = tk.Tk()
root.title("AUTOREG")
root.geometry("800x600")


# Crear el menú lateral
frame_menu = tk.Frame(root, bg="#2c3e50", width=200)
frame_menu.pack(side=tk.LEFT, fill=tk.Y)
root.protocol("WM_DELETE_WINDOW", on_closing)
# Boto per al menu de inici
btn_inicio = tk.Button(frame_menu, text="Inici", font=("Arial", 12), bg="#34495e", fg="white", relief="flat", command=mostrar_inici)
btn_inicio.pack(fill=tk.X, pady=10, padx=10)

# Botó per a la finestra de configuració
btn_configuracion = tk.Button(frame_menu, text="Configuració", font=("Arial", 12), bg="#34495e", fg="white", relief="flat", command=mostrar_comfigruacio)
btn_configuracion.pack(fill=tk.X, pady=10, padx=10)

# Boto per a la finestra dels nodes
btn_nodos = tk.Button(frame_menu, text="Nodes", font=("Arial", 12), bg="#34495e", fg="white", relief="flat", command=mostrar_nodes)
btn_nodos.pack(fill=tk.X, pady=10, padx=10)

# Creacio del frame per al contingut principal
frame_contingut = tk.Frame(root, bg="white")
frame_contingut.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Mostrar la finestra per defecte
mostrar_inici()

# Executar el bucle principal de la finestra
root.mainloop()
```

Codi 14. Codi principal

En el moment d'executar el codi la primera pàgina que es mostra per defecte és la d'inici, corresponent al **Codi [15]** . En aquesta part, en primer lloc es pot observar la crida a una funció anomenada "**llimpiar_contingut**". La seva funció s'explicarà més endavant amb detall, però la seva funció es netejar el "**frame_contingut**" per mostrar el contingut desitjat en cada moment.

A continuació, es pot veure la creació de la imatge del logotip, la qual se redimensiona a la mida desitjada i s'ubica a la part superior de la pàgina. D'altra banda, s'ha creat el indicador del reg, aquest és un cercle creat mitjançant la llibreria "***Canvas"*** , que per defecte es mostra en color vermell, Aquest color canviarà depenent de la decisió del reg.

Per veure de forma més gràfica com ha quedat aquesta pàgina, es pot observar la **Figura [12]**.

```python
# Función para mostrar la ventana de inicio
def mostrar_inici():
    global canvas, cercle
    llimpiar_contingut()

    # Carregar i redimensionar la imatge del logotip
    try:
        imatge_original = Image.open("C:/Users/lluis/Desktop/URV/CINQUE/TFG/ProgramaPortatil/TFG_portatil/LOGO.png")
        imatge_redimencionada = imatge_original.resize((200, 200), Image.Resampling.LANCZOS)  # Redimensionem la imatge
        imatge_titol = ImageTk.PhotoImage(imatge_redimencionada)
        label_imatge = tk.Label(frame_contingut, image=imatge_titol, bg="white")
        label_imatge.image = imatge_titol  # Es guarda la referencia a la imatge per evitar que sigui garbage collected
        label_imatge.pack(pady=(10, 0))  # Movem el logotip més amunt
    except Exception as e:
        messagebox.showerror("Error", f"No s'ha pogut carregar la imatge: {e}")

    # Indicador de l'estat de la vàlvula
    frame_valvula = tk.Frame(frame_contingut, bg="white")
    frame_valvula.pack(pady=(20, 0))  # Movem el frame de la vàlvula més amunt

    # Mosterem el text de l'estat de la vàlvula
    label_valvula = tk.Label(frame_valvula, text="Estat de la vàlvula:", font=("Arial", 14), bg="white")
    label_valvula.pack(pady=(5, 0))  # S'ajusta al margin superior
    canvas = tk.Canvas(frame_valvula, width=50, height=50, bg="white", highlightthickness=0)
    canvas.pack()

    # Dibuixar el cercle inicial, per defecte esta de color roig
    color = "green" if valvula else "red"
    cercle = canvas.create_oval(10, 10, 40, 40, fill=color, outline=color)

```

Codi 15. Mostrar finestra d'inici

Una de les altres finestres afegides és la corresponent a la secció de configuració. Com en la resta de finestres, el primer pas consisteix en netejar el contingut previ mitjançant la funció que elimina la informació mostrada anteriorment, per tal d'afegir el nou contingut de manera ordenada.

En aquesta finestra es disposen tots els camps necessàries perquè l'usuari pugui afegir tots els llindars necessaris. A més, s'afegeix un camp addicional que permet especificar la ciutat on està la plantació per poder fer les prediccions necessàries.

Per observar amb més detall l'aspecte d'aquesta finestra, es pot consultar la **Figura [13].**

```python
# Función para mostrar la ventana de configuración
def mostrar_comfigruacio():
    llimpiar_contingut()
    label_config = tk.Label(frame_contingut, text="Configuració", font=("Arial", 16))
    label_config.pack(pady=20)

    # Crear un frame per organizar les columnes
    frame_columnes = tk.Frame(frame_contingut, bg="white")
    frame_columnes.pack(pady=10)
    # Columna esquerra
    frame_esquerra = tk.Frame(frame_columnes, bg="white")
    frame_esquerra.pack(side=tk.LEFT, padx=20)

    # Camp per a la configuració de la temperatura
    label_temperatura = tk.Label(frame_esquerra, text="Temperatura (°C):", font=("Arial", 12), bg="white")
    label_temperatura.pack(pady=5, anchor="w")
    entry_temperatura = tk.Entry(frame_esquerra, font=("Arial", 12))
    entry_temperatura.pack(pady=5)

    # Camp per a la configuració de la humitat del sòl
    label_humitat = tk.Label(frame_esquerra, text="Humitat del sòl (%):", font=("Arial", 12), bg="white")
    label_humitat.pack(pady=5, anchor="w")
    entry_humitat = tk.Entry(frame_esquerra, font=("Arial", 12))
    entry_humitat.pack(pady=5)

    # Columna dreta
    frame_dreta = tk.Frame(frame_columnes, bg="white")
    frame_dreta.pack(side=tk.LEFT, padx=20)

    # Camp per a la configuració de la humitat ambiental
    label_humitat_ambiental = tk.Label(frame_dreta, text="Humitat ambiental (%):", font=("Arial", 12), bg="white")
    label_humitat_ambiental.pack(pady=5, anchor="w")
    entry_humitat_ambiental = tk.Entry(frame_dreta, font=("Arial", 12))
    entry_humitat_ambiental.pack(pady=5)

    # Camp per a la configuració de la lluminositat
    label_lluminositat = tk.Label(frame_dreta, text="Lluminositat (Lux):", font=("Arial", 12), bg="white")
    label_lluminositat.pack(pady=5, anchor="w")
    entry_lluminositat = tk.Entry(frame_dreta, font=("Arial", 12))
    entry_lluminositat.pack(pady=5)

    # Camp central per a la configuració de la ciutat
    label_ciutat = tk.Label(frame_contingut, text="Ciutat:", font=("Arial", 12))
    label_ciutat.pack(pady=10)
    entry_ciutat = tk.Entry(frame_contingut, font=("Arial", 12), width=30)
    entry_ciutat.pack(pady=5)
```

Codi 16. Finestra de configuració

Una funcionalitat afegida ha estat el botó de configuració, el qual permet cridar la funció anomenada **"guardar_configuracio"** , que s'explicarà més endavant.

```
    # Botó para guardar la configuración
    btn_guardar = tk.Button(
        frame_contingut,
        text="Guardar Configuració",
        font=("Arial", 12),
        bg="#1abc9c",
        fg="white",
        relief="flat",
        command=lambda: guardar_configuracio(entry_temperatura, entry_humitat, entry_humitat_ambiental, entry_lluminositat, entry_ciutat)
    )
    btn_guardar.pack(pady=20)

```

Codi 17. Boto de configuració

Per acabar amb la descripció de les diferents finestres, es disposa una secció dedicada a la mostra de tots els nodes connectats actualment al sistema. Aquesta funció es pot veure al **Codi[18]**. Tal com s'observa, es tracta d'una secció composta per diversos elements.

En primer lloc, s'implementa un bucle per generar dinàmicament tots els contenidors necessaris depenent del nombre de nodes actius. Dins de cada contenidor es mostra tant l'identificador del node com les dades que ha recollit en temps real.

A més, s'ha incorporat una estructura que agrupa les variables corresponents a cada sensor, amb l'objectiu de permetre l' actualització de forma dinàmica. Aquesta gestió facilita tant la visualització clara de la informació com la gestió eficient dels valors associats.

Finalment, per observar amb més detall l'aspecte d'aquesta finestra, es pot consultar la **Figura [14].**

```
# Funció per mostrar la finestra de nodes en temps real
def mostrar_nodes():
    global num_nodes, frame_contingut, shared_data3, nodes_labels

    llimpiar_contingut()

    label_nodes = tk.Label(frame_contingut, text="Finestra dels Nodes", font=("Arial", 16))
    label_nodes.pack(pady=20)

    # Contenedor per als nodes
    frame_nodes = tk.Frame(frame_contingut, bg="white")
    frame_nodes.pack(pady=10, fill=tk.BOTH, expand=True)


    # Es crea de forma dinamica cada node, ja que depenent de la implementacio es pot una quantitat diferent de nodes.
    for i in range(1, num_nodes + 1):
        # Es crea el contenedor per a cada node
        frame_node = tk.Frame(frame_nodes, bg="#ecf0f1", bd=2, relief="groove")
        frame_node.pack(pady=10, padx=10, fill=tk.X)

        # El titol del node
        label_titol = tk.Label(frame_node, text=f"Node {i}", font=("Arial", 14), bg="#ecf0f1")
        label_titol.pack(pady=5)

        # Etiquedtes de cada sensors
        label_temperatura = tk.Label(frame_node, text="Temperatura: -- °C", font=("Arial", 12), bg="#ecf0f1")
        label_temperatura.pack(anchor="w", padx=10)

        label_lluminositat = tk.Label(frame_node, text="Lluminostiat: -- Lux", font=("Arial", 12), bg="#ecf0f1")
        label_lluminositat.pack(anchor="w", padx=10)

        label_humitat_ambiental = tk.Label(frame_node, text="Humitat Ambient: -- %", font=("Arial", 12), bg="#ecf0f1")
        label_humitat_ambiental.pack(anchor="w", padx=10)

        label_humitat_placa = tk.Label(frame_node, text="Humitat del sòl: -- %", font=("Arial", 12), bg="#ecf0f1")
        label_humitat_placa.pack(anchor="w", padx=10)

        # Com aquestes dades s'han de modificar de forma dinamica, s'afegeixen en un vector, cada valor del vector fa referencia a un node en concret.
        nodes_labels[i] = {
            "temperatura": label_temperatura,
            "lluminositat": label_lluminositat,
            "humitat_ambient": label_humitat_ambiental,
            "humitat_placa": label_humitat_placa,
        }
```

Codi 18. Finestra dels nodes

Després d'haver explicat detalladament totes les finestres incloses en aquesta aplicació, cal destacar altres funcions fonamentals per al correcte funcionament del sistema. Una d'elles és la funció "**guardar_configuracio**" la qual es la encarregada de validar totes les dades introduïdes per l'usuari i, en cas que hi haguí algun error, mostrar-ho per pantalla.

D'altra banda, si les dades introduïdes són correctes i no es detecta cap error de validació, la funció procedeix a realitzar diverses operacions addicionals, necessàries per al bon funcionament del sistema.

La primera operació que es realitza és l'emmagatzematge de totes les dades a la variable global "**Shared_data3**", ja que com hem comentat anteriorment, aquestes dades han d'estar disponible de manera dinàmica per a altres funcions del sistema.

Després d'emmagatzemar totes les dades configurades per l'usuari, s'ha pres la decisió de disseny de crear i iniciar tots els ***threads*** des d'aquesta mateixa funció. Aquesta elecció respon a la necessitat que tot el sistema s'activi només quan l'usuari prem el botó de guardar. D'aquesta manera, es garanteix que no s'inicií cap procés de recepció de dades, consulta ***APIs***, ni anàlisi del reg fins que es disposi de tota la configuració necessària.

D'altra banda, veiem que tenim un fil que sol es pot executar una vegada. Aquesta implementació s'ha fet ja que l'usuari pot canviar en tot moment la configuració dels llindars i de la ciutat, però no es recomanable tornar a cridar la funció de **"inciar_servidor"**, ja que reiniciar el servidor podria provocar errors inesperats.

Finalment, al **Codi[19]** es poden observar diverses crides de funcions auxiliars. Aquestes es detallaran més endavant, amb una explicació especifica de la seva funcionalitat i del seu paper dins del funcionament general.

```python
def guardar_configuracio(entry_temperatura, entry_humitat, entry_humitat_ambiental, entry_lluminositat, entry_ciutat):
    global count, shared_data3, thread1, thread2, thread3, thread4, queue_esp32, queue_reg
    try:
        # En primer lloc es validen que tots els camps estiguin plens
        if not all([entry_temperatura.get(), entry_humitat.get(), entry_humitat_ambiental.get(), entry_lluminositat.get(), entry_ciutat.get()]):
            raise ValueError("Todos los campos deben estar llenos.")

        # Es validar que tots els camps siguin numerics
        temperatura_gui = float(entry_temperatura.get())
        humetat_gui = float(entry_humitat.get())
        humetat_ambient_gui = float(entry_humitat_ambiental.get())
        lluminositat_gui = float(entry_lluminositat.get())


        # Es valida que el camp de ciutat sigui text
        city = entry_ciutat.get()
        if not city.isalpha():
            raise ValueError("El camp 'Ciutat' sol pot contenir lletres")

        print(f"humetat_placa actualizada a: {humetat_placa}")
        print(f"Ciudad actualizada a: {city}")

        # Despres de fer totes les validacions, s'emmagatzema totes aquestes dades al shared_data3 per despres compartir-les
        # en el thread corresponent
        shared_data3["lluminositat_gui"] = lluminositat_gui
        shared_data3["humetat_ambient_gui"] = humetat_ambient_gui
        shared_data3["humetat_gui"] = humetat_gui
        shared_data3["temperatura_gui"] = temperatura_gui
        # Creacio dels Thread
        thread1 = threading.Thread(target=function_wetherapi, args=(shared_data1, API_KEY_WEATHERAPI, city))
        thread2 = threading.Thread(target=function_openwether, args=(shared_data2, API_KEY_OPENWEATHER, city))
        #En aquet cas com sol hem de cridar una vegada a la funció "Inciar servidor" es necessari afegir aquet IF
        if count == 0:
            thread3 = threading.Thread(target=iniciar_servidor, args=(queue_esp32,shared_data1,))
            thread3.start()

            count += 1
        thread4 = threading.Thread(target=comprobacions, args=(shared_data3, shared_data1["tax_api_wether"], shared_data2["tax_api_open"],queue_reg))
        # Inicialitzar els threads restants
        thread1.start()
        thread2.start()
        thread4.start()

        # Mostrar el missatge
        messagebox.showinfo("Éxito", "Configuració guardada correctament")
        print(f"Configuració guardada:\nTemperatura: {temperatura}°C\Humitat: {humetat_gui}%\Humitat Ambiental: {humetat_ambient}%\\Lluminositat: {lluminositat} Lux\nCiutat: {city}")
```

Codi 19. Funció per guardar la configuració

A continuació, s'explicarà amb mes detall les funcions encarregades de gestionar la crida de les **APIs** meteorològiques utilitzades en el sistema. Aquestes s'encarreguen de realitzar les peticions oportunes, extraient les dades rellevants i utilitzant-les per a l'anàlisi.

A més, es descriurà l'algoritme implementat per avaluar la fiabilitat d'aquestes dades obtingudes de cada una de les fonts. Aquesta avaluació es basa en la comparació entre dues prediccions, amb l'objectiu de determinar quina ofereix resultats més precisos en el context del sistema. Aquesta informació serà clau per a la presa de decisions automàtica relacionada amb l'activació del reg.

Per començar, es pot observar en el **Codi [20]** la implementació d'un d'aquests algoritmes de prediccions meteorològiques. En aquest primer fragment de codi, s'inicia el procés amb la crida de la funció "**temp_openwether**", s'encarrega de realitzar una petició a *l'**API*** d'***OpenWether**[2]* i retorna la predicció meteorològica corresponent del dia següent. Aquesta funció comentada anteriorment serà explicada en un apartat posterior.

Un cop s'obté la predicció completa, es filtra per extreure si el valor obtingut indica precipitacions o no. Aquesta dada s'emmagatzema ja que és una variable clau que permet l'inici del procés de validació.

Tot seguit, el sistema entra en un bucle que es manté actiu fins que s'assoleix l'hora programada per realitzar la fiabilitat. Un cop arribat aquest moment, es comprova si la predicció ha estat correcta o no. En cas que la predicció sigui correcta, es suma un valor a la variable de taxa d'encerts en cas contrari es resta una unitat.

D'altra banda, aquest valor té un rang entre 5 i -5. Aquest actua com un indicador de la fiabilitat assignada a cada predicció meteorològica i s'actualitza automàticament després de cada encert o error. El límit superior representa l'alta fiabilitat d'aquesta ***API*** i el límit inferior representa la baixa fiabilitat. Aquest mecanisme permet al sistema ajustar de forma dinàmica la confiança de la font de dades, ja que aquesta dada és fonamental per a la decisió automàtica de reg.

```python
# Amb aquesta funció tenim el mateix objectiu que amb l'anterior, però utilitzant una altra API, ja que la finalitat es tenir com a referent
# l'api que tingui la taxa més elevada d'encerts.
# En aquest cas no modificarem els valors que estan dins de la finestra ja que ho hem fet en l'anterior funció.
def function_openwether(shared_data2, key_api, city):
    global api_open
    global shared_data3
    global pred_actual_open
    # Es defineix l'hora exacta per comprovar si l'API ha encertat o no.
    hora_objectiu = datetime.now().replace(hour=18, minute=44, second=1, microsecond=0)
    while not stop_event.is_set():
        # Cridem a l'API openwether per veure si plourà o no.
        Probabilitat_pluja = temp_openwether(key_api, city)
        print(f"(openwether) Probabilitat de pluja: {Probabilitat_pluja}")
        #D'aquesta manera, l'API no ens retorna la probabilitat, sinó que ens indica si plou, està ennuvolat o altres condicions.
        # En aquest cas, només utilitzarem la informació sobre si està plovent, que és la que ens interessa.
        if Probabilitat_pluja == "Rain":
            api_open = True
        else:
            api_open = False
        # Guardem el resultat al shared_data3 per poder-ho utilitzar al thread4.
        shared_data3["api_open"] = api_open
        # Bucle bloquejant que serveix per comprovar si la predicció s'ha encertat o no, en funció de l'hora configurada.
        while not stop_event.is_set():
            ahora = datetime.now()
            if ahora.hour == hora_objectiu.hour and ahora.minute == hora_objectiu.minute and ahora.second == hora_objectiu.second:
                print(f"Hora objetiu: {hora_objectiu.strftime('%H:%M')}")
                break
            stop_event.wait(1)
        pred_actual_open = temp_openwether_today(key_api, city)
        # Comprovació per veure si l'api a encertat
        if pred_actual_open > 0 :
            pred_actual_open = True
        else:
            pred_actual_open = False
        # Increment o decrement de la taxa d'encerts
        if shared_data2["tax_api_open"] < 6 and shared_data2["tax_api_open"] > -6:
            if pred_actual_open == api_open:
                shared_data2["tax_api_open"] += 1
                print(f"Taxa api_open: {shared_data2['tax_api_open']}")
            elif pred_actual_open != api_open:
                shared_data2["tax_api_open"] -= 1
        stop_event.wait(1)
```

Codi 20. Funció API OpenWether

Després d'haver explicat detalladament la primera funció sobre la crida de la ***API* OpenWether**[2], es el moment de descriure el comportament de la segona ***API***. Tal com es pot observar en el **Codi [21],** la seva implementació presenta certes diferències respecte a l'anterior. Concretament, per tal de fer més eficient el programa, s'ha incorporat dins del bucle bloquejant, la recollida constant de de tota la informació dels nodes encastats i l'actualització de la finestra dels nodes, es pot veure a la **Figura[14]**.

D'altra banda, aquesta ***API*** no et retorna l'estat de la predicció, sinó que retorna la probabilitat de precipitacions expressada en percentatge. En aquest punt, com a decisió de disseny, s'ha establert que si la probabilitat de precipitació expressada en percentatge és superior al 50%, es considera que la predicció indica pluja; en cas contrari, s'assumeix que no plourà.

```python
# incrementa la taxa d'encerts en 1; en cas contrari, la decrementa en 1.
def function_wetherapi(shared_data1, key_api, city):
    global nodes_labels, id_placa, cercle, canvas
    global api_wether
    global shared_data3
    global pred_actual_wether
    global humetat_placa, temperatura, lluminositat, humetat_ambient, valvula
    # Es defineix l'hora exacta per comprovar si la API ha encertat o no
    hora_objectiu = datetime.now().replace(hour=18, minute=44, second=1, microsecond=0)
    # Bucle principal que s'executarà fins que es tanqui la finestra o es pari el thread
    while not stop_event.is_set():
        # Obtenim la probabilitat de l'api "wetherapi"
        Probabilitat_pluja = temp_wetherapi(key_api, city)
        print(f"(wetherapi) Probabilitat de pluja: {Probabilitat_pluja}%")
        # Com necessitem un valor boolean i aquesta api ens retorna una probabilitat, entenem que si és major d'un 50%, plourà,
        # i si és menor, no plourà.
        if Probabilitat_pluja > 50:
            api_wether = True
        else:
            api_wether = False
        # Emmagatzemem el valor al shared_data3 que ultilitzarem per al thread4
        shared_data3["api_wether"] = api_wether
        # Bucle bloquejant té dos treballs principals:
        #  1. Modificar els valors de la finestra a temps real per veure els valors actuals de les dades.
        #  2. Tenir bloquejat aquest bucle fins que es compleixi l'hora exacta per veure si l'api a encertat aquell dia.
        # En cada iteració del bucle, el deixem en repós 1 segon per no saturar el processador
        while not stop_event.is_set():
            # TODO: Afegir aquet try i cach en una funció separada

            try:
                mensaje = queue_esp32.get(timeout=1)  # Esperar hasta 1 segundo para obtener datos
                datos = json.loads(mensaje)  # Decodificar el mensaje JSON
                id_placa = int(datos.get("id_placa", "0"))  # Obtenim l'id de la placa
                lluminositat = datos.get("lux", "0")
                humetat_placa = datos.get("humetat_sol", "0")
                temperatura = datos.get("temperatura", "0")
                humetat_ambient = datos.get("humetat_ambient", "0")
                print(f"Actualitzant dades del node {id_placa}")
                valvula = queue_reg.get()  # Obtenim el valor de la valvula de la cua
                root.after(0, actualizar_circulo_valvula)  # Actualizamos el círculo en el hilo principal
                id_placa = int(id_placa)  # Convertir a enter
                if id_placa in nodes_labels:
                    nodes_labels[id_placa]["temperatura"].config(text=f"Temperatura: {temperatura} °C")
                    nodes_labels[id_placa]["lluminositat"].config(text=f"Luminosidad: {lluminositat} Lux")
                    nodes_labels[id_placa]["humitat_ambient"].config(text=f"Humedad Ambiental: {humetat_ambient} %")
                    nodes_labels[id_placa]["humitat_placa"].config(text=f"Humedad Placa: {humetat_placa} %")

            except Empty:
```

Codi 21. Primera part Funció WetherApi

Per altra banda, la segona part del codi (visible al **Codi [22]**) segueix la mateixa lògica i estructura de funcionament que la presentada anteriorment al **Codi [20].**

```python
            # Agafem l'hora actual
            ahora = datetime.now()

            # Comprovem si l'hora actual coincideix amb l'hora configurada
            if ahora.hour == hora_objectiu.hour and ahora.minute == hora_objectiu.minute and ahora.second == hora_objectiu.second:
                print(f"Hora objectiu: {hora_objectiu.strftime('%H:%M')}")
                break  # En cas d'arribar a l'hora objectiu, sortirem del bucle.
            stop_event.wait(1)  # Fem esperar en cada iteració 1 segon per no saturar el processador.
        # Cridem a temp_openwether_today per veure si avui ha plogut o no, i incrementar o decrementar la taxa d'encerts.
        pred_actual_wether = temp_openwether_today(key_api, city)
        if pred_actual_wether > 0 :
            pred_actual_wether = True
        else:
            pred_actual_wether = False
        # Comparem els valors de l'api per veure si ha encertat o no.
        # Per una altra banda, observem que només pot sumar fins 5 o restar fins -5, ja que volem aquet rang.
        if pred_actual_wether == api_wether and shared_data1["tax_api_wether"] <= 5:
            shared_data1["tax_api_wether"] += 1
            print(f"Taxa api_wether: {shared_data1['tax_api_wether']}")
        elif pred_actual_wether != api_wether and shared_data1["tax_api_wether"] >= -5:
            shared_data1["tax_api_wether"] -= 1
        stop_event.wait(1)
```

Codi 22. Segona part Funció WetherApi

8.2 Fitxers secundaris

Després d'explicar en detall el fitxer principal i la seva funcionalitat, cal comentar també els fitxer secundaris, ja que tenen un paper molt rellevant i aporten funcionalitats essencial dins d'aquest projecte.

*8.2.1 **ConnexioNodes.py***

En primer lloc, es tracta la comunicació entre el sistema central i els nodes arrel. Per fer aquesta connexió, el sistema central crea un punt d'accés, on després el node arrel es podrà connectar i enviar les dades.

Cal tenir en compte que, si sol esta el punt d'accés obert però l'aplicació no està funcionant, el sistema central no rebrà les dades fins que aquest sistema estigui actiu. Per obtenir aquestes dades, el sistema central cridarà a la funció anomenada "**iniciar_servidor**", la qual es pot observar al **Codi [23]**.

Aquest funció s'encarrega d'iniciar el servidor d'escolta per tal de poder recollir les dades enviades i després amb un bucle bloquejant, escolta reiteradament totes les dades que s'envien des del node arrel. Posteriorment, aquestes dades s'afegeixen dins de la cua per a que el programa principal pugui recollir-les i analitzar-les.

```
3   HOST = '0.0.0.0'  # Accepta les connexions des de qualsevol IP
4   PORT = 12345      # Es el mateix port que esta configurat en els nodes ESP32
5
6   def iniciar_servidor(queue_esp32):
7       with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as servidor:
8           servidor.bind((HOST, PORT))
9           servidor.listen()
10
11          while True:
12              print(f"ESCOLTANT\n")
13              connexio, direccio = servidor.accept()
14              with connexio:
15                  print(f"Conexiò entrant desde {direccio}")
16                  dades = connexio.recv(1024).decode().strip()
17                  if dades:
18                      queue_esp32.put(dades)
19
```

Codi 23. Fitxer connexió nodes

8.2.2 ***Apis_wether.py***

Una de les parts fonamentals d'aquest projecte és l'ús de prediccions meteorològiques, atesa la necessitat de determinar si l'endemà es preveu pluja. Per implementar aquest sistema, s'han utilitzat dos ***APIs*** meteorològiques principals: ***OpenWether****[2]* i ***WetherAPI****[3]*. Tal com es pot observar al **Codi[24]**, s'han implementat tres funcions.

La primera funció fa referencia a la crida de l'***API WetherAPI****[3]*, on s'inclou la clau d'accés generada en el moment de crear el compte, així com la ciutat d'interès. Aquesta petició retorna un fitxer en format ***JSON****[5]* amb la informació meteorològica prevista.

Un cop rebuda la resposta, s'aplica un filtre per extraure únicament les dades importants. En aquest cas, com l'objectiu és buscar la predicció per a l'endemà, s'obté automàticament la data actual i se li suma un dia per obtenir la data. A continuació, es recorre l'estructura del ***JSON****[5]* fins a localitzar la informació corresponent a aquesta data, i finalment s'extreu la probabilitat de pluja, que és enviada al fitxer principal per ser analitzada.

Pel que fa a la funció "**temp_openwether**", l'execució segueix un patró molt similar a la funció comentada anteriorment. En aquest cas, s'utilitza ***l'API OpenWether****[2]* per obtenir la predicció meteorològica. Tot i que la lògica és equivalent, el recorregut de l'estructura ***JSON*** [5] en lloc de retornar la probabilitat de pluja, proporciona directament la predicció (si plourà, estarà núvol, etc.). Després aquest valor s'envia a la funció principal per ser analitzada.

Finalment, es presenta la funció "**temp_openwetherapi_today**", que té com a objectiu consultar el percentatge de pluja registrat a la ciutat i retorna un valor 0 en cas de que no s'hagi registrat pluja o un valor major a 0 en cas contrari.

A continuació, s'utilitza un condicional per interpretar aquest valor i retornar-ho al programa principal.

```python
import requests
from datetime import datetime, timedelta, timezone
def temp_wetherapi(api_key, ciutat):
    url = f"http://api.weatherapi.com/v1/forecast.json?key={api_key}&q={ciutat}&days=2&aqi=no&alerts=no"
    response = requests.get(url)
    data = response.json()
    dema = (datetime.now(timezone.utc) + timedelta(days=1)).strftime('%Y-%m-%d')
    # Calcular la fecha exacta para mañana
    for prediccio in data['forecast']['forecastday']:
        if prediccio['date'].startswith(dema):
            return prediccio['day']['daily_chance_of_rain']

def temp_wetherapi_today(api_key, ciutat):
    avui = datetime.now().strftime('%Y-%m-%d')
    url = f"http://api.weatherapi.com/v1/history.json?key={api_key}&q={ciutat}&dt={avui}"
    response = requests.get(url)
    data = response.json()
    # Precipitació total del dia
    precip = data['forecast']['forecastday'][0]['day']['totalprecip_mm']
    if precip > 0: #Si la precipitacio a estat major a 0, vol dir que ha plogut durant el dia.
        return True
    return False


def temp_openwether(api_key, ciutat):
    url = f"http://api.openweathermap.org/data/2.5/forecast?q={ciutat}&appid={api_key}&units=metric"
    response = requests.get(url)
    data = response.json()

    dema = (datetime.now(timezone.utc) + timedelta(days=1)).strftime('%Y-%m-%d')
    # Calcular la fecha exacta para mañana
    for prediccio in data['list']:
        if prediccio['dt_txt'].startswith(dema):
            return prediccio['weather'][0]['main']

```

Codi 24. Fitxer Apis_wether

*8.2.3 **Api_tec.py***

Finalment, es pot observar el fitxer anomenat **"api_tec.py".** Aquest fitxer s'encarrega de recollir les dades obtingudes tant del sistema de nodes encastats com de les prediccions meteorològiques, així com de taxes d'encerts. A més, aquest fitxer te dues funcions principals.

La primer funció, anomenada "**comprovacions**" (visible al **Codi[25]**), s'encarrega d'obtenir totes les dades compartides del sistema principal per després analitzar-les. Aquestes estan incloses en tres variables diferents. La variable anomenada **"Shared_data"** conté tant els valors dels sensors com els llindars configurats per l'usuari.

A més, "**tax_api1**" emmagatzema la taxa d'encerts de *l'**API WetherAPI**[3]* i la "**tax_api2**" conté l'***API OpenWether**[2].* D'altra banda, s'observa la cua anomenada "**queue_regadiu**", la qual permet transmetre la decisió automàtica de reg directament al programa principal per després mostrar-ho directament a l'usuari.

Un cop obtenies totes les dades, s'ha pres la decisió d'afegir un rang a la humitat del sòl, ja que és la dada mes rellevant, ja que calcula la quantitat d'humitat que arriba a les arrels de les plantes. L'objectiu d'aquesta tècnica és assolir el nivell d'humitat òptim com més aviat possible.

A continuació, es calculen les taxes d'encerts. En aquest cas, s'ha decidit que només es tindran en compte les prediccions de les ***APIs*** quan el valor sigui igual o superior a 0, ja que aquest valor indica una fiabilitat acceptable en les prediccions recents. Així, es descarten aquells valors que hagin presentat més errors que encerts, evitant que influeixin negativament en la presa de dedicions del sistema.

Un cop analitzada la taxa d'encerts i en cas que la predicció indiqui que l'endemà plourà, el sistema descarta automàticament la necessitat de reg durant aquell dia. En canvi, si les taxes són incorrectes o les prediccions diuen que demà no hi haurà precipitacions, s'accedeix a la funció "**regadiu**".

Aquesta funció, que es pot consultar al **Codi[26]**, implementa un conjunt de prioritats a cada dada. Per exemple, si la humitat del sòl es troba per sota del llindar recomanat, s'afegeixen 2 punts a la variable "**prioritats**". En canvi, la resta de valors sol valen un punt, ja que s'ha valorat que la humitat del sòl es una dada a destacar per sobre de la resta.

Finalment, després de calcular la prioritat, es comprova si el valor de la variable és igual o superior a 2. En cas afirmatiu, es genera l'ordre d'activació del reg.

```python
# Funcio d'analisis de dades d'entrada tant de les APIS com de les dades de la placa ESP32
def comprovacions(shared_data, tax_api1, tax_api2,queue_regadiu):
    # Variable el qual sera la que retornara si la valvula de regadiu s'ha d'obrir
    valvula = True
    while True:
        # Descarrega del shared_data tots els valors necessaris, sobre valors de la placa ESP32
        temperatura_placa = shared_data["temperatura"]           # temperatura_placa actual de la placa
        lluminositat_placa = shared_data["lluminositat"]                 # Lluminositat de la placa ESP32
        humetat_placa = shared_data["humetat_placa"]       # Humetat del terra de la placa ESP32
        humetat_hambient_placa = shared_data["humetat_ambient"]  # Humetat ambient de la placa ESP32
        pred_api1 = shared_data["api_wether"]              # Prediccio de "wetherapi"
        pred_api2 = shared_data["api_open"]                # Prediccio de "openwether"


        temperatura_gui = shared_data["temperatura_gui"]          # Temperatura configurada per l'usuari en la finestra de configuracio
        humetat_hambient_gui = shared_data["humetat_ambient_gui"]       # Humetat ambient configurada per l'usuari en la finestra de configuracio
        lluminositat_gui = shared_data["lluminositat_gui"]          # Lluminositat configurada per l'usuari en la finestra de configuracio
        humetat_gui = shared_data["humetat_gui"]            # Humetat configurada per l'usuari en la finestra de configuracio

        # Tranformar els valors en enters
        humetat_placa = int(humetat_placa)
        humetat_hambient_placa = int(humetat_hambient_placa)
        temperatura_placa = int(temperatura_placa)
        lluminositat_placa = int(lluminositat_placa)
        # Per no posar el valor de la humetat configurada, generem un rang de 10 unitats per sota per intentar arribar al valor estimat el mes pronte possible.
        humitat_rang = humetat_gui - 10

        # En primer lloc comprovem si les taxes d'encerts de alguna de les APIS es igual o superior a 0, ja que en cas contrari
        # directament no hem de regar
        if tax_api1 >= 0 or tax_api2 >= 0:
            # En cas de que alguna de les APIS hagi tingut una taxa d'encerts correcta, comprovem si les APIS ens diuen si plourà.
            # En cas de ploure directament no reguem.
            if pred_api1 == True or pred_api2 == True:
                valvula = False
            else:
                # En cas de que no plogui, cridem a la funcio regadiu, el qual ens retorna un boolean que ens indica si s'ha de regar.
                valvula = regadiu(humitat_rang, temperatura_placa, lluminositat_placa, humetat_placa, humetat_hambient_placa, temperatura_gui, humetat_hambient_gui, lluminositat_gui)
        else:
            valvula = False
        # Afegim el resultat final de la valvula a la cua de regadiu, ja que necessitem aquet valor al main.
        print(f"valvula: {valvula}")
        queue_regadiu.put(valvula)
        time.sleep(2)  # En cada iteracio esperem 2 segons per no saturar el processador
```

Codi 25. Funció comprovacions

```python
# Funcio que ens indica si regarem en funcio les dades recollides de la placa ESP32.
def regadiu (humedad_rango, temperatura_placa, lluminositat_placa,humetat_placa, humedad_ambient_placa, temperatura_gui, humetat_hambient_gui, lluminositat_gui):
    # En aquet cas el que farem sera tenir una variable anomenada "prioritat", el qual depenent de les condicions que ens donin,
    # s'anira sumant aquet valor, al final de la funcio si aquet valor es superior o igual a 2 i la hora es valida, regarem.
    prioritat = 0
    # Comparacio entre el rang d'humetats que volem sobre la humetat del terra de la placa ESP32.
    if humetat_placa < humedad_rango:
        prioritat += 2
    # Comparacio entre la lluminositat ambiental i el humbral de llum, ja que si hi ha molta llum,
    # el terra es seca mes rapidament fent que sigui necessari regar mes aviat.
    if lluminositat_placa > lluminositat_gui:
        prioritat += 1
    # Com amb el cas anterior, si la temperatura_placa es superior a l'umbral, tindrem que regar mes aviat ja que el terra s'asseca mes rapidament.
    if temperatura_placa > temperatura_gui:
        prioritat += 1

    if humedad_ambient_placa < humetat_hambient_gui:
        prioritat += 1
    print(f"prioritat: {prioritat}")
    # Despres de calcula el numero de prioritat, el que fem es comprovar si
    # la prioritat ens dius si es necessari regar.
    if prioritat >= 2:
        return True
    else:
        return False
```

Codi 26. Funció regadiu

9 Avaluació

Per tal de comprovar que totes les funcionalitats d'aquest projecte funcionin correctament, s'ha elaborat conjunt extens de proves. Aquest s'ha estructurat en diversos apartats, que es descriuran a continuació.

9.1 Lectura dels sensors

Descripció	Resultats esperats	Estat
Obtenir tots els valors dels sensors ambientals.	El dispositiu encastat mostra els valors recollits per pantalla	Correcte.
Simular valors fora de rang.	El sistema detecta error i mostra valors sentinella.	Correcte.
Generar modificacions ambientals.	S'ha de poder observar els canvis dels valors en temps real.	Correcte

Taula 1. Lectura dels sensors

9.2 Comunicació al sistema central

Descripció	Resultats esperats	Estat
Establir una connexió entre el node arrel i el sistema central.	El node arrel s'ha de poder connectar al punt d'accés del sistema central.	Correcte.

El sistema central s'engega per obtenir les dades.	El node arrel envia les dades dels sensors i el sistema central rep aquestes dades.	Correcte.
Tanca la connexió durant l'execució	El sistema mostra un error de pèrdua de connexió.	Correcte.
Després de la pèrdua de connexió, es torna a executar el sistema central.	El node arrel es torna a connectar al sistema automàticament.	Correcte.
Es modifica manualment la ***IP*** del sistema central.	El node arrel busca el sistema dins la xarxa.	Incorrecte. El node arrel no és capaç de fer una cerca del sistema.

Taula 2. Comunicació al sistema central

9.3 Proves d'interfície gràfica

Descripció	Resultats esperats	Estat
Afegeix llindars a la configuració.	Es recull aquests llindars i s'emmagatzemen al sistema.	Correcte.
Es modifica els llindars.	Es recull aquests llindars i es modifiquen pels antics dins del sistema central.	Correcte.
Introducció invalida dels llindars.	Mostra un missatge d'error explicant el problema.	Correcte.
Introducció invalida del format en la configuració de la ciutat	Mostra un missatge d'error explicant el problema.	Correcte
Introducció de una ciutat no existent.	Mostra un missatge d'error.	Incorrecte, aquet sistema no ho te en compte.

Taula 3. Proves d´interfície gràfica

9.4 Proves de prediccions meteorològiques i fiabilitat

Descripció	Resultats esperats	Estat
Sol·licitar les prediccions meteorològiques de les dues fonts externes.	El sistema s'ha de rebre les dues prediccions meteorològiques.	Correcte.
Comprovar la predicció.	Depenent del resultat es suma o resta la taxa d'encerts.	Correcte.
Rebre predicció de pluja i una taxa d'encerts positiva.	El sistema no ha d'activar el reg.	Correcte.
Rebre predicció de pluja i una taxa d'encerts negativa.	El sistema descarta la font externa temporalment.	Correcte

Modificar manualment les taxes d'encerts.	El sistema s'adapta automàticament al pes de cada font.	Correcte.

Taula 4. Proves de prediccions meteorològiques i fiabilitat

9.5 Proves de decisions del reg

Per fer aquest joc de proves es descarten les prediccions meteorològiques.

Descripció	Resultats esperats	Estat
Humitat del sòl més baixa que el llindar.	Com el pes de la humitat del sòl es el mes alt activa directament el reg.	Correcte.
Tots els paràmetres estan dins del llindar.	No s'activa el reg.	Correcte.
Humitat ambient i lluminositat estan més baix del llindar.	S'activa el reg.	Correcte.
Força la prioritat a 2 o més.	El sistema de reg s'activa	Correcte
La humitat ambient és més baixa del llindar però els altres paràmetres estan correctes.	El sistema comprova les prioritats i veu que la prioritat no és prou alta.	Correcte.

Taula 5. Proves de decisions del reg

9.6 Proves de comunicació entre nodes

En primer lloc, cal destacar que tota la implementació de la xarxa ***Mesh*** ha estat desenvolupada i es troba plenament funcional. Tanmateix, no s'ha disposat de la capacitat de provar-la amb diversos nodes simultanis per fer la comprovació.

Descripció	Resultats esperats	Estat
Node arrel crea la xarxa ***Mesh***	El node arrel ha de crear i s'ha de poder veure la xarxa creada	Correcte.
Els nodes fills es connecten en aquesta xarxa.	Els nodes es connecten en aquesta xarxa sense inconvenients	Correcte.
Els nodes envien tota la informació la xarxa.	Tots els nodes després de la connexió envien les dades des sensors a la xarxa.	Com no tenim varis nodes no es pot comprovar.

Taula 6. Proves de comunicació entre node

10 Conclusions

10.1 Conclusions sobre l'aplicació

Aquet projecte ha permès desenvolupar un sistema de reg intel·ligent basat en tecnologia ***IoT***, orientat a optimitzar el consum d'aigua en plantacions agrícoles. Aquest ha integrat una xarxa de nodes encastats amb sensors ambientals , una arquitectura ***Mesh*** i un sistema central capaç de recollir totes les dades obtingudes dels nodes encastats i analitzar-les per a una presa de dedicions autònoma. Per altra banda, aquest sistema central obté diverses prediccions meteorològiques amb una alta fiabilitat, les quals es tindran en compte en el moment de prendre decisions sobre l'activació del reg.

Els resultats obtinguts no sol demostren que el sistema no nomes es funcional en condicions reals, sinó que també aporta una millora amb la optimització d'un recurs tant essencial com es el aigua.

Tanmateix, aquest sistema també te varies limitacions importants, com es el un d'internet, en cas de no haver-hi no es podran fer les consultes pertinents sobre les prediccions i la precisió de la decisió disminuirà notablement. Un altre punt important es el manteniment i calibratge dels sensors ambientals, ja que al llarg del temps serà necessari fer aquestes accions.

En aquest projecte es deixen les portes obertes per fer altres millores que podrien augmentar la precisió i la fiabilitat de la decisió final, com ara ampliar la compatibilitat en altres sensors ambientals per obtenir mes informació que ajudi amb aquesta decisió o implementar el sistema central en un dispositiu mòbil ja que augmentaria considerablement la comoditat del usuari final.

En definitiva aquest projecte representa una solució factible, escalable i eficient per a una gestió autònoma del reg en plantacions agrícoles, amb una alta adaptació depenent del àmbit agrícola.

10.2 Conclusions personals

El desenvolupament d'aquest projecte ha representat una oportunitat per aplicar de manera pràctica els coneixements adquirits al llarg del grau, alhora que ha permès afrontar reptes tècnics reals amb un alt grau d'autonomia i capacitat de resolució. L'experiència ha estat valuosa tant des d'un punt de vista professional com personal.

Durant tot aquest procés, he hagut de aprofundir en àmbits de programació en sistemes encastats, gestió i creació de xarxes de comunicació que s'adaptin correctament en aquest projecte, l'ús d'***APIs*** externes. També ha estat una oportunitat per aprendre a documentar, plantificar i justificar en tot el transcurs d'un projecte.

En definitiva, es considera que els objectius plantejats inicialment han estat assolits amb èxit. El sistema resultant destaca per la seva autonomia, flexibilitat i fiabilitat en la gestió de l'aigua, alhora que deixa oberta la porta a futures millores o aplicacions complementàries. Aquest treball ha contribuït a demostrar que la tecnologia pot oferir solucions eficients i sostenibles per a la gestió hídrica en l'àmbit agrícola.

11 Referències

[1] Nathan Seidle, “SparkFun Thing Plus - ESP32-S2 WROOM,” https://www.sparkfun.com/sparkfun-thing-plus-esp32-s2-wroom.html. Accessed: Jun. 03, 2025. [Online]. Available: https://www.sparkfun.com/sparkfun-thing-plus-esp32-s2-wroom.html

[2] OpenWeather Ltd., “OpenWeather API,” https://openweathermap.org/api. Accessed: Jun. 03, 2025. [Online]. Available: https://openweathermap.org/api

[3] OPLAO FZCO, “WeatherAPI,” https://www.weatherapi.com/. Accessed: Jun. 03, 2025. [Online]. Available: https://www.weatherapi.com/

[4] Lucid Software Inc., “Instral·lacio i utilitzacio del generador de diagrames LucyChard,” https://www.lucidchart.com/pages. Accessed: Jun. 03, 2025. [Online]. Available: https://www.lucidchart.com/pages/landing/diagram-software?utm_source=google&utm_medium=cpc&utm_campaign=_chart_en_tier1_mixed_pmax_strategic_&km_CPC_CampaignId=17817470124&km_CPC_AdGroupID=&km_CPC_Keyword=&km_CPC_MatchType=&km_CPC_ExtensionID=&km_CPC_Network=x&km_CPC_AdPosition=&km_CPC_Creative=&km_CPC_TargetID=&km_CPC_Country=9049225&km_CPC_Device=c&km_CPC_placement=&km_CPC_target=&gad_source=1&gad_campaignid=17817597735&gclid=Cj0KCQjwucDBBhDxARIsANqFdr2Ukb8lhoHMDsSodY6ct9zpxNyKZwi__T5MRH8DT4GKjjkrEUelSc4aAjqKEALw_wcB

[5] Douglas Crockford, “Introduccio JSON,” https://www.json.org/json-en.html.

[6] Standard C++ Foundation, “Standard C++,” https://isocpp.org/. Accessed: Jun. 03, 2025. [Online]. Available: https://isocpp.org/

[7] PlatformIO Labs OÜ, “PlatformIO,” https://platformio.org/.

[8] Microsoft, “Visual Studio Code,” https://code.visualstudio.com/.

[9] Python Software Foundation, “Python,” https://www.python.org/. Accessed: Jun. 03, 2025. [Online]. Available: https://www.python.org/

[10] Nathan Seidle, “Sensor d’Humitat del Sòl,” https://www.sparkfun.com/sparkfun-soil-moisture-sensor.html.

[11] Nathan Seidle, “Sensor de lluminositat,” https://www.sparkfun.com/sparkfun-ambient-light-sensor-breakout-temt6000.html. Accessed: Jun. 03, 2025. [Online]. Available: https://www.sparkfun.com/sparkfun-ambient-light-sensor-breakout-temt6000.html

[12] Nathan Seidle, “Esquema ESP32-S2,” https://cdn.sparkfun.com/assets/a/d/0/6/d/ESP32-S2_thing_plus_schematic_2.pdf.

[13] Nathan Seidle, “Esquema sensor d’humitat del sòl,” https://cdn.sparkfun.com/datasheets/Sensors/Biometric/SparkFun_Soil_Moisture_Sensor.pdf. Accessed: Jun. 03, 2025. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Biometric/SparkFun_Soil_Moisture_Sensor.pdf

[14] o0timbo0o, “Conversió de senyal a persentatge,” https://forum.arduino.cc/t/baffled-by-ranging-an-analog-value-into-a-percentage/364834.

[15] David Williams, “Design a Luxmeter Using a Light Dependent Resistor,” https://www.allaboutcircuits.com/projects/design-a-luxmeter-using-a-light-dependent-resistor/.

[16] Python Software Foundation, “tkinter - Python interface to Tcl/Tk,” https://docs.python.org/3/library/tkinter.html.

12 Annexes

12.1 Guia d'instal·lació

A continuació, es presenta una guia per a la instal·lació i funcionament del projecte. Aquest es divideix en dos programes principals: el sistema de nodes encastats i el sistema central. En primer lloc, es descriurà la guia d'instal·lació del sistema de nodes encastats.

1. Descarregar i instal·lar ***Visual Studio Code****[8].*
2. Dins del propi ***Visual Studio Code****[8],* hi ha un apartat de extensió. En aquest s'ha de buscar e instal·lar la extensió ***PlantformIO****[7]***.** És essencial obtenir aquesta extinció per a l'execució d'aquet sistema dins d'un node encastat.
3. Descarrega el zip corresponent i implementar aquet projecte en el ***Visual Studio Code****[8]***.**
4. Connectar el dispositiu ***ESP32****[1]* amb els sensors ja implementats en l'ordinador.
5. Un cop obert el projecte a ***Visual Studio Code****[8]* i amb l'extensió ***PlatformIO****[7]*, instal·lada correctament, cal compilar i carregar el projecte al dispositiu encastat.
6. Un cop el projecte hagi estat carregat al dispositiu, es podrà observar com aquest intenta connectar-se al sistema central. A continuació, es crearà la xarxa ***Mesh*** i els sensors començaran a recollir dades.

Aquest procediment s'haurà de repetir des del punt 3 per cada dispositiu encastat.

Per tal d'executar el sistema central, cal seguir els passos següents:

1. Descarregar i instal·lar ***Visual Studio Code****[8].*
2. Descarregar el zip corresponent i implementar aquest projecte dins de ***Visual Studio Code****[8]*.
3. Crear un punt d'accés des de l'ordinador, per tal que el sistema de nodes encastats pugui connectar-s'hi i enviar la informació:
    a. Accedir a "**Configuració > Red e Internet > Zona amb cobertura inalàmbrica mòbil**".
    b. En aquest punt, s'han de fer diverses configuracions. La més important és editar la connexió i establir els valors següents:
        i. **Nom de la xarxa**: AutoReg.
        ii. **Contrasenya de la xarxa**: adminAutoReg
        iii. **Banda de la xarxa**: 2.4 Ghz
    c. Finalment, cal activar la configuració, i el punt d'accés es crearà automàticament.
4. Un cop el projecte s'hagi importat a ***Visual Studio Code****[8]***,** cal compilar-lo i executar-lo per iniciar el funcionament del sistema central.