UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Proyecto de automatización de temperatura y humedad de un terrario para reptiles

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Fernández Robles, Víctor

Tutor: Bernabeu Soler, Enrique Jorge

Curso 2019/2020

Agradecimientos

A mi padre y a mi madre por el apoyo incondicional que me han dado toda la vida.

A mis amigos por soportarme los días que no funcionaban las cosas, especialmente a Bogdan y a Conrado por no librarse ni en cuarentena con las videollamadas.

A mi tutor por la ayuda recibida durante la realización del proyecto.

A mi hermano por motivarme y relajarme desde la distancia.

Y en especial a mi abuelo, por ser mi fuente de inspiración cuando las cosas no salen.

Resumen

En este proyecto se realiza un sistema para la automatización de terrarios para reptiles, el cual consiste en el control de la temperatura y la humedad, como elementos fundamentales, además de la radiación UVB, la alimentación y la hidratación. Para la creación del sistema de control se emplean unos sensores o *inputs*, encargados de captar información acerca de los factores a automatizar, unos dispositivos actuadores u *outputs*, que se encargan de actuar sobre dichos factores, y por último una Raspberry Pi, que contiene el código con la lógica para recibir la información desde los sensores, procesarla y, si fuera necesario, actuar sobre el medio a través de los actuadores. La interacción con el sistema de automatización se realiza a través de una aplicación Android para móviles, desde la cual se pueden modificar los valores relacionados con las magnitudes a automatizar, y la aplicación, mediante sockets, se encarga de informar de dichos cambios al sistema de automatización.

Palabras clave: terrario, reptiles, humedad, temperatura, aplicación, actuadores, sensores, Raspberry Pi, radiación UVB, Android.

Abstract

In this project, a system for the automation of reptile terrariums has been developed consisting of the control of temperature and humidity, as fundamental elements, in addition to UVB radiation, food and hydration. For the creation of the control system, sensors or inputs are used, in charge of capturing information about the factors to be automated, actuator devices or outputs, which are responsible for acting on these factors, and finally a Raspberry Pi, which contains the code with the logic to receive the information from the sensors, process it and, if it is necessary, act on the environment through the actuators. The interaction with the automation system is carried out through an Android mobile application, from which the values related to the factors to be automated can be modified, and the application, through sockets, is responsible for reporting such changes to the automation.

Keywords: Terrarium, reptiles, humidity, temperature, application, outputs, inputs, Raspberry Pi, UVB radiation, Android.

Índice

Índice de imágenes

1. Introducción

Los reptiles son animales muy particulares, una de sus características principales es que son animales de sangre fría, esto no significa que no se ponen nerviosos ante situaciones tensas, sino que su temperatura corporal se adapta al clima en el que se encuentran. Para que se entienda mejor, si están en un entorno frio su temperatura corporal es fría y si están en un entorno más caluroso su temperatura corporal también es más alta. Dicha particularidad repercute en su dinamismo, es decir, cuando se encuentran en lugares con temperaturas altas están más activos, y por el contrario con temperaturas bajas están apacibles.

Las particularidades climáticas de estos animales no se quedan en la temperatura, la humedad también es un factor muy importante para ellos. La hidratación de los reptiles no se resume en un gesto tan simple como beber agua, muchos de ellos obtienen agua de los alimentos que digieren y sobretodo se mantienen hidratados por la humedad del ambiente.

Pero no todo van a ser distinciones, los reptiles como los humanos y otros animales obtienen beneficios de la luz solar, en concreto de los rayos UVB.

¿Se imaginan que la temperatura marcara radicalmente el ritmo de nuestras vidas? ¿O que por un nivel bajo de humedad estuviéramos sedientos todo el día? Seguro que buscaríamos la forma de ponerle solución. Los reptiles no han evolucionado tanto como nosotros así que en este proyecto trataremos de echarles una mano con sus peculiares características.

1.1 Motivación

Al terminar todas las asignaturas del grado de ingeniería informática hice un repaso de mi época como universitario, me gusta pensar que ha sido y será la mejor época de mi vida. Nuevas amistades, días de estudiar, días de fiesta, encerrarse en la biblioteca en época de exámenes, largas noches de estudio, café como forma de supervivencia, tardes de hacer trabajos con los compañeros y tardes de quedar a tomar algo con ellos, cenas de clase post-exámenes, un erasmus inmejorable, conocer a gente de todas partes, perderte por una ciudad que no conoces, momentos felices tras las notas y otros no tanto, pero en general experiencias que hacen madurar.

Por supuesto también hice un repaso de lo aprendido a nivel académico, en este grado se abarcan muchos aspectos distintos de la informática, y esto me llevo a pensar si realmente sabía aplicar todo lo aprendido. Este pensamiento me llevo a decantarme por dos opciones para mi trabajo final de grado, realizar un trabajo de un concepto que esta fuera de mi rama y que me parece realmente interesante como es el *Machine learning*, y así comprobar si todo lo aprendido en la carrera me servía de herramienta para adaptarme a un aspecto de la informática y la programación ajeno a mis conocimientos, o realizar un trabajo relacionado con la rama que cursé, ingeniera de computadores, y tratar de mezclar distintos conceptos aprendidos en la carrera.

Finalmente me decanté por la segunda opción, realizar un proyecto de automatización, ya que es un tema que también me resulta muy interesante y decidí que hay mejores formas para iniciarse con el *Machine learning* que realizar un trabajo final de grado. Dicho proyecto incluye contacto con tecnología no vista directamente durante el grado, desarrollo de aplicaciones en Android y diferentes protocolos de conexión entre computadores.

1.2 Objetivos

El objetivo de este trabajo final de grado es el de realizar un sistema de automatización de terrarios para reptiles, con el fin principal de atender a las necesidades climáticas y nutricionales que necesitan los reptiles para vivir, así como permitir a los dueños de estos animales controlar dichos cuidados de una manera sencilla desde una aplicación móvil. Para ello se deben perseguir unos objetivos más específicos:

- Conocer en todo momento la temperatura y humedad en la que habita el reptil.
- Disponer de un mecanismo para detectar los niveles de agua y comida, así como para el suministro de los mismos.
- Detectar la presencia o no de luz solar, más concretamente de rayos UVB.
- Crear una aplicación móvil, en base Android, para mostrar los datos obtenidos, ajustar niveles de temperatura y humedad, y guardar horarios de comida y horas de luz.
- Actuar sobre los factores de temperatura y humedad acorde a los valores guardados en la aplicación.
- Suministrar luz UVB siempre que no se detecte la luz natural y acorde a los horarios establecidos en la aplicación.

1.3 Estructura de la memoria

En primer lugar, se expondrán diferentes productos que en este momento ocupan la necesidad que intentamos cubrir con este proyecto, para así reflejar las diferencias entre los productos ya existentes y nuestro producto. Después de esto se realiza un análisis exhaustivo del problema, seguido de una explicación de los requisitos a cubrir para solventarlo y de una exposición de los diferentes dispositivos necesarios para dicha tarea. Una vez se conocen todos los dispositivos necesarios, se realiza una propuesta para la solución del problema apoyándonos en dichos dispositivos, y para cerrar este apartado, se incluye el presupuesto para la obtención de los materiales necesarios.

A continuación, se procede con el diseño de la solución, donde nos desmarcamos de la parte hardware y se pasa a exponer las decisiones acerca de cómo se va llevar a cabo la solución al problema.

En primer lugar, se muestran las diferentes arquitecturas software que se van a emplear y como se realizara la comunicación entre ellas, mostrando también un diagrama de flujo para presentar la lógica. Para finalizar este apartado se expondrán las diferentes tecnologías empleadas para realizar la parte software, desde plataformas para el desarrollo hasta librerías empleadas.

En el siguiente punto se explica con detalle el desarrollo de la solución propuesta, comenzando por la aplicación móvil, siguiendo con el sistema de automatización y acabando con el desarrollo de la parte hardware.

Una vez explicado todo el desarrollo del proyecto, se mostrará como implantar el sistema operativo empleado para realizar el trabajo. Seguido de, las pruebas realizadas para verificar que nuestro producto funciona correctamente, tanto la parte hardware como la software. A continuación, tendremos las conclusiones obtenidas de nuestro trabajo, donde se explica si se ha obtenido el resultado esperado y qué se ha aprendido durante la realización de este trabajo, tanto en la parte personal como profesional.

Para terminar, se expondrán una serie de trabajos futuros que hemos pensado mientras se trabajaba en el proyecto, y que no se han podido realizar.

2. Estado del arte

Se han buscado tecnologías actuales que cubren la necesidad que intentamos satisfacer con el producto de este proyecto, hemos encontrado diferentes alternativas. Podemos destacar las siguientes tres, pero estos productos no están destinados al mismo objetivo, cubren únicamente las necesidades relacionadas con la temperatura y humedad, y no proporcionan una plataforma para su gestión.

- **Hygro Therm**:

  Se trata de un dispositivo [1] capaz de automatizar la humedad y la temperatura de un terrario, dispone de 1000W para la conexión de humidificadores, calefactores o ventiladores, lo que significa que el dispositivo sin complementos solo es capaz de medir el nivel de estas dos variables. Con este dispositivo existe la posibilidad de programar una disminución de la temperatura por la noche, a través de una fotocélula integrada. Opera con un rango de temperaturas de entre 10 y 50 grados centígrados, y un rango de humedad de entre el 15 y el 95 por ciento. El precio de este producto es de 146’31 euros sin complementos.

Figura 1.1 Dispositivo Hygro Therm

○ Terrarios YCDJCS

Se trata de un terrario para reptiles [2] capaz de controlar la temperatura de forma inteligente, incorpora una pantalla para poder programar la temperatura, en la cual se indica, además, el nivel actual de la temperatura y la humedad. El precio oscila entre los 100’72 y los 711’86 euros, dependiendo del tamaño y el material de fabricación.

Figura 1.2 Terrario YCDJCS

○ Terrarios Zoo Med

Existe una gran variedad de terrarios de esta marca [3], se trata de terrarios de un tamaño y con unos accesorios determinados para cada tipo de reptil. Existe un gran variedad y entorno al 70% disponen de medidor de temperatura y humedad. El precio oscila entre los 120 y los 330 euros.

Figura 1.3 Terrario Zoo Med

2.1 Crítica al estado del arte

Las alternativas mencionadas en el apartado anterior están relacionadas con el factor más importante para el bienestar de los reptiles, como es la temperatura y humedad de su entorno, no obstante, no proporcionan una solución completa. Los terrarios encontrados solo muestran la temperatura y humedad, y en el mejor de los casos disponen de un calefactor para aumentar la temperatura si fuera necesario. En el caso del dispositivo Hygro Therm, es un complemento para terrarios que necesita de más complementos para cumplir con su función al completo, y pese a que es lo más parecido al sistema que estamos desarrollando, solo aborda las necesidades de temperatura y humedad. Por lo tanto, para conseguir un terrario que automatice estos factores climáticos habría que gastar una cantidad mínima de más de 250 euros y no cubriría todas las necesidades del reptil, ya que la radicación solar es muy necesaria y, aunque algunos tipos de reptiles puedan aguantar sin beber o comer durante días, si nos ausentamos durante un largo periodo de tiempo estas alternativas no bastarían.

2.2 Propuesta

La propuesta que se realiza en este proyecto tiene el objetivo de no solo garantizar las condiciones climáticas de los reptiles sino también las nutricionales, permitiendo a los dueños poder gestionar estas necesidades de una manera sencilla desde una aplicación móvil. De esta manera, si los dueños han de ausentarse, su mascota tendrá todo lo necesario para subsistir. Otro aspecto positivo es que el sistema no es particular para un tipo de reptil, ya que se pueden adaptar las condiciones según las necesidades que necesite el reptil que queremos cuidar. Las características que presenta este proyecto tendrían buena utilidad en tiendas de mascotas, donde se podría controlar las condiciones de los terrarios de cada tipo de reptil desde la aplicación.

En definitiva, es la unión entre el Hygro Term y una aplicación móvil, más un sistema para el control de la alimentación, la hidratación y la radiación UV necesaria.

3. Análisis del problema

Al adquirir animales como los reptiles se puede caer en el error de desentenderse de algunos de sus cuidados por el hecho de que en general son animales muy resistentes y que pueden aguantar prolongadas cantidades de tiempo sin alimentarse. Esta situación puede llegar a hacer enfermar a los reptiles.

En primer lugar, los reptiles son animales que se adaptan al medio en el que habitan y puede dar la sensación de que por ello se encuentran en un estado de salud óptimo. Sin embargo sin unas condiciones de temperatura y humedad concordes con el tipo de reptil las reacciones metabólicas[1] que tienen lugar en el organismo se verían alteradas [4] . Un caso parecido ocurre con la radiación UVB [5], se trata de rayos ultravioleta B de onda corta que forma parte de la energía que viene del sol, los cuales todos los reptiles diurnos necesitan recibir en su piel para poder sintetizar la vitamina D3, que es necesaria para absorber el calcio de los alimentos. Sin esta vitamina los animales se descalcifican[2] y pueden llegar a padecer una enfermedad ósea metabólica[3], que puede llegar a ser muy grave en casos avanzados. Las lámparas incandescentes comunes y de amplio espectro no emiten rayos UVB, además, estos rayos no atraviesan el cristal ni el plástico. Por último, la alimentación e hidratación también es fundamental y en ocasiones se descuida más de lo que se debería.

3.1 Especificación de requisitos

Conociendo las necesidades generales de los reptiles, mencionadas en el apartado anterior, se muestran las especificaciones que debe disponer nuestro proyecto:

- El terrario debe contar con unas condiciones de temperatura adecuadas.
- El terrario debe contar con un nivel de humedad adecuado.
- El terrario debe suministrar alimento al reptil en los horarios que se indican desde la aplicación.
- El reptil debe disponer de agua en todo momento.
- El reptil debe disponer de radiación UVB durante las horas que se indiquen en la aplicación.
- Los datos referentes a las condiciones actuales del terrario han de estar reflejados en la aplicación.

[1] **Reacciones metabólicas:** Conjunto de reacciones químicas que tienen lugar en las células del cuerpo y cuya función es convertir el alimento ingerido en combustible necesario para la actividad de los seres vivos.

[2] **Descalcificar:** Hacer perder la sustancia calcárea contenida en los huesos u otros tejidos orgánicos.

[3] **Enfermedad ósea metabólica:** La enfermedad ósea metabólica (EOM), es una de las enfermedades más comunes en reptiles. También es conocida como hiperparatiroidismo nutricional secundario, osteoporosis, osteodistrofia fibrosa, o raquitismo, y es probablemente el problema nutricional más visto en los reptiles que acuden a la clínica de exóticos.

3.2 Identificación y análisis de soluciones posibles

En el siguiente apartado se van a exponer los diferentes dispositivos hardware necesarios para la realización del proyecto y algunas alternativas que también se podrían utilizar. Se indicarán los pros y contras de cada alternativa, y se señalará cual es la más adecuada para cada aspecto. Hay que tener en cuenta que, pese a que se va a mostrar una opción para cada dispositivo requerido, no se ha podido contar con todos los dispositivos que nos gustarían y en algunos casos se emplearan otros dispositivos, como ledes, para mostrar el funcionamiento del sistema a modo de simulador no software.

3.2.1 Placa base

Empezaremos por el dispositivo que hace de cerebro de todo el proyecto. Estas placas son mini computadores donde se conectan todos los sensores y actuadores necesarios. Su cometido principal en este proyecto será la recepción e interpretación de los datos recibidos por los sensores y la actuación pertinente a través de los actuadores. Disponen de una serie de pines para conectar todos los dispositivos. Existen diferentes tipos de pines y se pueden organizar en tres grandes grupos, pines de tierra (GND), pines de alimentación (VCC) y pines de salida de datos, estos últimos son los encargados de manejar el comportamiento del dispositivo, activarlos o desactivarlos y recibir la información que recopilen en el caso de los sensores.

- **Arduino**

Probablemente la placa más conocida en la actualidad sea la placa de Arduino [6]. Arduino es una plataforma electrónica de código abierto basada en hardware y software, para la creación de prototipos. Está basada en una tarjeta con un microcontrolador que permite conectar sensores, actuadores y otros elementos mediante sus entradas y salidas, analógicas y digitales. Dispone de su propio entorno de programación llamado Arduino IDE.

Especificaciones técnicas: [7]

- Microcontrolador: ATMega328
- Voltaje de operación: 5V
- Voltaje de alimentación: 7-12V
- Pines digitales I/O: 14
- Pines de entrada analógica: 6
- Programación: Por medio de USB
- Frecuencia de reloj: 16Mhz

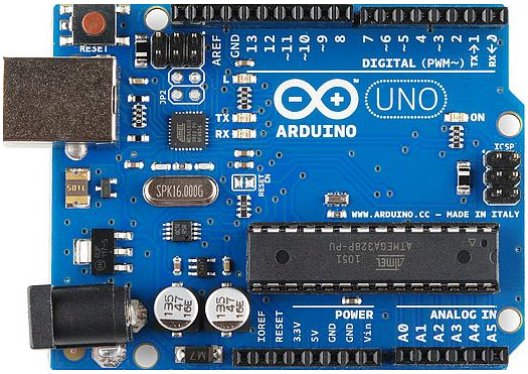


Figura 2.1 Placa de Arduino

Raspberry Pi

Se trata de una placa base del tamaño de una tarjeta de crédito en la que se monta un ordenador, provisto principalmente de un procesador, un chip gráfico, una memoria RAM y una ranura para una tarjeta SD en la que se debe descargar el sistema operativo que se desee utilizar. Consta también de una serie de pines digitales para la conexión con sensores y actuadores. Esta placa será la utilizada en este proyecto, en concreto la Raspberry Pi 3B+.

Figura 2.2 Placa Raspberri Pi 3B+

Especificaciones técnicas: [8]

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC 1.4GHz.
- RAM: 1GB LPDDR2 SDRAM
- Wi-Fi + Bluetooth: 2’4GHz y 5GHz IEEE 802.11. b/g/n/ac, Bluetooth 4.2, BLE
- Ethernet: Gigabit Ethernet sobre USB2.0 (300 Mbps)
- GPIO de 40 pines
- HDMI
- 4 puertos USB 2.0
- Puerto CSI para conectar una cámara.

- Puerto DSI para conectar una pantalla táctil
- Salida de audio estéreo y vídeo compuesto
- Micro-SD
- Power-over-Ethernet (PoE)

VENTAJAS Y DESVENTAJAS DE LA ELECCIÓN

- **Ventajas**
  - Fácil conexión vía Wifi
  - Más económica que la placa de arduino
  - Más potente que la placa de arduino
  - Se puede elegir el sistema operativo
  - Se puede elegir el lenguaje de programación

- **Desventajas**
  - No dispone de pines analógicos, por lo que habría que comprar material adicional si necesitáramos convertir un pin digital en analógico.
  - Se puede llegar a sobrecalentar y para evitarlo se necesitan ventiladores y/o disipadores de calor.

3.2.2 Sensores

Los sensores son dispositivos capaces de detectar variaciones de diferentes magnitudes del medio en el que se encuentran. Para ello tienen una propiedad sensible a una magnitud específica, la cual varía cuando lo hace la magnitud que se desea medir con dicho sensor.

3.2.2.1 Humedad y temperatura

Vamos a comenzar con el sensor de temperatura y humedad, por suerte hemos encontrado sensores que miden estas dos magnitudes y no ha hecho falta buscar uno para cada magnitud. Se trata de un sensor fundamental para la realización del proyecto ya que para garantizar el adecuado cuidado del clima de un reptil se necesita controlar en todo momento el estado de estas magnitudes.

- **DHT22**

Se trata de un sensor con señal de salida digital, lo cual encaja a la perfección con la Raspberry Pi que solo dispone de pines digitales. Está compuesto por un sensor capacitivo para medir la humedad y de un termistor[4] para medir la temperatura. Consta de 4 pines, el pin de GND que se debe conectar al pin de tierra de la placa base, el pin de VCC que se debe conectar al pin de 5V de la placa base, el pin de DATA o de salida de datos, que se debe conectar a uno de los pines GPIO de salida de datos de la placa base y es por donde se reciben y tratan los datos, y el pin NC (*Not Connected*), el cual no tiene un propósito funcional para el circuito externo pero pueden tener una funcionalidad no expuesta, por lo tanto no deben conectarse a ninguna de las conexiones del circuito. Ha de equiparse con una resistencia que se debe colocar entre la alimentación y la salida de datos como se muestra en la figura 2.3.

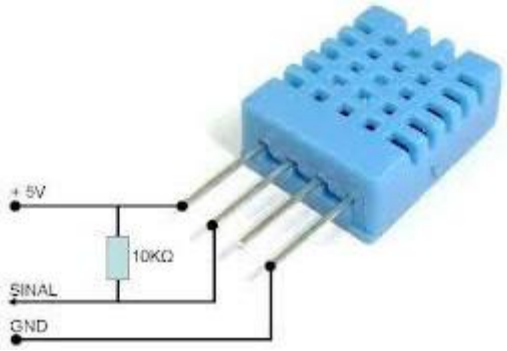


Figura 2.3 Esquema conexión DHT22

Especificaciones técnicas: [9]

- Alimentación: 3.3Vdc ≤ Vcc ≤ 6Vdc
- Señal de salida: Digital
- Rango temperaturas: -40°C a 80 °C
- Precisión medida: <±0.5 °C
- Resolución temperatura: 0.1°C
- Rango de medida de humedad: De 0 a 100% RH
- Precisión humedad: 2% RH
- Resolución humedad: 0.1% RH
- Tiempo de respuesta: 2s
- Tamaño: 14 x 18 x 5.5mm

[4] **Termistor:** Tipo de resistencia cuyo valor varía en función de la temperatura.

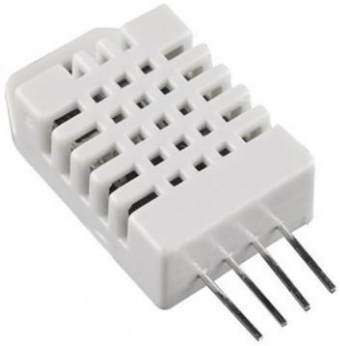


Figura 2.4 Sensor DHT22

DHT11

El sensor de temperatura y humedad DHT11 es la versión inferior al DHT22, por lo que las características generales son las mismas, la distribución de pines es la misma, se trata de un sensor digital y contiene un sensor capacitivo y un termistor.

Especificaciones técnicas: [10]

- Alimentación: 3Vdc ≤ Vcc ≤ 5Vdc
- Señal de salida: Digital
- Rango temperaturas: 0C a 50 °C
- Precisión medida: <±2 °C
- Resolución temperatura: 0.1°C
- Rango de medida de humedad: De 20 a 90% RH
- Precisión humedad: 4% RH
- Resolución humedad: 1% RH
- Tiempo de respuesta: 1s
- Tamaño: 12 x 15.5 x 5.5mm

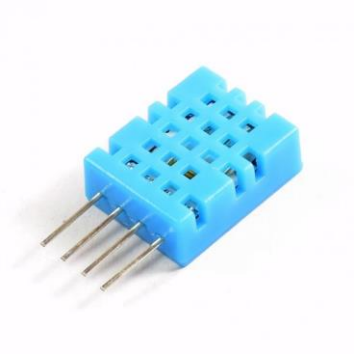


Figura 2.5 Sensor DHT11

Se puede observar que por condiciones el DHT22 tiene mejores características, no obstante, se ha elegido el DHT11 principalmente porque para el trabajo que estamos realizando esa diferencia de

condiciones no afectan de forma notable, el ciclo de operación del DHT11 es menor, por lo que es más rápido, y el precio es también menor. Además, existe más contenido sobre programación en DHT11 que en DHT22 la cual cosa facilita la búsqueda.

VENTAJAS Y DESVENTAJAS DE LA ELECCIÓN

- **Ventajas**
  - Fácil de conectar y fiable
  - Menor precio
  - Facilidad en la búsqueda de información para programarlo
  - Ciclo de operación menor
- **Desventajas**
  - Rangos de medida menos amplios
  - Rangos de precisión en las medidas más amplios

En nuestro caso el sensor DHT11 adquirido incorpora una resistencia integrada (Figura 2.6), por lo que cada pin debe ir a su pin correspondiente en la Raspberry Pi, sin necesidad de conectarlo a una resistencia.

Figura 2.6 DHT11 con resistencia incorporada

3.2.2.2 Nivel del agua

Este sensor será el encargado de detectar el nivel de cantidad de agua de la que dispone el reptil, de esta manera se podrá cumplir la necesidad de la disposición permanente de agua dentro del terrario.

- **Water Level Detection Sensor Module 1PC**

Se trata de un sensor analógico para la detección del nivel del agua, consta de 3 pines, el pin (-) GND de tierra, el pin (+) VCC que debe conectarse a 5V y el pin (S) de señal que debe conectarse a los pines GPIO de salida de datos. Se trata de un sensor alargado que detecta a lo largo de su superficie a que nivel está el agua en todo momento y habría que programarlo para que llegado cierto nivel se informara al usuario mediante la aplicación móvil que el agua está a un nivel demasiado bajo.

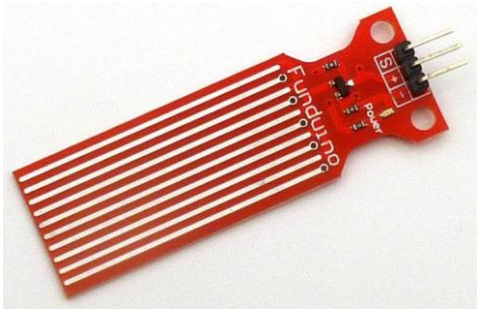


Figura 2.7 Water Level Detection Sensor Module 1PC

Especificaciones técnicas: [11]

- Alimentación: 3-5V
- Área de detección: 40 mm x 16 mm
- Corriente: menos de 20 mA
- Humedad: 10% - 90%
- Dispositivo analógico

El problema que trae este sensor es la complejidad añadida de que se trata de un pin analógico y, por lo tanto, puesto que la Raspberry Pi no dispone de pines para este tipo de dispositivos, habría que utilizar otro dispositivo llamado MCP3008 para permitirnos trabajar con él. El dispositivo MCP3008 es un convertidor analógico-digital de 16 pines y que funciona a una tensión máxima de 5.5V. Este supuesto también conllevaría una dificulta añadida en el aspecto de la programación.

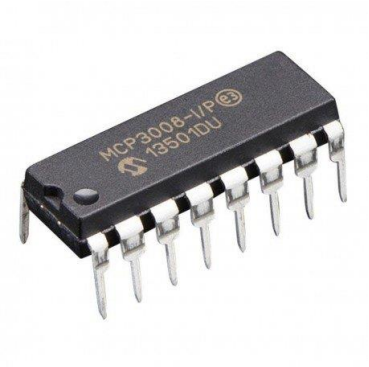


Figura 2.8 Chip MCP3008

- **Sensor XKC-Y25-V**

Se trata de un sensor de nivel de líquidos sin contacto, es decir, no es necesario colocarlo dentro del recipiente que contiene el agua. Esto es gracias a que utiliza tecnología avanzada de procesamiento de señales mediante el uso de un chip con una velocidad de operación alta. Este

sensor permite trabajar hasta con recipientes con un espesor máximo de 20 mm, siempre que no sean metálicos. Para regular el punto de referencia incorpora un potenciómetro ajustable mediante un tornillo, de esta manera podremos ajustarlo de tal forma que cuando el agua no se encuentre en ese punto el sensor lo detecte.

Especificaciones técnicas: [12]

- Alimentación: 5-24V
- Corriente: 5mA
- Tension de salida (nivel alto): InVcc
- Voltaje de salida (nivel bajo): 0V
- Corriente de salida: 1 – 50mA
- Tiempo de respuesta mínimo: 500mS
- Temperatura de trabajo: 0 – 105 Grados Celsius
- Humedad: 5 – 100%
- Impermeabilidad: IP 67

Figura 2.9 Sensor XKC-Y25-V

Para este apartado esta sería la mejor opción, principalmente porque se trata de un sensor digital y no harían falta más complementos. Además, por condiciones es un sensor más fiable que el Sensor Module 1PC.

3.2.2.3 Nivel de comida

El sensor de nivel de comida tiene una funcionalidad similar al sensor de nivel de agua explicado en el apartado anterior, pero detectando sólidos, en concreto el producto alimentario de los reptiles. Su función será el de detectar si hay comida en el depósito de comida y en el comedero del reptil.

- **Sensor de proximidad capacitivo LJC30A3-H-Z/BY**

Se trata de un sensor de la marca DIANQI, cuyo funcionamiento se basa en la detección de interferencias de un campo electroestático, es decir el sensor crea un campo electroestático y si algún elemento solido se sitúa en dicho campo se causa una interferencia que hace variar el estado del sensor de bajo a alto, para regular el rango de detección incorpora un potenciómetro. Se trata además de un sensor digital, lo que facilita su uso con Raspberry Pi.

Especificaciones técnicas: [13]

- Corriente: 300mA
- Alimentación: 6 – 36V
- Temperatura de funcionamiento: -30 a 65ºC
- Rango detección: 1mm – 10mm

Figura 2.10 Sensor LJC30A3-H-Z/BY

3.2.2.4 Sensor de luz UVB

Los reptiles necesitan radiación UVB y para detectar dicha radiación no basta con un sensor detector de luz, ya que la luz de bombillas normales y de ledes provocan un cambio de estado en estos sensores y evidentemente las luces que disipan no proporcionan radiación UVB, como tampoco lo hace la luz solar cuando se percibe a través de vidrio o plástico. Por lo que se necesita un sensor especializado en detección de rayos UVA y UVB.

- **Sensor VEML6075**

Se trata de un sensor de luz ultravioleta capaz de detectar y medir radiaciones de luz UVB y UVA y convertirlas a datos digitales. Para medir la intensidad de estas radiaciones incorpora fotodiodos. También consta de 4 pines para su conexión con la placa, esto es porque utiliza dos pines para la transferencia de datos, por el hecho de emplear el protocolo de comunicación I2C, que necesita un pin para datos y otro para la señal de reloj. Este es el sensor que se ha elegido para este apartado, ya que trabaja con señales digitales y proporciona datos fiables.

Especificaciones técnicas: [14]

- Corriente: 480µA
- Tensión de alimentación: 1’7 a 3’6 V
- Temperatura de funcionamiento: -40 °C a +85 °C
- Comunicación salida: I2C
- Resolución por canal: 16 bits
- Señal de salida: Digital
- Máxima sensibilidad rayos UVA: 365nm
- Máxima sensibilidad rayos UVB: 330nm
- Ancho de banda espectral: ± 10nm
- Humedad máxima recomendada: 60%

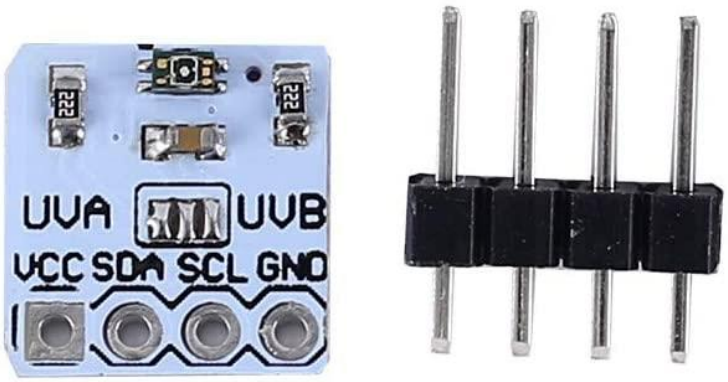


Figura 2.11 Sensor VEML 6075

3.2.3 Actuadores

Los actuadores u *outputs* son dispositivos que a través de energía eléctrica activan el sistema de control de un proceso. Este proyecto requiere el uso de diferentes tipos de actuadores, tantos como sea necesario para controlar las magnitudes que se miden con los sensores. Por lo tanto, se necesita

actuadores para la temperatura, para la humedad, para la radiación UVB y para el suministro de comida y agua. Por desgracia, no se han podido conseguir todos los actuadores necesarios para este sistema, pese a ello se van a mostrar diferentes opciones que servirían para este proyecto.

3.2.3.1 Luz UVB

Para la correcta dosis de rayos UVB se podría optar por colocar el terrario en algún lugar donde el sol incida durante una gran cantidad de horas, pero nos encontraríamos con el problema de tener que dejar el terrario abierto, ya que si los rayos de sol inciden a través de la superficie de plástico o vidrio de un terrario los rayos UVB no traspasan dicho material. Además, existen países cuyas horas de sol no son abundantes o directamente pasan meses sin ver la luz del sol, por lo que está solución no es acertada. Para la correcta resolución de este factor necesario para los reptiles, se ha buscado una bombilla especifica que irradie rayos UVB.

- **Bombilla UVB-UVA NomoyPet**

Se trata de una bombilla que irradia rayos UVA y UVB, además, también desprende calor, así que sirve de apoyo para el dispositivo de calefacción. La marca de la bombilla que hemos buscado es NomoyPet [15] pero existen más marcas que comercializan con este tipo de bombillas.

Especificaciones técnicas: [16]

- Distancia luminosa de hasta 500 metros
- Resistente al agua
- Radiación UVA y UVB
- Diferentes potencias: 25 W/40 W/50 W/60 W/75 W/100 W

Figura 2.12 Bombilla NomoyPet

3.2.3.2 Humedad

La humedad relativa [17] describe la cantidad de agua que se transporta por el aire, es una medida porcentual que refleja la saturación de un volumen especifico de aire a una temperatura específica. Por lo tanto, depende de la temperatura y la presión de un determinado volumen de aire, oscilando entre aire completamente seco, o 0%, y aire saturado, o 100%. Para conseguir una humedad alta se podrían establecer pequeños depósitos de agua que al aumentar la temperatura generaran vapor de agua, pero de esta manera el control del aumento de la humedad estaría completamente ligado a la temperatura, por lo que parece mejor opción disponer de un humidificador. Por otro lado, para la tarea de disminución de la humedad relativa bastaría con un ventilador, ya que de esta manera se consigue disminuir la saturación del aire y es una solución cómoda ya que también se necesita para la gestión de la temperatura. Véase ventilador del apartado 3.2.3.3.

- **Mini humidificador USB DIY**

El mini humidificador USB DIY [18] se trata de un dispositivo micro poroso con 740 orificios de pulverización de 5µm cada uno, capaz de humidificar pequeños espacios. No es un humidificador muy potente, pero junto al entorno húmedo que disponen los terrarios sirve para acelerar el incremento de humedad cuando sea necesario. En el caso de que se necesitara abastecer la humedad de un terrario de grandes dimensiones con este humidificador no bastaría.

Especificaciones técnicas:

- Tensión alimentación: 5V
- Corriente: 300mA
- Potencia: 2W
- Frecuencia: 108kHz
- Diámetro: 16mm

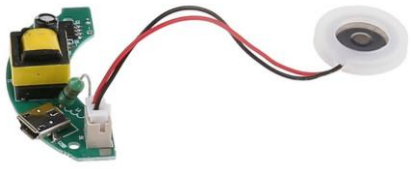


Figura 2.13 Mini humidificador

3.2.3.3 Temperatura

Para el control de esta variable se necesitan dos actuadores, un dispositivo calefactor para aumentar la temperatura siempre que fuera necesario, y un dispositivo refrigerador para disminuir la temperatura cuando esta sea demasiado alta.

- **Almohadilla calefactora eléctrica**

Empezaremos por el dispositivo calefactor, el cual estará apoyado en su labor por la bombilla UVB expuesta en el apartado 3.2.3.1. Hemos pensado que una buena solución para cubrir este apartado son las almohadillas calefactoras, ya que calentará directamente la superficie del interior del terrario y al reptil le será fácil encontrar zonas de calor.

Especificaciones técnicas: [19]

- Composición: Filamentos de poliéster y microfibras de acero inoxidable
- Tensión alimentación: Entre 5 y 12V
- Completamente flexibles

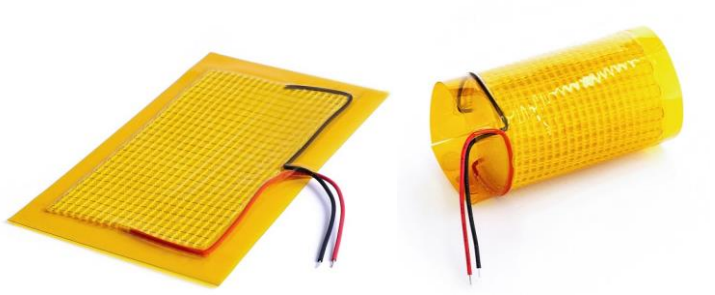


Figura 2.14 Almohadilla calefactora eléctrica

- **Ventilador BLS12/96**

Un ventilador es una buena solución para este apartado ya que no solo sirve para bajar la temperatura del terrario cuando este exceda la temperatura fijada, sino que también sirve como dispositivo para regular la humedad cuando esta sea más alta de lo indicado.

Especificaciones técnicas: [20]

- Dimensiones (A x A x B): 92 x 92 x 25mm
- rodamiento: cojinete liso
- Tensión alimentación: 12V
- Caudal de aire: 74m³/h

- Corriente: 0.16A
- Velocidad: 2400rpm
- peso: 73g
- temperatura de funcionamiento: de -20°C a +40°C

Figura 2.15 Ventilador BLS12/96

3.2.3.4 Hidratación

Para el correcto suministro de agua al reptil, será necesaria una válvula que permita o impida el flujo de agua del tanque al terrario.

- **Válvula de solenoide Hitiland**

Hay que emplear para dicha tarea un tipo de válvula específica para líquidos, llamadas válvulas magnéticas o solenoides [21], que incorporan una bobina que produce un campo magnético que controla el mecanismo de apertura o cierra de agua. Las hay de dos tipos, normalmente cerradas y normalmente abiertas, trabajando de forma que las normalmente abiertas, se mantienen así hasta que les llega la corriente y se cierran, y de forma inversa para las normalmente cerradas.

Especificaciones técnicas: [22]

- Tensión alimentación: 220V
- Modo de funcionamiento: Normalmente cerrado
- Bobina solenoide
- Diámetro orificio: 1/2 "
- Cuerpo de válvula: latón
- Temperatura de funcionamiento: 0 - 40 ℃
- Humedad de funcionamiento: <95% (a 25 ℃)
- Fluido: agua

Figura 2.16 Válvula agua

3.2.3.5 Alimentación

Del mismo modo que en el caso del agua, para la comida también se necesitan válvulas diseñadas para este fin.

- **Válvula de bola de latón motorizada eléctrica DN20**

Este tipo de válvulas disponen de una bola con un agujero, el cual, debido al funcionamiento de un motor eléctrico conectado al sistema, se encontrará abierto o cerrado, permitiendo o no el paso del alimento.

Especificaciones técnicas: [23]

- Tensión alimentación: 220V
- Material: latón
- Diámetro orificio: 3/4 "
- Clase de protección: IP65

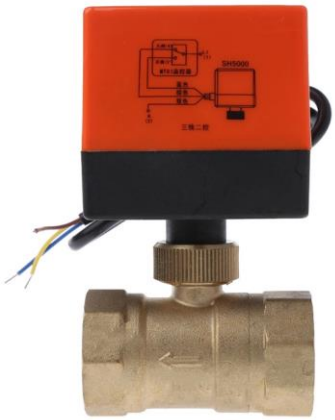


Figura 2.17 Válvula de comida

3.2.4 Otros dispositivos

- **Placa de relés**

Como se puede observar, varios de los dispositivos mostrados trabajan a voltajes superiores a 5 voltios, por lo tanto, la Raspberry Pi por si misma sería incapaz de alimentarlos. Por ello, sería necesario el uso de una placa de relés [24], la cual necesita 5 voltios para su alimentación y puede trabajar con voltajes de hasta 250 voltios, por lo que se podría conectar y controlar desde la Raspberry Pi y proporcionaría el voltaje suficiente para la alimentación del resto de dispositivos.

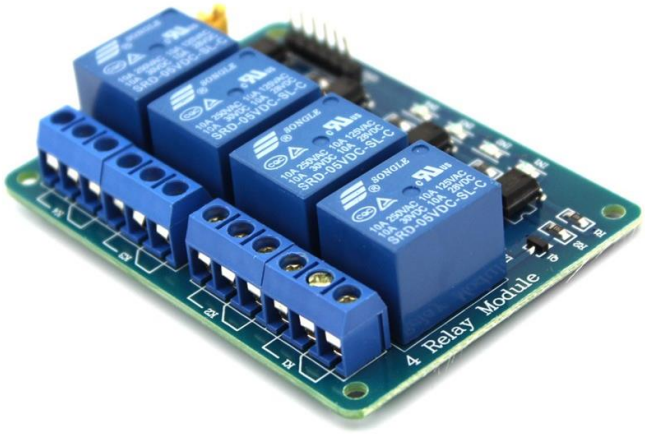


Figura 2.18 Placa de Relés

- **Placa de pruebas (*Protoboard*)**

Para la realización del conexionado de los dispositivos será de gran ayuda el uso de una placa de pruebas, que facilita la organización y conexionado entres sensores, actuadores y la placa base.

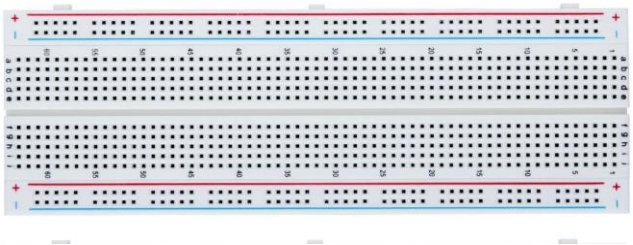


Figura 2.19 Placa *Protoboard*

3.3 Solución propuesta

La solución propuesta en este proyecto consiste en un sistema para automatizar un terrario para reptiles y hacerlo así un lugar idóneo para la vida de cualquier reptil, además también busca facilitar el trabajo de mantenimiento a los propietarios de estos animales.

Con este fin, se han buscado todos los factores regulables para satisfacer las mejores condiciones de vida de los reptiles, en definitiva, diversos factores climáticos y, factores nutricionales y de hidratación. Para ello se han utilizado sensores y actuadores, los cuales interactúan por medio de una lógica programada, la cual también permite a los usuarios ajustar todos los parámetros para las correctas condiciones del reptil. El sistema constará de almohadillas calefactoras, un ventilador, una bombilla UVB, un humidificador, válvulas de comida y agua, sensor de temperatura y humedad, sensor de nivel de agua, sensor de nivel de comida y un sensor de luz solar. El sistema será controlado por la placa base, la cual contiene el código para el correcto funcionamiento de este, y conectara con una aplicación móvil donde el usuario registra las variables que la placa base debe utilizar en su proceso de automatización.

Para el desarrollo del proyecto, una vez identificadas las variables para el correcto cuidado de los reptiles, la primera fase será la creación de la aplicación, donde habrá un apartado para la modificación de cada variable, así como para mostrar la temperatura y humedad en vivo del terrario. A continuación, se realizará la conexión de la aplicación con la Rasberry Pi por medio de Sockets, mediante los cuales puedes enviar información de un lado al otro. El siguiente paso será conocer el conexionado de los sensores y actuadores y generar código para controlarlos, una vez realizado esto, pasaremos a la interacción entre sensores, actuadores y la placa base para que la respuesta de los actuadores dependa de la información recabada por los sensores y por tanto poder automatizar el proceso. Por último, se automatizará el proceso y se montará una simulación hardware del sistema de automatización empleando ledes, y de esta manera se podrá comprobar el correcto funcionamiento del sistema creado.

3.4 Presupuesto

En este apartado se muestra el desembolso monetario necesario para la creación de este sistema, pudiendo evidentemente comprar materiales más caros y más potentes en el caso de que este sistema se quisiera implantar en un terrario de dimensiones grandes, mayores de las de un terrario estándar para hogares. Como se apuntó en el apartado 3.2 no se dispone de todos los dispositivos necesarios, pero para una mejor visión del presupuesto se han añadido todos los dispositivos, distinguiendo entre los adquiridos, color verde, y los no adquiridos, color negro.

Producto		Coste unitario	Cantidad	Total
Tipo	**Nombre**			
Placa base	Raspberry Pi [25]	48’99€	1	48’99€
Sensor temperatura y humedad	DHT11 [26]	1’32€	1	1’32€
Sensor luz solar	VEML 6075 [27]	16’99€	1	16’99€
Sensor nivel agua	XKC-Y25-V [28]	6’02€	2	12’04€
Sensor nivel comida	LJC30A3-H-Z/BY [13]	16’89 €	2	33’78€
Ventilador	BLS12/96 [29]	3’99€	1	3’99€
Calefactor	Almohadilla calefactora eléctrica [19]	6’90€	1	6’90€
Válvula agua	Válvula de solenoide Hitiland [22]	13’09€	1	13’09€
Válvula comida	Válvula de bola de latón [23]	14’38€	1	14’38€
Rayos UVB	Bombilla UVB NomoyPet [16]	5’23€	1	5’23€
Humidificador	humidificador USB DIY [18]	4€	1	4€
Placa de pruebas	ELEGOO Placa *protoboard*	10’99€	1	10’99€
Placa de relés	Placa de Relés [24]	10’83€	1	10’83€
TOTAL general:			**182’53 €**	
TOTAL REAL:			**82’29 €**	

4. Diseño de la solución

4.1 Arquitectura del Sistema

En este apartado hablaremos de la arquitectura software que vamos a utilizar para la creación de nuestro proyecto. La arquitectura software indica la estructura, funcionamiento e interacción entre las diferentes partes del software. Una arquitectura correcta es fundamental para la organización, desarrollo y posterior mantenimiento de una aplicación. El patrón arquitectónico que se va a emplear para la visión global del proyecto es el de Modelo-Vista-Controlador (MVC) [30], el cual tiene una validez contrastada al ser utilizado en todo tipo de aplicaciones, sobre multitud de lenguajes y plataformas de desarrollo.

Modelo-Vista-Controlador es un estilo arquitectónico que separa los datos de la aplicación (Modelo), la interfaz de usuario (Vista) y la lógica de control (Controlador). Vamos a explicarlos con más detalle:

- **Modelo**: Se encarga de la gestión de la información del sistema, es decir cualquier tipo de interacción con los datos pertenecientes al sistema, ya sea consulta o actualización, pasan por la capa de modelo. Esta capa está conectada tanto a la Vista como al Controlador, del Controlador recibe solicitudes para actuar sobre los datos, y a la Vista le envía información, si se le solicita desde el Controlador, para que esta la muestre.

- **Controlador**: Se encarga de la lógica, es la capa que actúa de intermediaria entre el Modelo y la Vista, recibiendo las peticiones de la Vista y realizando solicitudes al Modelo para gestionar la comunicación entre ambas.

- **Vista**: Es la representación visual de los datos, es decir se trata de la interfaz gráfica con la que el usuario ha de interactuar. Su función principal, es servir de puente entre el usuario y el Controlador, ya que el usuario realiza acciones en la Vista, esta realiza peticiones al controlador para completar estas acciones, el Controlador solicita información al Modelo si es preciso, y la Vista muestra al usuario las consecuencias de las acciones realizadas, ya sea mostrando datos o no.

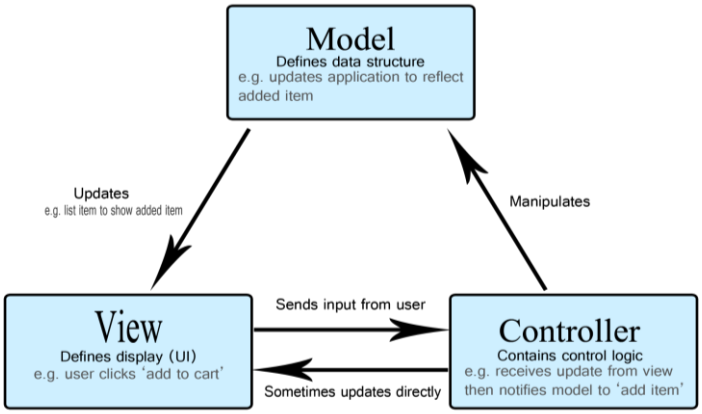


Figura 3.1 Diagrama MVC

En este proyecto, como se ha dividido el código en dos partes, la arquitectura también está dividida. Quedaría distribuida de la siguiente manera, el Modelo y Controlador estarían gestionados desde Eclipse y la parte de Vista estaría en el proyecto de Android Studio, es decir en el código de la aplicación Android, la cual a su vez tiene una arquitectura propia, pero desde la visión global del proyecto la parte Android corresponde a la interfaz gráfica.

4.1.1 Sub-arquitectura para Android

La arquitectura que se va a utilizar en la parte Android viene inspirada en la arquitectura Clean [31], la cual más que una arquitectura se trata de una serie de guías para el desarrollo de software, en la que se definen una serie de capas y se le otorga una responsabilidad a cada una. El objetivo es escribir software de la forma menos acoplada posible a nuestro modelo de datos, a la presentación de este y al entorno de trabajo que estemos usando, con el fin de incrementar la estabilidad de nuestro código y facilitar la portabilidad a otros entornos. Vamos a organizar la aplicación en cuatro capas siguiendo la guía Clean:

- **Modelo:** En esta capa se guardan las clases que presentan la lógica interna de la aplicación y como representamos los datos con los que vamos a trabajar.
- **Datos:** Está formada por las clases que se encargan de almacenar los datos y el acceso a ellos.
- **Casos de Uso:** Se trata de clases que definen las operaciones que el usuario puede realizar con la aplicación.
- **Presentación:** Esta capa representa la interfaz de usuario, está formado por las actividades, *fragments*, vistas y demás elementos con los que el usuario interactúa.

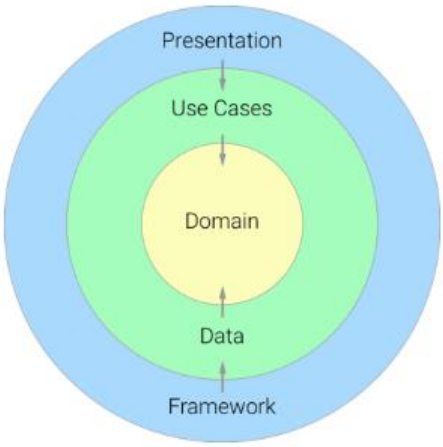


Figura 3.2 Diagrama de dependencia entre capas

Las capas más internas del diagrama, las cuales están más cercanas a nuestra lógica de dominio, no deben depender de las capas más externas del software, aquellas que están más cerca de los agentes externos como el entorno de desarrollo, o el interfaz de usuario.

4.1.2 Comunicación entre sistemas

La comunicación entre la aplicación y el sistema de automatización, es decir la comunicación entre las tres capas de la arquitectura global se ha realizado mediante sockets. Los sockets son un tipo de software que proporcionan un método para la comunicación fiable y ordenada entre un servidor y un cliente a través de una red bidireccional, se puede definir por tanto como el punto final en una conexión. Están constituidos por direcciones IP local y remota, un protocolo de transporte y puertos local y remoto. Diferenciando por el protocolo de comunicación tenemos dos tipos de sockets, TCP y UDP.

El protocolo **UDP** (*User Datagrama Protocol*) no está orientado a la conexión, es decir la transferencia de datos se realiza sin una previa conexión entre los procesos y de forma asíncrona, por lo tanto, es unidireccional, y la máquina de destino cuando recibe los datos, no envía una confirmación a la máquina emisora. Esto se debe a que la encapsulación de datos de este protocolo no contiene más información del emisor que su IP.

El protocolo **TCP** (*Transmission Control Protocol*) sí está orientado a la conexión, es decir que a diferencia del UDP, los procesos hacen una conexión previa por lo cual, cuando el cliente envía información al servidor, este es informado de la llegada de datos, y se comunica para confirmar su recepción.

En este proyecto se emplearán sockets TCP por la necesidad de una conexión síncrona, bidireccional y orientada a la conexión.

4.2 Diseño detallado

En este apartado vamos a detallar el diseño de la parte software, entrando también en la parte de la lógica del sistema de automatización.

4.2.1 Diseño Software

Como se ha explicado en el apartado anterior, por un lado, se harán las capas de Modelo y Controlador y por otra, la capa de Vista. Para ello se utilizan dos plataformas de programación que se explicaran en el primer apartado del punto 4.3 referente a la tecnología utilizada. La organización quedaría de la siguiente manera:

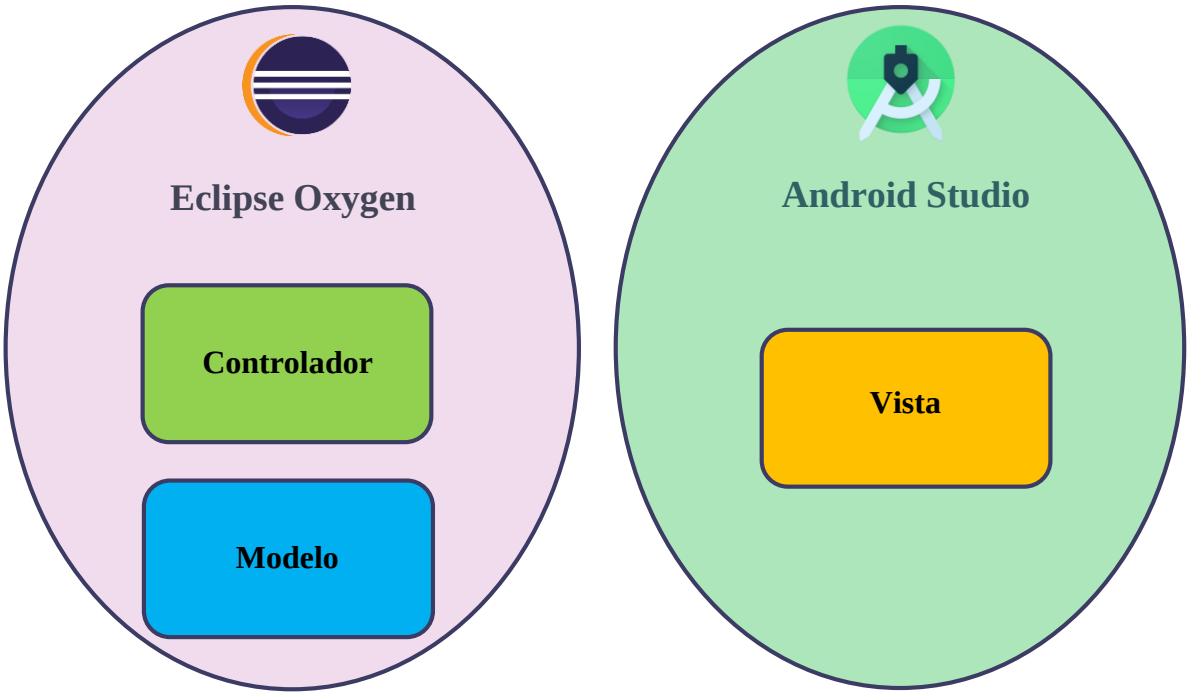


Figura 3.3 Vista general del proyecto

En este punto vamos a mostrar el diagrama con las clases que permiten la interacción entre las tres capas, incluyendo las clases que implementan los sockets en ambos lados.

En primer lugar, para la parte del sistema de automatización, hay una clase que reúne la lógica para el control de sensores y actuadores, la cual necesita instancias del resto de clases para dicha lógica, llamada **HandlerTerrarium**, a su vez esta clase estará instanciada en la clase **Main**, que ejecutará la lógica de **HandleTerrarium** y en la cual esta implementado el socket de esta parte, el socket servidor.

Por la parte de la aplicación, como ya hemos explicado en el apartado anterior (Apartado 4.1.1), la arquitectura es otra, pero el resumen es similar, todo el peso recae en su clase *main*, que en este caso se encarga de iniciar la aplicación y ejecutar el socket cliente. El diagrama general quedaría de la siguiente manera:

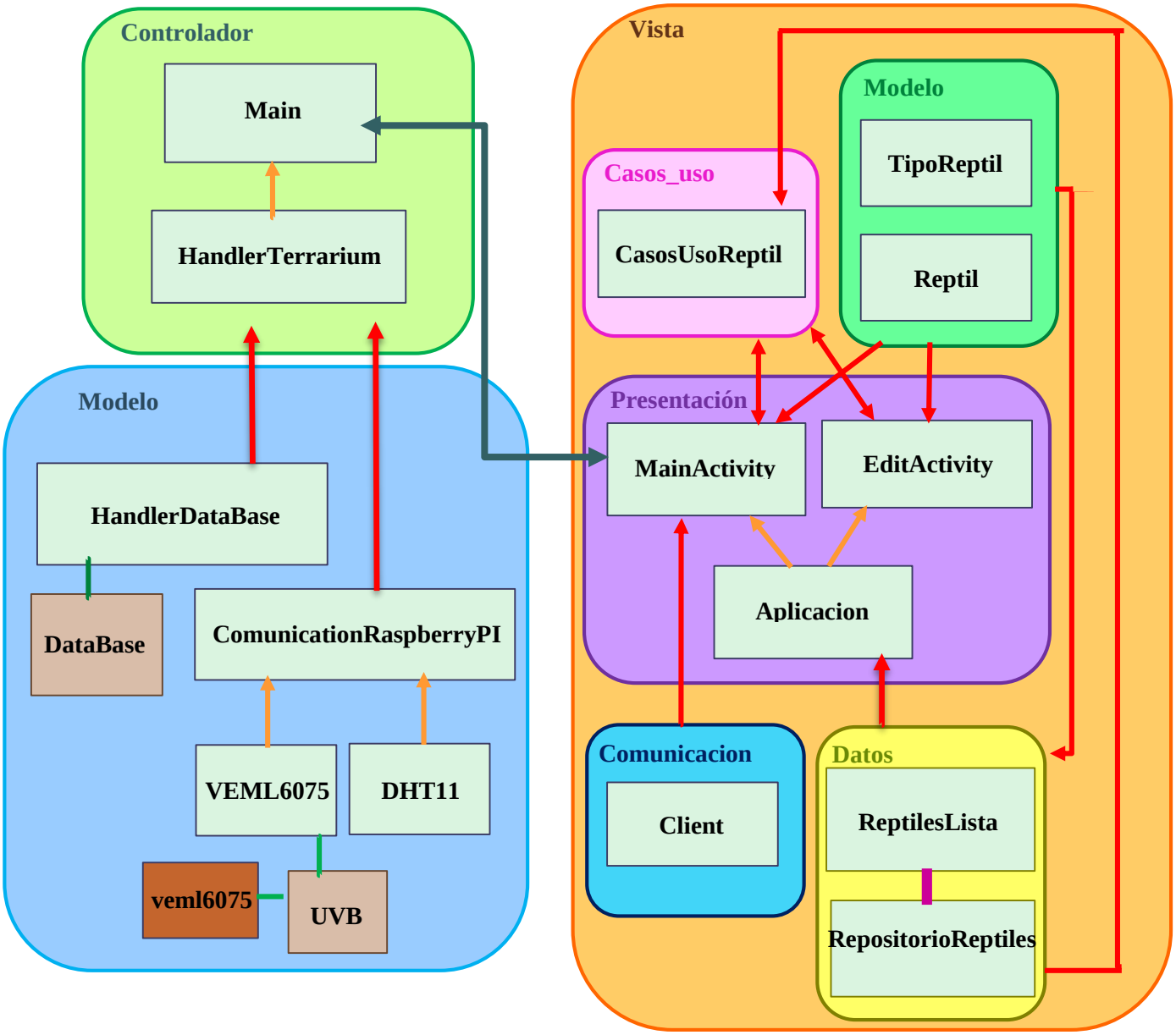


Figura 3.4 Diagrama general

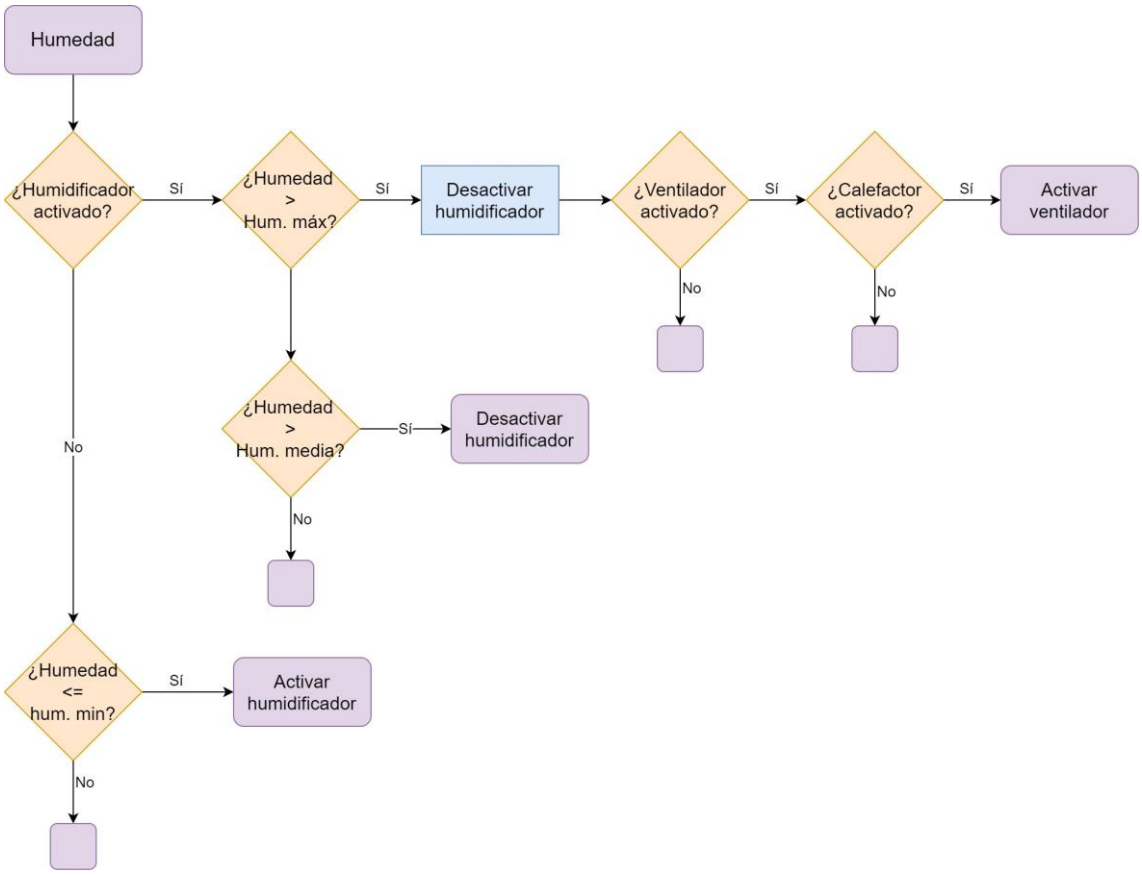
Las flechas rojas representan los *imports* entre clases, las naranjas representan cuando una clase utiliza una instancia de otra de la misma capa, la morada representa cuando una clase hereda de una interfaz, la verde el acceso a una base de datos, y la azul oscura la comunicación por medio de sockets. Los cuadrados blancos representan clases Java, el naranja una clase Python y los rosas ficheros de texto.

4.2.2 Diagrama de flujo

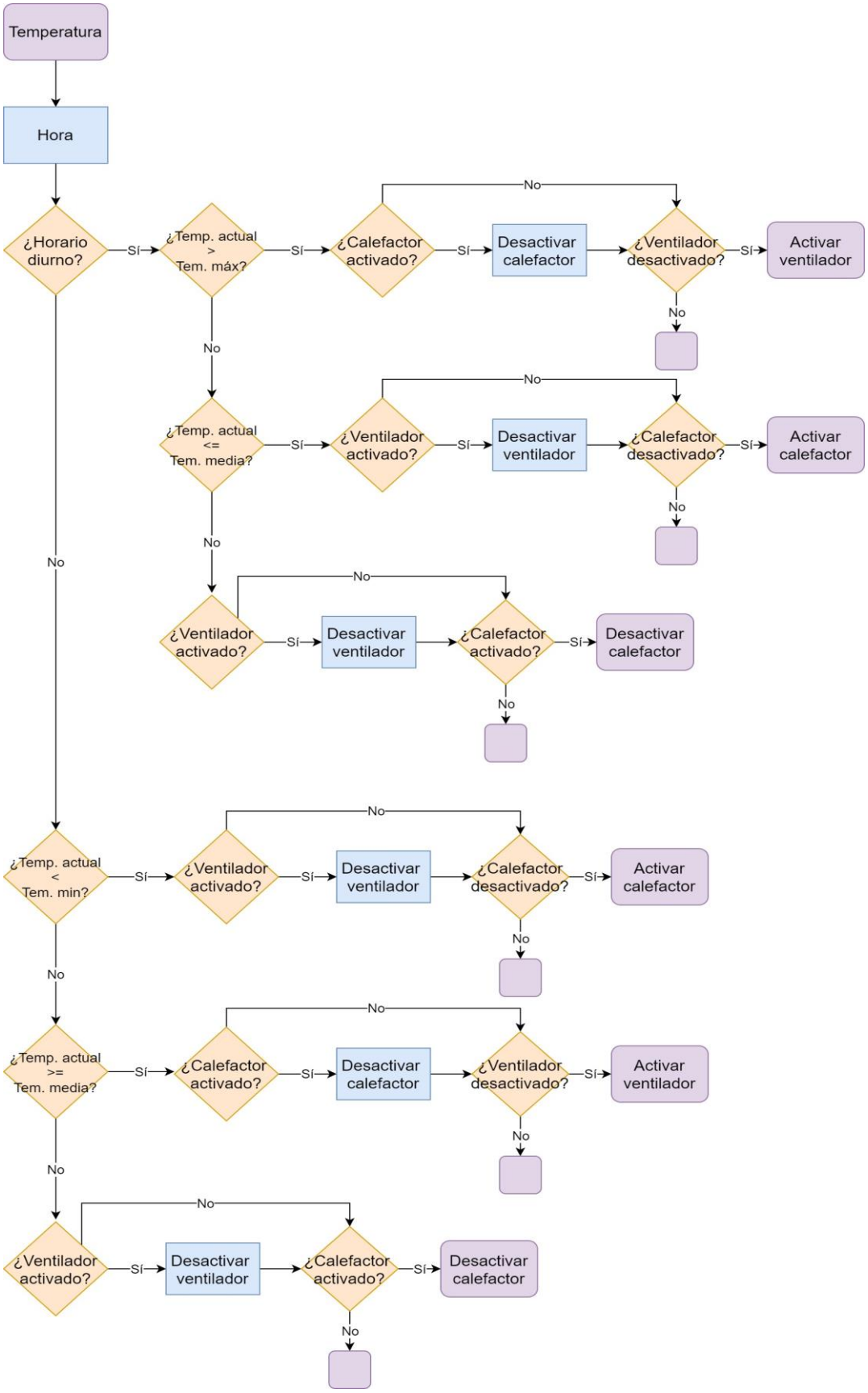
En este apartado se va a mostrar el diagrama de flujo correspondiente a la lógica del sistema de automatización. Los diagramas de flujo permiten observar de forma gráfica las transiciones o pasos que se realizan en la lógica de un programa. En nuestro caso, vamos a plasmar los pasos a realizar por la lógica de nuestros métodos de automatización de la temperatura, humedad, radiación UVB, alimentación e hidratación.

La simbología para los diagramas será la siguiente [32]: Los óvalos hacen referencia al punto inicial o final del sistema, las flechas al orden de ejecución de las operaciones, los rectángulos indican cualquier tipo de operación y los rombos se usan para decisiones de sí o no.

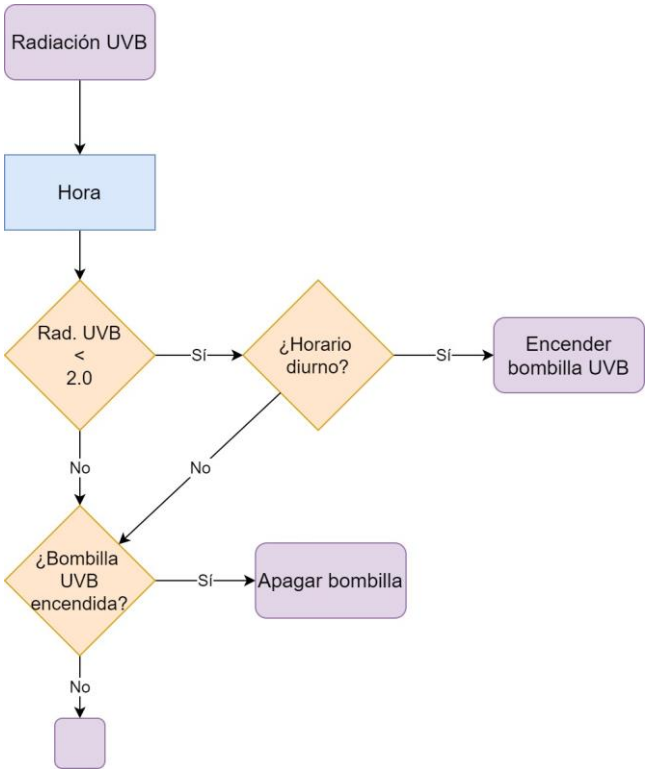
4.2.2.1 Diagrama de flujo humedad



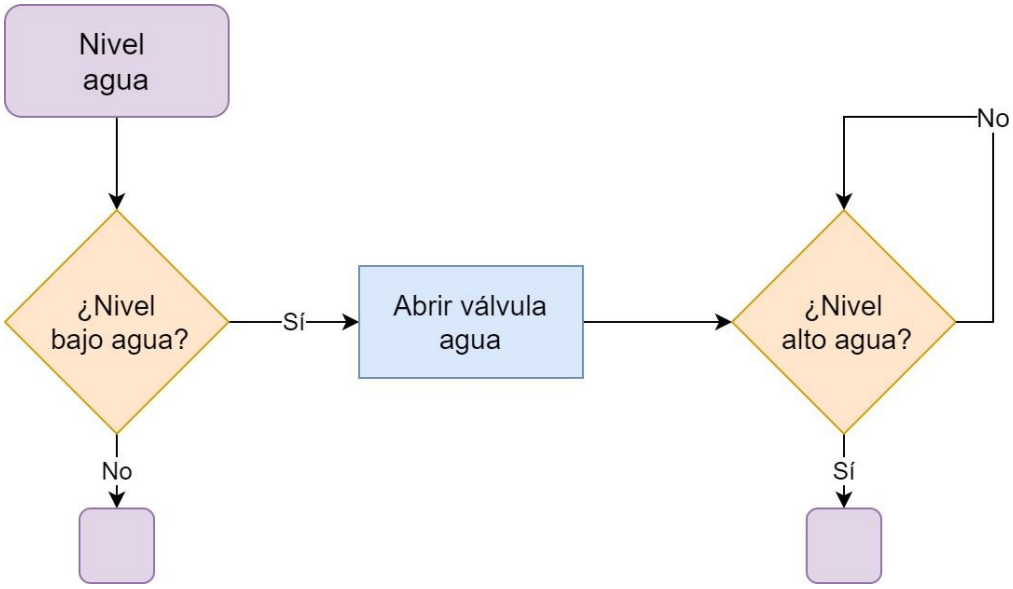
4.2.2.2 Diagrama de flujo temperatura



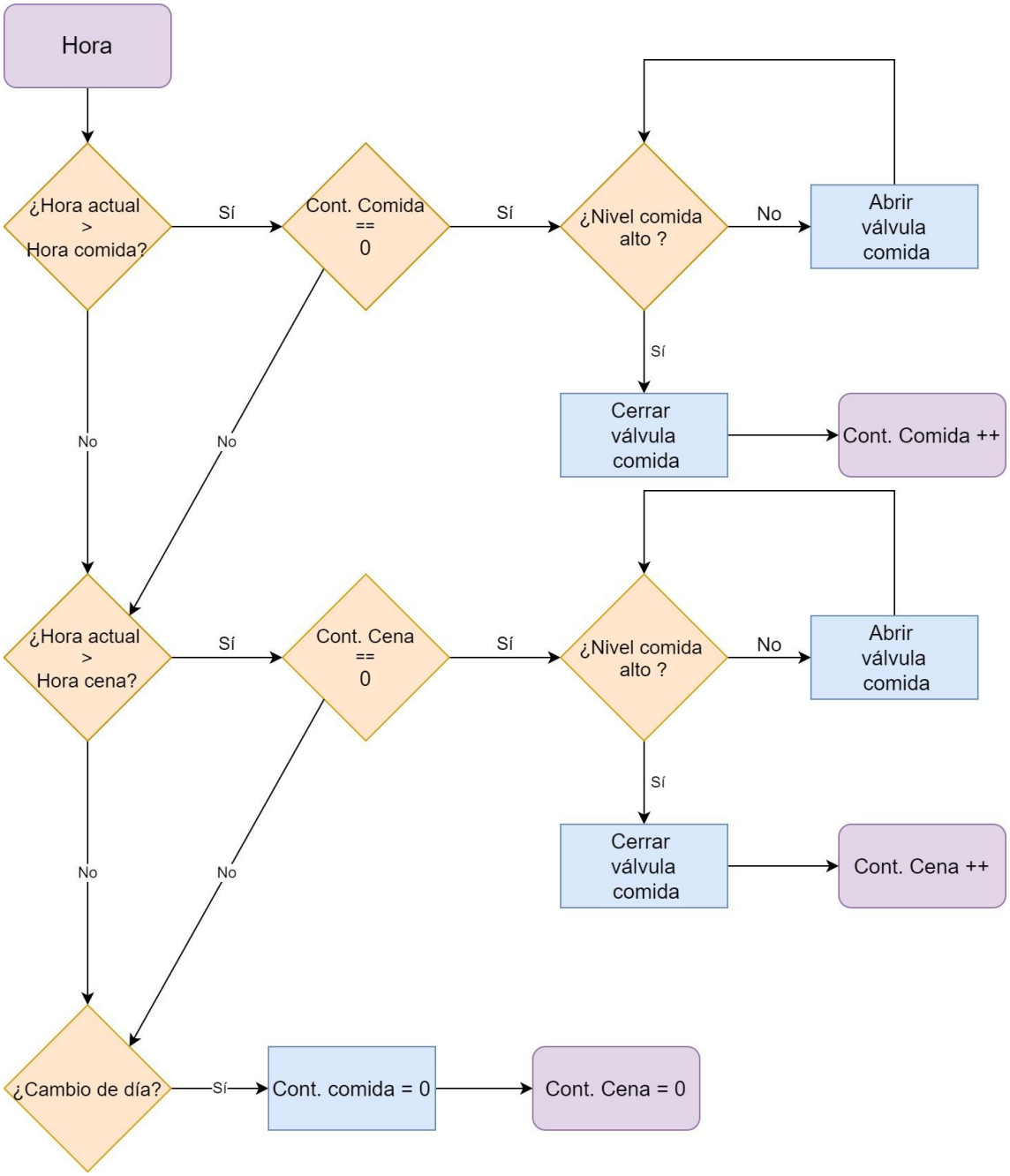
4.2.2.3 Diagrama de flujo radiación UVB



4.2.2.4 Diagrama de flujo hidratación



4.2.2.5 Diagrama de flujo alimentación



4.3 Tecnología utilizada

En esta apartado se van a mostrar las herramientas software y tecnologías utilizadas para la realización del proyecto.

4.3.1 Entornos de programación

En este proyecto se han utilizado dos entornos de programación. Por un lado, tenemos el entorno utilizado para la creación de la aplicación móvil y por otro el entorno donde se ha programado la parte del código que automatiza el sistema, el cual se guarda en la Raspberry Pi para que esta lo ejecute.

4.3.1.1 Android Studio

Para el desarrollo de la aplicación se decidió crear una aplicación Android por lo que se escogió como entorno de programación Android Studio [33]. Se trata del entorno de desarrollo integrado (IDE) para la plataforma Android, por lo que se consideró como la mejor opción. Este entorno proporciona todas las herramientas necesarias para la creación de una aplicación completamente operativa, además es fácil encontrar documentación sobre dudas o información para realizar el código, ya que es una plataforma muy importante y gestionada por Google.

Figura 3.5 Logo Android Studio

4.3.1.2 Eclipse Oxygen

En cuanto al código que se usará para la gestión del sistema automatizado se realiza en Eclipse Oxygen [34], el cual es un entorno de desarrollo integrado que permite construir aplicaciones y administrar códigos. En nuestro caso lo usamos para compilar y organizar código, y además para comprimirlo todo en un archivo *.tar* y poder ejecutarlo desde la Raspberry Pi.

Figura 3.6 Logo Eclipse Oxygen

4.3.2 Sistema operativo

En la Raspberry Pi se ha instalado el sistema operativo Raspbian [35], el cual es una distribución de GNU/Linux basado en Debian, de ahí su nombre. Es el más recomendado para Raspberry Pi ya que está optimizado para su hardware, se trata además de un sistema operativo cuyo desarrollo se mantiene en activo. Para nuestro proyecto hemos instalado la versión con entorno gráfico por comodidad a la hora de interactuar con él, pero podríamos haber instalado la versión sin entorno ya que para el correcto funcionamiento del proyecto no es necesario conectar la Raspberry Pi a una pantalla.

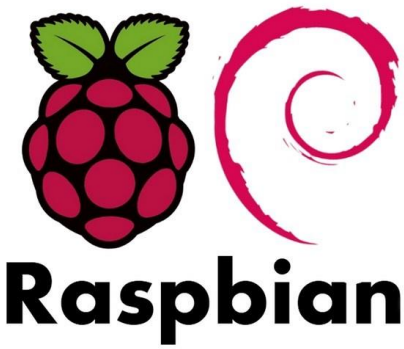


Figura 3.7 Logotipo Raspberry Pi y Debian

4.3.3 Lenguajes de programación

En este proyecto se han empleado fundamentalmente dos lenguajes de programación, uno se ha utilizado para la realización de la aplicación móvil y otro para el funcionamiento del sistema de automatización, o, dicho de otra forma, con uno se ha trabajado en Android Studio y con el otro en Eclipse Oxygen.

4.3.3.1 Java

Java es un lenguaje de programación orientado a objetos, simple, distribuido, portable y uno de los lenguajes más populares, especialmente en aplicaciones cliente-servidor. Se ha escogido por el mayor conocimiento de este lenguaje respecto a otros, por la presencia de la máquina virtual Java en la mayoría de sistemas operativos (incluido Raspbian) y por la compatibilidad con los entornos de programación mencionados en el apartado 4.3.1, ambos con lenguaje predeterminado Java. El proyecto se podría haber hecho en su totalidad con este lenguaje, pero solo se ha empleado para la parte de automatización, es decir la parte de Eclipse.

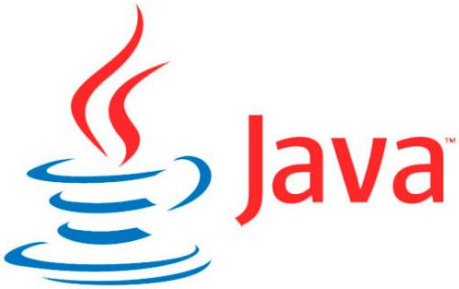


Figura 3.8 Logo Java

4.3.3.2 Kotlin

Kotlin [31] es el lenguaje que vamos a usar para la realización de la parte de Android. Java y Kotlin son los lenguajes oficiales de Android, lo que significa que Android Studio viene con un 100% de soporte para Kotlin. Anteriormente solo estaba Java, pero a partir de 2010 Google decidió que Kotlin también sería lenguaje oficial, esto es porque es un lenguaje más conciso y expresivo que Java e igual de potente. Kotlin surgió de la necesidad de los programadores de migrar a otro lenguaje más simple que Java, pero aparte de simple también necesitaban que no fuera muy potente, ya que el exceso de potencia le da mucha libertad al programador, que podría escribir el código de varias maneras, lo cual es un problema cuando se trabaja en equipo o se

realiza *testing*[5]. En definitiva, crearon Kotlin que se ejecuta en la máquina virtual de Java (JVM), que es plenamente interoperable con Java y que añade técnicas de lenguajes más modernos. Desde su salida Kotlin está cogiendo fuerza entre grandes empresas y se puede decir que es la versión mejorada de Java, es por esto que pese a no haber trabajado con este lenguaje se ha elegido como lenguaje para la parte de Android.

Figura 3.9 Logo Kotlin

4.3.4 Herramientas de apoyo

Para poder interactuar con la Raspberry Pi desde otro ordenador y así facilitar y agilizar el trabajo se han utilizado varios programas dotados con protocolos de comunicación. Se trata de programas para el envío de ficheros, para el control de la consola de la Raspberry Pi y para poder observar e interactuar con la interfaz de nuestra Raspberry remotamente. En todos ellos es necesario introducir la IP del dispositivo remoto, la Raspberry Pi, y la contraseña necesaria para acceder a este.

4.3.4.1 WinSCP

Para realizar la transferencia segura de archivos entre el ordenador que se ha utilizado para la elaboración del proyecto y la Raspberry Pi, se ha empleado WinSCP. WinSCP es un programa para Windows consistente en un cliente SFTP[6] gráfico que basa su funcionamiento en los protocolos SSH[7] y FTP[8].

[5] **Testing:** Actividad en el proceso de control de calidad cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada.
[6] **SFTP (SSH File Transfer Protocol):** Protocolo de transferencia de archivos por omisión.
[7] **SSH (Secure Shell):** Protocolo para el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada.
[8] **FTP (File Transfer Protocol):** Protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor.

Figura 3.10 Logo WinSCP

Al ser un programa gráfico es realmente sencillo trabajar con él, ya que basta con arrastrar los archivos de un lado al otro para poder enviar desde el sistema local, el ordenador en este caso, al remoto, la Raspberry Pi.

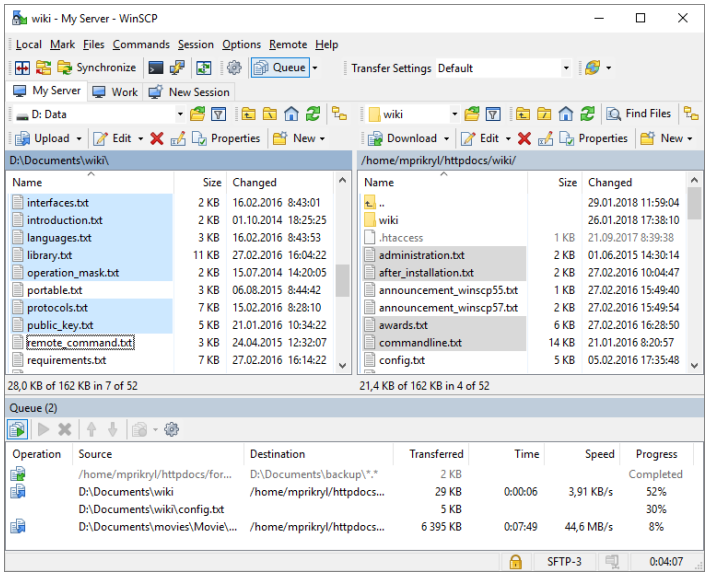


Figura 3.11 Interfaz WinSCP

Para su uso en este proyecto, aparte de su descarga en el ordenador con sistema operativo Windows, es necesario habilitar en la configuración de la Raspberry Pi el protocolo SSH.

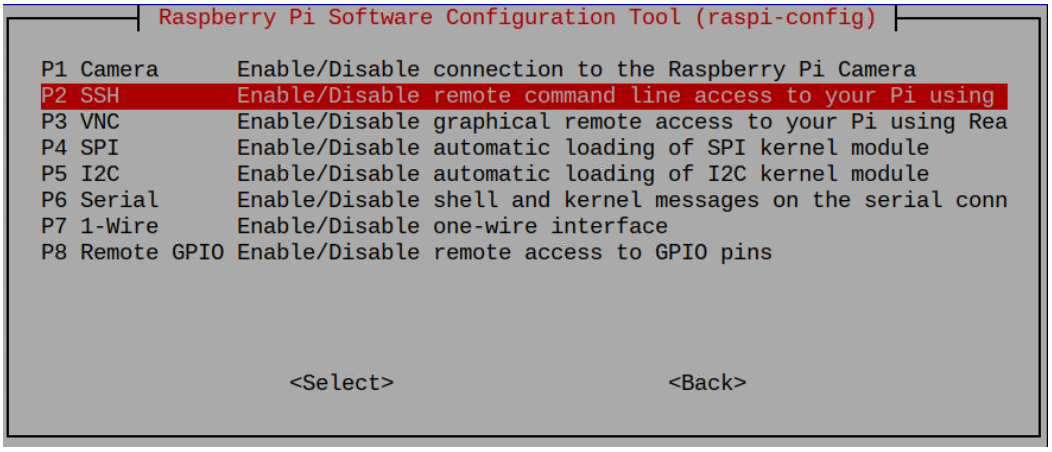


Figura 3.12 Panel habilitación SSH

4.3.4.2 PuTTY

Figura 3.13 Logo PuTTY

Se trata del programa que vamos a utilizar para el acceso y control de la consola de la Raspberry Pi. PuTTY es un cliente SSH y Telnet[9] que permite la conexión a servidores remotos iniciando sesión en ellos.

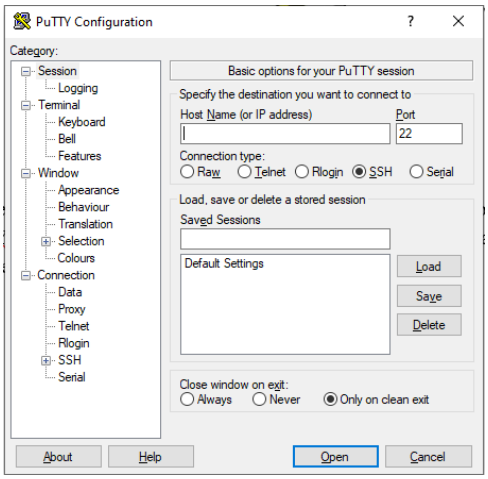


Figura 3.14 Pantalla inicio PuTTY

4.3.4.3 VNC Viewer

Figura 3.15 Logo VNC Viewer

[9] **Telnet (Teletype Network):** Protocolo de red que permite el acceso a otras máquinas para manejarlas remotamente.

Por último, hemos utilizado VNC Viewer para la interacción remota con la interfaz gráfica de nuestra Raspberry Pi. VNC (Virtual Network Computing) es un programa basado en una estructura cliente-servidor que permite al cliente observar la interfaz del servidor de forma remota. Se puede acceder desde cualquier dispositivo, siempre y cuando los dos dispongan de VNC. En este caso también es necesario ir a la configuración de la Raspberry Pi y activarla.

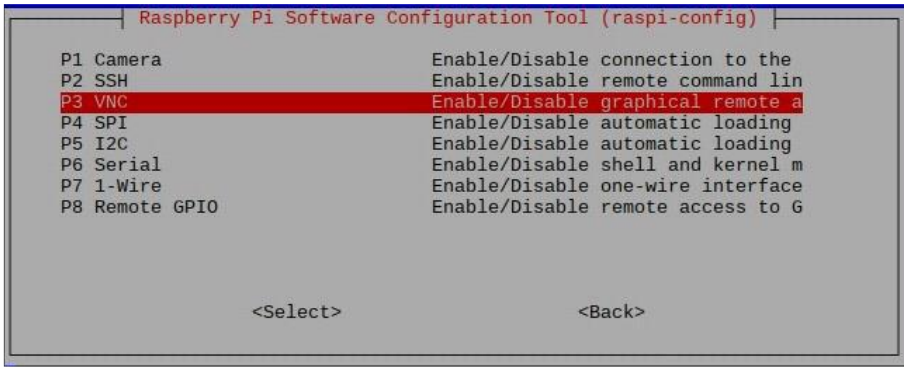


Figura 3.16 Panel configuración VNC

4.3.5 Librerías

Es necesario el uso de ciertas librerías de Java para este proyecto. La más importante es la librería Pi4J, la cual es necesaria para interactuar con los pines de la Raspberry Pi. Pi4J es un proyecto que provee un puente entre librerías nativas y Java para conseguir acceso a los pines de comunicación GPIO, en otras palabras, se trata de una API para que los programadores de Java tengan la capacidad de trabajar con las herramientas de entrada y salida que ofrece la Raspberry Pi.

Figura 3.17 Librería Pi4j

Es necesario instalar esta librería tanto en la Raspberry Pi como en nuestro proyecto de Eclipse para poder compilar y ejecutar el código. La instalación en la Raspberry Pi se resume en el siguiente comando:

```
curl -sSL https://pi4j.com/install | sudo bash
```

Este comando se ha encontrado en la página oficial de Pi4J [36], donde también podemos encontrar más comandos útiles, como para instalar la misma librería pero sin conexión a Internet,

actualizar la librería, ya que es un proyecto activo, etc. De todos estos comandos vamos a destacar los siguientes, necesarios para la compilación y ejecución de los programas:

- Compilar:

```
javac -classpath.: Classes: / opt / pi4j / lib / '*' ...
```

o

```
pi4j --compile programa.java
```

- Ejecutar programas:

```
sudo java -classpath.: Classes: / opt / pi4j / lib / '*' ...
```

o

```
sudo pi4j --run programa
```

- Ejecutar .jar:

```
sudo java -classpath '.:classes:*classes:/opt/pi4j/lib/*' -jar documentoJAR.jar
```

Para la instalación en el proyecto Eclipse, en la página oficial [36] donde encontramos los comandos, también se encuentra los pasos para descargarla y poder utilizarla en plataformas como Eclipse. Primero se descarga en formato .zip y a continuación se importa a el proyecto Eclipse.

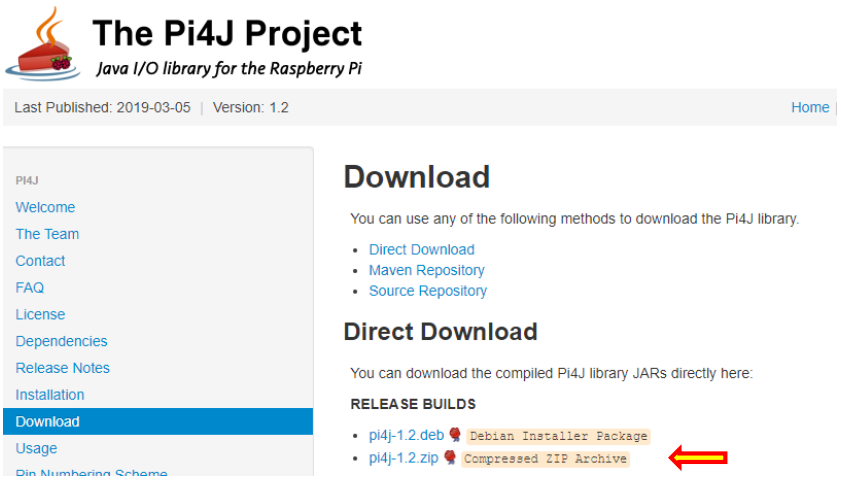


Figura 3.18 ZIP librería Pi4j

5. Desarrollo de la solución propuesta

En este apartado vamos a desarrollar la solución dividiéndola en tres partes. En este orden, primero se explicará el desarrollo de la aplicación móvil, a continuación, el código del sistema de automatización y, por último, el aspecto hardware del sistema.

5.1 Aplicación Android

Vamos a comenzar con la explicación del desarrollo de la aplicación para móviles en Android. La idea principal de la aplicación es la de conseguir una plataforma para gestionar los valores necesarios para el bienestar del reptil, dichos valores son, la temperatura, la humedad, el horario de radiación UVB y el horario de comida. Además, también se debe mostrar la temperatura y humedad del terrario en tiempo real y activar una alerta cuando los niveles de comida y agua en los depósitos sean bajos.

Para la realización de la aplicación hemos empleado la arquitectura Clean, como ya se explicó en el apartado 4.1.1 referente a la arquitectura del sistema Android. Por lo tanto, las clases creadas se dividen principalmente en cuatro grupos o paquetes, modelo, datos, casos de uso y presentación, aunque también se incluye un paquete llamado comunicación donde se encuentra la clase que implementa el socket cliente.

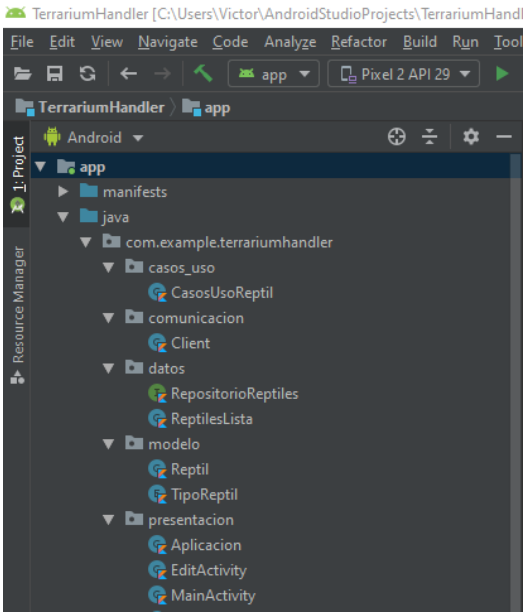


Figura 4.1 Estructura Android

5.1.1 Modelo

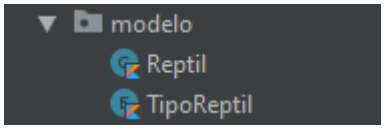


Figura 4.2 Paquete Modelo

Comenzamos por la capa de modelo, es la primera capa que se realiza ya que representa las clases que proporcionan la lógica de los datos de la aplicación, es decir, son las clases que marcan el tipo de datos con los que se va a trabajar.

Clase Reptil

Se trata de la clase POJO (*Plain Old Java Object*) de este proyecto. Las clases POJO [31] son clases muy simples en las cuales se describe la información necesaria para representar un objeto, que itera con otras clases de manera que es fundamental. En este caso, representaremos el objeto principal del proyecto, los reptiles. En la clase **Reptil** se definen las variables que necesitamos conocer para identificar y cuidar de estos animales.

```
package com.example.terrariumhandler.modelo

data class Reptil(var nombre: String = "",
                  var tipoReptil: TipoReptil,
                  var temp_min: Int = 0,
                  var temp_max: Int = 0,
                  var hum_min: Int = 0,
                  var hum_max: Int = 0,
                  var ini_luz: Int? = null,
                  var fin_luz: Int? = null,
                  var hora_comida: Int = 0,
                  var hora_cena: Int = 0
)
```

Clase TipoReptil

Se trata de una clase de tipo enumerado (*enum*) y sirve para diferenciar entre diferentes tipos de reptiles. Las clases *enum* sirven para nombrar y representar un determinado número de constantes de un tipo, y de esta forma crear un nuevo tipo de datos. Para que quede más claro, divide la clase **Reptil** en tipos de reptil, y de esta manera podemos ajustar valores estándar a tantas especies de reptiles como queramos.

```
package com.example.terrariumhandler.modelo

enum class TipoReptil private constructor(val tipo:String, val t_min:Int, val t_max:Int,
val h_min:Int, val h_max:Int) {
    CAMALEON_VELADO("Camaleon Velado", 26, 35, 50, 50),
    CAMALEON_JACKSON("Camaleon Jackson", 21, 29, 50, 80),
    CAMALEON_PANTERA("Camaleon Pantera", 24, 35, 60, 85),
    GECKO_LEOPARDO("Gecko Leopardo", 22, 35, 55, 55),
    TORTUGA_TIERRA("Tortuga de tierra", 27, 30, 50, 50),
```

5.1.2 Datos

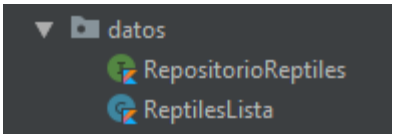


Figura 4.3 Paquete Datos

Esta capa almacena y da acceso a los datos, de tal forma si se quieren modificar o consultar datos se tendrá que trabajar con las clases de este paquete. En esta aplicación esta capa se encarga de los datos relacionados con las condiciones de los reptiles, que se muestran y se modifican desde la aplicación, almacenando también en esta capa los datos que vienen desde el sistema de automatización, y compartiendo los datos que modifica el usuario con dicho sistema.

Clase RepositorioReptiles

Esta clase es una interfaz. Una interfaz es una clase en la que se define una colección de métodos abstractos para cierto tipo de objetos, con los que se especifica que se debe hacer, pero no se incluye implementación. En nuestro trabajo el **RepositorioReptiles** nos permitirá almacenar una lista de objetos **Reptil**. Debido al alcance de esta aplicación, no haría falta implementar esta clase, pero para realizar una aplicación más óptima en futuras implementaciones, serviría para almacenar los datos en una base de datos en lugar de en memoria, y de esta manera desacoplar el método de almacenamiento de datos del resto de la aplicación.

```
interface RepositorioReptiles {
    fun element(id: Int): Reptil  //Devuelve el reptil dado su id
    fun add_eptil(reptil: Reptil)      //Añade el reptil indicado
    fun new_reptil(): Int       //Añade un reptil en blanco y devuelve su nombre
    fun delete(id: Int)    //Elimina el reptil con el id indicado
    fun size(): Int     //Devuelve el número de reptiles
    fun update(id: Int, reptil: Reptil)  //Reemplaza un reptil
```

Clase ReptilesLista

Implementa la interfaz **RepositorioReptiles** con el objetivo de almacenar y gestionar, dentro de una lista, un conjunto de objetos **Reptil**.

```
package com.example.terrariumhandler.datos

import com.example.terrariumhandler.modelo.Reptil
import com.example.terrariumhandler.modelo.TipoReptil

class ReptilesLista : RepositorioReptiles {
    val listaReptiles = mutableListOf<Reptil>()
    init {
        añadeEjemplos()
    }

    override fun element(id: Int): Reptil {
```

```
        return listaReptiles[id]
    }

    override fun add_reptil(reptil: Reptil) {
        listaReptiles.add(reptil)
    }

    override fun new_reptil(): Int {
        val reptil = Reptil(
            "Nuevo Reptil",
            tipoReptil = TipoReptil.OTROS
        )
        listaReptiles.add(reptil)
        return listaReptiles.size - 1
    }

    override fun delete(id: Int) {
        listaReptiles.removeAt(id)
    }

    override fun size(): Int {
        return listaReptiles.size
    }

    override fun update(id: Int, reptil: Reptil) {
        listaReptiles[id] = reptil
    }
```

5.1.3 Comunicación

Figura 4.4 Paquete Comunicación

Clase Client

Existe únicamente esta clase en el paquete comunicación. Se trata de la clase donde se implementa el socket TCP de la parte del cliente, es decir el socket que inicia la comunicación. Dicha inicialización se hará al abrir la aplicación por lo que existirá una instancia de dicha clase en el método m*ain* de la aplicación (**MainActivity**).

El constructor Client necesita cuatro atributos, *address*, para la dirección IP del servidor, *port*, para introducir el puerto donde conectara con el servidor, y *humedadFile* y *temperaturaFile*, se trata de variables tipo *TextView* que se utilizaran en la clase Main para mostrar en la aplicación el valor de temperatura y humedad actuales enviados desde la Raspberry Pi. (Se explicará con más detalle en el apartado 5.1.4 Presentación)

```
class Client(address: String, port: Int, humedadFile: TextView, temperaturaFile :
TextView){
    val cliente: Socket = Socket(address, port)
//Se inicia la conexión socket con la dirección IP y el puerto del servidor.
```

Para la lectura de la información que llega desde el servidor, y el envío de datos a este, son necesarios objetos especiales. Para la lectura utilizaremos un *BufferedReader*, y para el envío se usará un *PrintStream.*

```
//Inicialización variables lectura y escritura de datos.
val reader : BufferedReader by lazy
{BufferedReader(InputStreamReader(cliente.getInputStream()))}

 val salida: PrintStream by lazy {PrintStream(cliente.getOutputStream())}
```

A continuación, se crea un método donde se utilizan estas variables para el envío y recepción.

```
//Lectura
temperatura = reader.readLine()
humedad = reader.readLine()

//Escritura
salida.println(tempMin)
salida.println(humedadMin)
```

Para la lectura, tanto la variable de temperatura, como la de humedad se crean de tal forma que, si no se recibieran valores por algún motivo, como un fallo en los sensores, se le dé a la variable el último valor que tenía.

```
var humedad : String by Delegates.observable("0") { property, oldValue, newValue ->
    if (newValue != "NF")
    humedadFile.text = newValue + "%"
    else oldValue
```

5.1.4 Presentación

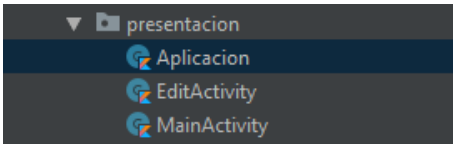


Figura 4.5 Paquete Presentación

En la capa de presentación se desarrolla la lógica de la interfaz gráfica, es decir se conecta con el diseño *.xml* creado para la aplicación, y se le da funcionalidad a todos los elementos que componen dichos *.xml* (Figura 4.6). Además, este mismo paquete, contienen también el código para comunicarse con la capa de Datos. La mayoría de las clases de este paquete son Actividades, que representan lo que llamamos coloquialmente como pantallas de la aplicación. Una aplicación tiene muchas actividades independientes pero que trabajan para un objetivo común.

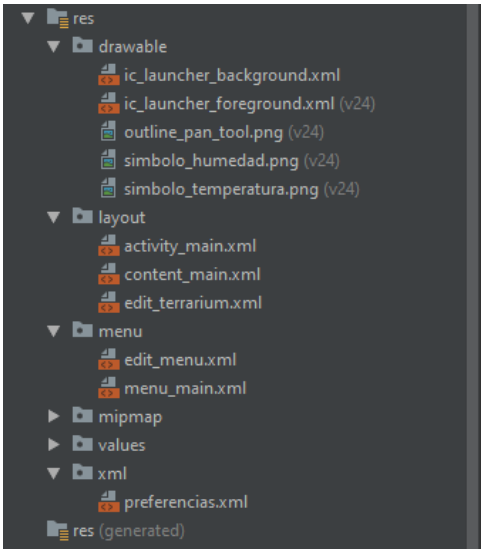


Figura 4.6 Paquetes diseño gráfico

Clase MainActivity

Aparte de la clase con el código de la actividad inicial, se trata de la clase m*ain* de la aplicación. Por lo tanto, es la clase que inicia la ejecución de la aplicación, en la cual se inicializa el socket para recibir la información proveniente del sistema de automatización almacenado en la Raspberry, y asocia la actividad con su *layout* correspondiente, que en este caso es **activity_main** alojado en la carpeta res/layout que se puede observar en la figura 4.7. Una vez hecho esto, otorga funcionalidades a todos los elementos creados en esta, como menús, botones y elementos de texto.

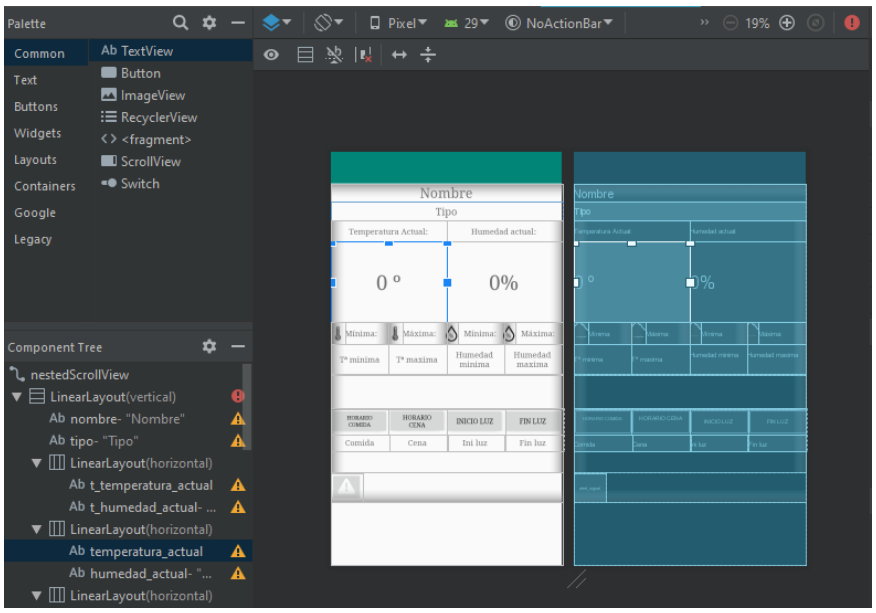


Figura 4.7 Layout activity_main

En la inicialización del socket, a parte de la dirección IP de la Raspberry Pi y el Puerto donde se va a realizar la comunicación, encontramos la variable *humedad_actual* y *temperatura_actual*, las cuales hacen referencia a dos *TextViews* del *layout* **activity_main**, más concretamente los que se indican en la figura 4.8. Por lo tanto, cuando estas variables se modifiquen se mostrará en pantalla.

Figura 4.8 TextView temperature_actual

```
val client : Client by lazy {Client("192.168.0.157", 9000, humedad_actual,
temperatura_actual)} // Inicialización socket
thread{
    try {
        client.run(reptil.temp min.toString(), reptil.temp max.toString(),
reptil.hora_comida.toString(),
            reptil.hora_cena.toString(), reptil.hum_max.toString(),
reptil.hum_min.toString(), reptil.ini_luz.toString(),
                    reptil.fin_luz.toString())
    }
    catch (ex : IOException){
        System.err.println(ex)
    }
```

```
//Código para conectar una actividad con su layout

override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
```

El código relacionado con los cambios de vista se implementa en el paquete de casos de uso. En el siguiente código se muestra como se llama a una instancia de la clase **CasosUsoReptil** y se utiliza al método implementado en esta clase para pasar a la actividad **EditActivity**.

```
val usoReptil by lazy { CasosUsoReptil(this, reptiles) }
```

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.

    return when (item.itemId) {
        R.id.action_settings -> {
            lanzarPreferencias()
            true
        }
        R.id.action_edit -> { //metodo editar de la clase CasosUsoReptil
```

```
            usoReptil.editar(0, RESULTADO_EDITAR) //RESULTADO_EDITAR == 1
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Los dos parámetros que se pasan en el método editar sirven para indicar sobre que reptil registrado en la aplicación se quiere realizar la consulta o modificación, y los resultados de que reptil deben aparecer. Puesto que vamos a trabajar únicamente con un reptil, a ambas variables se les pasa un valor fijo, pero se ha hecho así para futuros avances de la aplicación.

Se ha creado también la posibilidad de cambiar cada condición del terrario de forma individual y así no tener que cambiar de actividad. Lo mostramos con el ejemplo para la edición de la temperatura mínima y la respuesta que genera. Además, para este código hace falta añadir un escuchador (*listener*) al botón.

```
rango_minimo_T.setOnClickListener{ //Escuchador de temperatura mínima
    editarTemperaturaMin()
}
```

```
fun editarTemperaturaMin(view: View? = null){
    val entrada = EditText(this)    //setInputType(InputType.TYPE_CLASS_NUMBER)
    entrada.inputType=InputType.TYPE_CLASS_NUMBER
    entrada.setText(reptil.temp_min.toString())
    AlertDialog.Builder(this)
        .setTitle("Temperatura minima")
        .setMessage("Tª:")
        .setView(entrada)
        .setPositiveButton("Guardar"){ dialog, whichButton ->
            val id = parseInt(entrada.text.toString())
            reptil.temp_min = id
            actualizaVistas()
            nestedScrollView.invalidate()

        }
        .setNegativeButton("Cancelar", null)
        .show()
}
```

Clase EditActivity

Se trata de la actividad a la que se accede para editar los valores de las condiciones asignadas a un reptil. Hay dos opciones para salir de esta actividad y volver a la actividad principal, cancelar, que deja los datos como se encontraban antes de entrar a esta actividad, o guardar, que también vuelve al **MainActivity** pero modifica los valores del terrario acorde con la información registrada en esta actividad. La opción guardar también requiere una llamada a un método de la clase **CasosUsoReptil**.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.getItemId()) {
            R.id.accion_cancelar -> {
                finish()
                return true
            }
            R.id.accion_guardar -> {
                val nuevoReptil = Reptil(nombre.text.toString(),
TipoReptil.values()[reptil.tipoReptil.posNombre(w_tipo.text.toString())],
                    Integer.parseInt(temp_min.text.toString()),
```

```
Integer.parseInt(temp_max.text.toString()),
                    Integer.parseInt(hum_min.text.toString()),
Integer.parseInt(hum_max.text.toString()),
                    Integer.parseInt(ini_luz.text.toString()),
Integer.parseInt(fin_luz.text.toString()),
                    Integer.parseInt(hora_comida.text.toString()),
Integer.parseInt(horario_cena.text.toString())
                    )
                usoReptil.guardar(0, nuevoReptil)
                finish()
                return true
            }
            else -> return super.onOptionsItemSelected(item)
        }
    }
```

Clase Aplicación

La clase **Aplicación** nace de la necesidad de compartir objetos entre todas las clases de la aplicación. Una posibilidad que da Android para acceder a información global desde cualquiera de nuestras clases es la creación de una clase comodín que hereda de la clase *Application*, la cual ha sido creada con el propósito de almacenar información global de toda la aplicación. En este proyecto, la clase Aplicación desciende de *Application* y debe estar provista de la información global y de métodos para la gestión de dicha información, más concretamente almacena el objeto **ReptilesLista** del cual queremos usar una instancia compartida por todas las clases.

```
package com.example.terrariumhandler.presentacion

import android.app.Application
import com.example.terrariumhandler.datos.ReptilesLista

class Aplicacion : Application() {
    val reptiles = ReptilesLista()
}
```

5.1.5 Casos de Uso

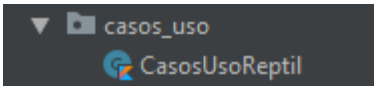


Figura 4.9 Paquete Casos de uso

Clase CasosUsoReptil

En esta clase se implementan los métodos con las diferentes acciones a realizar en la aplicación. Este tipo de clases les quita mucha carga a las actividades. En todas las actividades de la aplicación se crea una instancia a esta clase, que tiene dos atributos, *actividad* de tipo *activity*, y *reptiles* de tipo **RepositorioReptiles**.

```
class CasosUsoReptil(val actividad: Activity,
                     val reptiles: RepositorioReptiles)
```

Solo tenemos tres métodos en esta clase, dos de ellos para el cambio de actividad, o lo que es lo mismo para el cambio de pantalla de la aplicación. En estos dos métodos se utilizan un recurso importante para la programación en Android, las intenciones (*Intent*), las cuales permiten ejecutar una actividad distinta de la que estamos usando, ya sea de nuestra aplicación o de otra. En este caso solo lanzamos actividades de nuestra propia aplicación. Las intenciones tienen dos atributos, el primero para la actividad origen y el segundo para la actividad destino, y permiten intercambiar información entre la actividad origen y destino.

En primer lugar, tenemos el método mostrar que se utilizará para actualizar la pantalla principal cuando se realice alguna modificación desde la pantalla de edición de las condiciones del reptil. Además, este método servirá para futuras implementaciones en las que se pueda manejar más de un sistema desde el mismo dispositivo.

```
fun mostrar(pos: Int) {
    val i = Intent(actividad, MainActivity::class.java)
    i.putExtra("pos", pos)
    actividad.startActivity(i)
}
```

En segundo lugar, tenemos el método editar, que se utiliza para saltar a la actividad **EditActivity** donde se modifican los valores del terrario, cuyo código es igual que el del método *mostrar* salvo por el hecho de que la actividad predecesora, es decir, la pantalla principal (**MainActivity**) se queda esperando la respuesta de esta, por lo que en lugar de *starActivity* usará el método *starActivityForResult.*

Por último, tenemos el método guardar que utiliza el método *actualiza* de la clase **ReptilesLista** para actualizar los valores de las condiciones del terrario que se han modificado en la pantalla de editar.

```
fun guardar(pos: Int, nuevoRpetil: Reptil){
    reptiles.actualiza(pos, nuevoRpetil)
}
```

5.2 Proyecto Eclipse

El objetivo del proyecto en Eclipse Oxygen es el de proporcionar la capa de Modelo y Controlador de la arquitectura software general del sistema (Modelo-Vista-Controlador). Es decir, en este proyecto se ha de crear la lógica y la capa de gestión de datos del sistema de automatización. Esto engloba, la gestión de todas las condiciones climáticas, el suministro de comida y agua, así como el aviso en caso de niveles bajos en los depósitos, el envío y recepción de información de la aplicación Android y el tratamiento de dicha información.

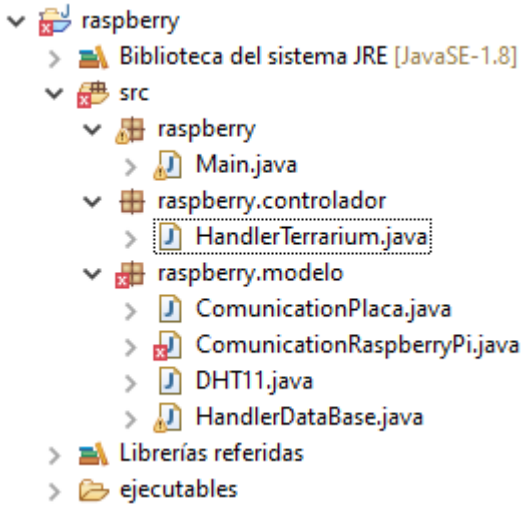


Figura 4.10 Estructura proyecto Eclipse

5.2.1 Modelo

En el paquete Modelo encontraremos las clases relacionas con el almacenamiento, adquisición y manejo de los datos.

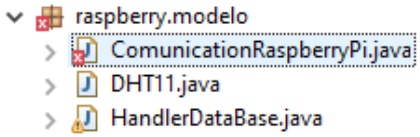


Figura 4.11 Paquete Modelo

Clase ComunicationRaspberryPi

En esta clase se implementan los métodos para el control de los sensores y actuadores, a través de los pines GPIO de la Raspberry Pi, para ello se utiliza la librería Pi4J ya mencionada en el apartado 4.3.5 Librerías.

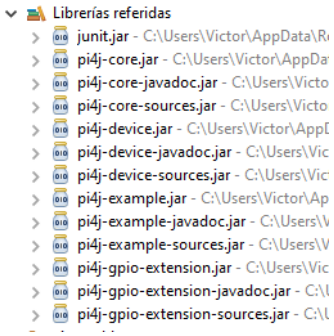


Figura 4.12 Librería Pi4J

Para el control de los pines GPIO la librería Pi4J proporciona una extensa lista de métodos, de los cuales los fundamentales son los que mostramos a continuación, cuyo código encontramos explicado en la página de Pi4J [36] en el apartado *Usage*:

```
//Crear instancia de controlador GPIO

final GpioController gpio = GpioFactory.getInstance();
```

Esta instancia es común a todo el proyecto y es necesaria para la inicialización de todos los pines.

```
//Aprovisionamiento de pin de salida.

ventilator = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03, "Ventilador",
PinState.HIGH);
ventilator.setShutdownOptions(true, PinState.HIGH);
```

```
//Aprovisionamiento de pin de entrada.

temperatureSensor = gpio.provisionDigitalInputPin(RaspiPin.GPIO_15, "Sensor
temperatura");
temperatureSensor.setShutdownOptions(true);
```

Se debe configurar cada pin en función de si se usará para controlar un dispositivo de entrada (*Input*) o un dispositivo de salida (*Output*). El método consta de parámetros para indicar el pin GPIO que se desea controlar, el nombre que se le desea dar y, en el caso de los pines para dispositivos de salida, el estado inicial del pin. Con el segundo método *setShutdownOptions* se indica el estado del pin cuando se cierra el programa, en el caso de los *Inputs* se desactiva el pin y no lee más datos hasta que se vuelva a activar.

```
// Activación pines Outputs
public void actVentilator() {
            ventilator.high();
      }

//Desactivación pines Outputs
public void deactVentilator() {
            ventilator.low();
      }
```

Los métodos *.high()* y *.low()* sirven para modificar el estado de los pines de salida, en otras palabras, activan y desactivan los actuadores, como por ejemplo el ventilador.

```
//Comprobar estado Inputs.

public boolean getHighLight() {
		return lightSensor.isHigh();
	}

public boolean getLowLight(){
		return lightSensor.isLow();
	}
```

Por el contrario, los métodos para comprobar si los sensores están a nivel alto o bajo son *.isHigh()* y *.isLow()* que devuelven *true* o *false* dependiendo del estado del sensor. Si por ejemplo el sensor de radiación UVB está detectando la luz solar el estado de este sensor será alto y el método *.isHigh()* devolverá *true*.

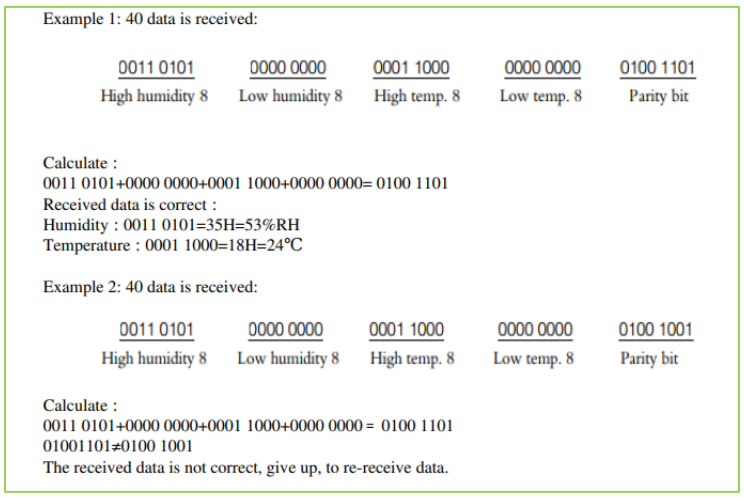
La lista de todos los métodos que se implementan en esta clase es la siguiente:

```
void pinsPower();
	void actHeater();
	void deactHeater();
	void actVentilator();
	void deactVentilator();
	void actHumidificator();
	void deactHumidificator();
	void actFoodValve();
	void deactFoodValve();
	void actWaterValve();
	void deactWaterValve();
	void actLight();
	void deactLight();
	void warning();
	float getTemperatura() throws InterruptedException;
	float getHumedad() throws InterruptedException;
	boolean getWaterLowLevel();
	boolean getWaterHighLevel();
	boolean isHighFood();
	boolean isLowFoodTank();
	boolean getLowLevelWaterTank();
	boolean isHighLight() throws IOException;
	void close();
	boolean isVentilatorHigh();
	boolean isHeaterHigh();
	boolean isHumidificatorHigh();
	boolean isLightHigh();
```

Todos ellos implementados con los métodos que incorpora la librería Pi4J que se han expuesto en este apartado, con la excepción de *getTemperatura()*, *getHumedad()* y *isHighLight()* en los que se necesita una instancia de otra clase que se expone a continuación.

Clase DHT11

En la clase **DHT11** se implementan dos métodos para la obtención de la temperatura y humedad a través del sensor DHT11. Este sensor envía transmisiones de 40 bits, divididos de la siguiente manera: 8 bits de la parte entera de la humedad, 8 bits de la parte decimal de la humedad, 8 bits de la parte entera de la temperatura, 8 bits de la parte decimal de la temperatura y 8 bits de paridad, necesarios para la detección de errores. Los métodos de esta clase trabajan con estos bits para obtener el valor numérico de la temperatura y la humedad que transmite el sensor. En la siguiente figura se muestran dos ejemplos de la lectura de dichos bits, ejemplos que se obtiene del manual del sensor DHT11 [10].


Example 1: 40 data is received:

0011 0101	0000 0000	0001 1000	0000 0000	0100 1101
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate :
0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101
Received data is correct :
Humidity : 0011 0101=35H=53%RH
Temperature : 0001 1000=18H=24℃

Example 2: 40 data is received:

0011 0101	0000 0000	0001 1000	0000 0000	0100 1001
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate :
0011 0101+0000 0000+0001 1000+0000 0000 = 0100 1101
01001101≠0100 1001
The received data is not correct, give up, to re-receive data.

Figura 4.13 Ejemplos lectura bits DHT11

En la figura se observa que el byte de latencia ha de ser igual a la suma del resto de bytes, sino se considera la información enviada como errónea, además, el sensor tiene un periodo de latencia de al menos 2 segundos, por lo que se programan dichas latencias y se ejecutan los métodos en bucle hasta que se obtenga un resultado válido.

```
//Metodo getTempratura de la clase ComunicationRaaspberryPi

public float getTemperatura() throws InterruptedException {
            float temp = 0;
            while(temp == 0.0){ //Se ha de indicar en que pin se conecta (4)
                  temp = dht11.getTemperature(4);
                  Thread.sleep(2000);
            }
            return temp;
      }
```

Clase HandlerDataBase

Esta clase se encarga del almacenamiento, actualización y muestreo de los datos, para ello se ayuda de un fichero de texto llamado **dataBase** que utilizaremos como base de datos. Sobre este fichero trabajan los métodos de la clase **HandlerDataBase** para gestionar la información del sistema de automatización.

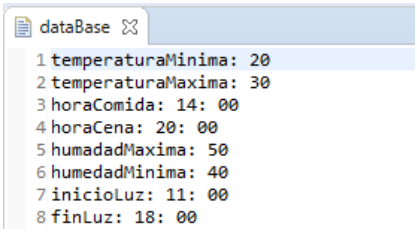


Figura 4.14 Base de datos

La forma de organizar los datos se muestra en la figura 4.14, conociendo la estructura de este fichero podemos crear un objeto de tipo *Scanner* para acceder a la información del fichero y almacenarla por líneas en un *array* de *Strings*. Una vez logrado esto se ha de aislar únicamente el valor numérico de cada línea y convertirlo en un valor de tipo *Int*, como se muestra en el fragmento de código siguiente:

```
	public int getTempMax() {
		Scanner delimitar = new Scanner(arrayData[1]);
		delimitar.useDelimiter("\\s*: \\s*");
		String a = delimitar.next();
		a = delimitar.next();
		int res = Integer.parseInt(a);
		delimitar.close();
		return res;
	}
```

Por último, para la modificación de datos se crea un método, cuya función real es eliminar el fichero existente, crear uno igual sustituyendo los datos por los nuevos y asignarle el mismo nombre y ruta.

```
String newFile = fichero.getParent()+"\\" + fichero.getName();//Obtención
nombre y ruta
fichero.delete();

System.out.println(newFile);

File nuevoFichero = new File(newFile);
BufferedWriter bw;

try
{
	bw = new BufferedWriter(new FileWriter(nuevoFichero,true));
	bw.write("temperaturaMinima: "+tmin+"\r\n" +
```

```
            "temperaturaMaxima: "+tmax+"\r\n" +
            "horaComida: "+mt+ ": "+ mtm +"\r\n" +
            "horaCena: "+dt+ ": "+ dtm + "\r\n" +
            "humedadMaxima: "+hmax+"\r\n" +
            "humedadMinima: "+hmin+"\r\n" +
            "inicioLuz: "+il+ ": "+ ilm +"\r\n" +
            "finLuz: "+fl+ ": "+ flm);
      bw.close();
}catch(Exception e) {
      System.out.println(e);
            }
```

Clase VEML6075

El sensor VEML6075 como explicamos en el apartado de sensores, es el sensor que utilizamos para la detección de la radiación UVB del exterior. Para el funcionamiento de dicho sensor ha sido necesario recurrir al lenguaje Python, ya que no se ha encontrado documentación suficiente para realizarlo con Java, y por el contrario en Python se realiza la obtención de la radiación UVB de manera sencilla, recurriendo a unas librerías concretas para este sensor. Por la tanto se han seguido los pasos para realizarlo en Python [37] y a dicho código se le ha añadido la funcionalidad de escribir la información obtenida en un fichero de texto llamado **UVB.text**. Así que en la clase **VEML6075** de java ejecutamos, con una llamada a la consola, el código para ejecutar la clase Python y después leemos el resultado en el fichero de texto, para finalmente analizar si el dato nos indica que se percibe radiación UVB o no, y devolver un *boolean* en consecuencia.

```
//Ejecución de consola desde Java
String[] cmd = {
                    "/bin/bash",
                    "-c",
                    "python3 veml6075.py"
                    };
            Runtime.getRuntime().exec(cmd);
```

5.2.2 Controlador

En el paquete de controlador se implementa la lógica para el correcto funcionamiento del sistema de automatización del terrario y se conecta la capa de Modelo y Vista, con las correspondientes modificaciones en ambas capas.

Clase HandlerTerrarium

En esta clase se desarrolla el código correspondiente a la lógica del sistema y se modifican los datos de la base de datos, por lo tanto, existen instancias de todas las clases de la capa de Datos, ya que estas incorporan los métodos para la activación y desactivación de los actuadores, la

obtención de datos de los sensores y el acceso a la base de datos. Además, se crea una clase de tipo *Calendar*, debido a que se necesita tener constancia de la hora para la automatización de ciertos procesos.

```
import java.util.Calendar;

	HandlerDataBase hdb = new HandlerDataBase();
	ComunicationPlaca terrarium = new ComunicationRaspberryPi();

	Calendar now;
```

Para la automatización de la temperatura se ha tratado de recrear las condiciones normales del exterior aún en las mejores condiciones, por esta razón se ha tenido en cuenta el horario para este factor. Al trabajar con una temperatura mínima y otra máxima se ha programado para que, en un horario considerado nocturno, tomando de referencia las horas de luz indicadas por el usuario, la temperatura estuviera más cerca de la temperatura mínima, y en un horario considerado diurno la temperatura se aproximara más a la temperatura máxima. Con el fin de realizar esta tarea de la mejor manera posible se saca la temperatura media, para que durante el horario diurno la temperatura no baje de ese umbral, y durante la noche no pase de él.

El código se divide en horarios, y la condición de activación y desactivación de actuadores se realiza en función del propio estado de los actuadores y por supuesto, de la temperatura en la que se encuentra el terrario.

```
//código horario diurno


//Si la temperatura actual es mayor que la temperatura máxima se debe activar el ventilador, si este está apagado, pero antes se comprueba si el calefactor está activado y si lo esta se desactiva.

if(terrarium.getTemperatura() > hdb.getTempMax() + latencia){
	//la variable latencia será 0, salvo que la diferencia entre la temperatura máxima y mínima sea menor que 5, en cuyo caso se le asigna un valor entre 2 y 5. Con esto conseguimos que los actuadores no estén activos todo el rato.

	if(terrarium.isHeaterHigh()) terrarium.deactHeater();
	if(terrarium.isVentilatorLow()) terrarium.actVentilator();

}
//Si no se cumple lo anterior y la temperatura actual es inferior a la media se debe activar el calefactor, si este está apagado, pero antes si el ventilador está activado se debe desactivar.

else if (terrarium.getTemperatura() <= temperaturaMedia - latencia {
	if(terrarium.isVentilatorHigh()) terrarium.deactVentilator();
	if(terrarium.isHeaterLow()) terrarium.actHeater();
}
```

```
else {
	if(terrarium.isVentilatorHigh()) terrarium.deactVentilator();
	if(terrarium.isHeaterHigh()) terrarium.deactHeater();
	}
```

El código del horario nocturno es muy similar, pero buscando que la temperatura se encuentre entre la mínima y la media.

En el caso de la humedad la automatización es parecida, no se separa por horarios pero se tiene en cuenta el nivel de humedad y si están funcionando los actuadores, principalmente el humidificador, pero también se necesita saber el estado del ventilador y el calefactor, lo cual se verá más claro con el código. Es importante recordar también que la temperatura y la humedad están ligadas, ya que activando el ventilador se baja el nivel de humedad.

```
public void automaticHumidity() throws InterruptedException {

	float mediaHumidity = (hdb.getHumMax() + hdb.getHumMin())/2;

//Margen es como la variable latencia del método automaticTemperature, si la humedad mínima y máxima están muy próximas se les asocia un margen para que los actuadores no estén siempre activos.


	if (hdb.getHumMax() - hdb.getHumMin() <= 1) {
			margen = 4;
	}
	else if(hdb.getHumMax() - hdb.getHumMin() <= 5)
			margen = 2;

	if(terrarium.isHumidificatorHigh()) {//Si el humidificador está activo

//si la humedad actual sobrepasa el límite, se desactiva el humidificador, y además si el ventilador y el calefactor están apagados, se activa el ventilador. Esto es porque si ambos están apagados implica que están en una temperatura intermedia entre el máximo y la temperatura media o entre el mínimo y la temperatura media, dependiendo de la hora en la que nos encontremos, y si estamos en dicha temperatura intermedia podemos activar el ventilador favoreciendo la humedad y sin perjudicar la temperatura.

		if(terrarium.getHumedad() > hdb.getHumMax() + margen) {
			terrarium.deactHumidificator();
			if(!terrarium.isVentilatorHigh() && !terrarium.isHeaterHigh())
				terrarium.actVentilator();
			}
//Si la humedad es mayor que la humedad media, desactivamos el humidificador
		else if(terrarium.getHumedad() > mediaHumidity)
				terrarium.deactHumidificator();
	}else{ //si el humidificador está desactivado.

//Y la humedad es menor que la humedad mínima menos el margen, activamos el humidificador.
```

```
if(terrarium.getHumedad() <= hdb.getHumMin() - margen)
                    terrarium.actHumidificator();

            }
      }
```

Para el método de las horas de luz se trabaja con dos factores, la detección de luz solar y los horarios fijados. Si el sensor de radiación UVB nos indica que se detecta dicha radiación proveniente desde fuera del terrario, no se debe activar la bombilla UVB, y en caso contrario se debe activar durante las horas fijadas como horario diurno.

```
public void automaticLight() throws IOException {
            int horarioIni = hdb.getIniLight(0) * 60 + hdb.getIniLight(1);
            int horarioFin = hdb.getEndLight(0) * 60 + hdb.getEndLight(1);

//Si la hora actual es mayor a la hora fijada como inicio de la luz y menor a
la hora de fin y no se detecta radiación UVB del exterior, se debe activar la
bombilla UVB.
            if (tiempoMins >= horarioIni && tiempoMins < horarioFin &&
!terrarium.getHighLight())
                    terrarium.actLight();

// Si no se da dicha circunstancia y la bombilla UVB está activa, se debe
desactivar.
            } else if(terrarium.isLightHigh()) terrarium.deactLight();
      }

{//la variable tiempoMins hace referencia a la hora actual pasada a minutos,
para compararla con la hora de inicio (horarioIni) y la hora de fin
(horarioFin) del periodo diurno también pasadas a minutos en la parte de
arriba.
```

Respecto al método del suministro de comida se ha de suministrar siguiendo los horarios indicados desde la aplicación y se debe detectar el nivel del depósito. Este método se ha realizado para el caso en el que se alimente a los reptiles dos veces al día.

```
public void automaticFood(){
      int horarioComida = hdb.getMealTime(0) * 60 + hdb.getMealTime(1);
      int horarioCena = hdb.getDinnerTime(0) * 60 + hdb.getDinnerTime(1);

//Para la comparación horaria se hace lo mismo que antes, pasarlo todo a
minutos.
      if (tiempoMins >= horarioComida && contComida < 1) {//contComida = 0
      //Si la hora actual es mayor o igual que la hora de comer y todavía no
se ha suministrado la comida, entonces se abre la válvula hasta que la comida
llego al nivel del sensor. En este punto se cierra la válvula y se actualiza
el contador de comida para saber que esta ya se ha suministrado.
            while(!terrarium.isHighFood()) {
                    terrarium.actFoodValve();
```

```
                    }
                terrarium.deactFoodValve();
                contComida++;

                                                    //contCena = 0
        }else if (tiempoMins >= horarioCena && contCena < 1) {
                while(!terrarium.isHighFood()) {
                        terrarium.actFoodValve();
                }
                terrarium.deactFoodValve();
                contCena++;

        //Cuando se cambia de día y todavía no se ha llegado al horario de la
primera comida, se pasa el horario tanto del contador de la primera comida como
el de la segunda a 0.
      } else if(tiempoMins > 0) {
                        contComida = 0;
                        contCena = 0;
                }
        }
```

Muy similar es el caso del método del suministro de agua, que en lugar de suministrar por horarios tiene un sensor para el suministro de agua, cuando el nivel de esta dentro del terrario sea bajo. Al igual que con la comida, debe tener un sensor en el depósito para alertar cuando el nivel de agua en este sea bajo.

```
if(terrarium.isLowFoodTank()){
                        terrarium.warningFood();
                }
```

Clase Main

La clase Main por supuesto es la clase principal del proyecto, desde la que se lanzan todas las funciones, por lo que contiene una instancia de la clase **HandlerTerrarium** donde se encuentra la lógica de todo el sistema, y otra a la clase **HandlerDataBase**, para poder modificar y guardar la base de datos. También es la clase donde se implementa el Socket TPC de la parte del servidor.

El método para la ejecución de la lógica se llama **automatic()** y en él es necesario crear una nueva instancia de la clase **HandlerTerrium**, ya que en esta clase se actualiza la base de datos que estamos utilizando, y si no se actualiza, los cambios que realizamos desde la aplicación Android no se tendrían en cuenta. Este método además de ejecutarse cada vez que se realice un cambio en las condiciones del terrario desde la aplicación, también debe ejecutarse cada treinta minutos para activar o desactivar los actuadores si fuera necesario. Para ello se ha tenido que crear una variable de tipo *Runnable* en la que se introduce este método, y crear un organizador (*Scheduler*), con los que permitimos al método ejecutarse cada media hora.

```
private static final ScheduledExecutorService scheduler =
Executors.newScheduledThreadPool(1);

public static void automatic() throws InterruptedException {
		ht = new HandlerTerrarium();
		ht.automaticTemperature();
		ht.automaticHumidity();
		ht.automaticLight();
		ht.automaticFood();
		ht.automaticWater();
	}
final static Runnable a = new Runnable() {
		public void run() {
			try {
				automatic();
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	};
```

```
scheduler.scheduleAtFixedRate(a, 0, 30, TimeUnit.MINUTES);
```

Para el Socket se crea un bucle infinito para que esté disponible siempre que desde la aplicación se trate de establecer conexión. Se envía la humedad y temperatura actual, y se reciben las nuevas condiciones del terrario.

```
public static void main(String args[]) throws IOException, InterruptedException {
		ServerSocket serv = new ServerSocket(9000);
		System.out.println("conectado");

	while(true){
			Socket socket = serv.accept();
			BufferedReader entrada = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
			tempMin = entrada.readLine();


				...

	PrintWriter salida = new PrintWriter(socket.getOutputStream(), true);
			salida.println(temperatura);
			salida.println(humedad);

	hdb.setDataBase(tempMin, tempMax, horaComida, minutComida, horaCena,
minutCena, humedadMax, humedadMin, hourLightIni, minutLightIni, hourEndLight,
minutEndLight);
			automatic();
			entrada.close();
			salida.close();
			socket.close();

}
```

5.3 Desarrollo Hardware

En este apartado se va a exponer el conexionado y distribución de pines usados en nuestro proyecto para los diferentes dispositivos. Cabe destacar que se va a mostrar la forma de conexión para todos los dispositivos necesarios para la creación del sistema, pero al no disponer de todos ellos el resultado final del sistema será diferente.

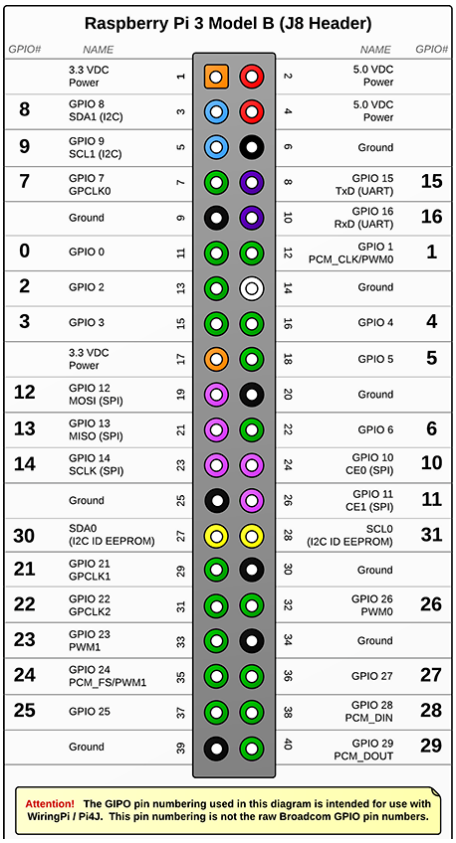


Figura 4.15 Numeración Pines Raspberry Pi 3B+ [37]

5.3.1 Conexionado

La Raspberry Pi consta de numerosos pines, con funcionalidades distintas. En primera instancia vamos a dividirlos en 3 grupos, pines de masa, pines de alimentación y pines de control o datos. Para la conexión y control de los dispositivos, todos ellos deben ir conectados a estos 3 tipos de pines, de esta manera la Raspberry Pi proporciona la polaridad positiva y negativa necesaria en un circuito eléctrico además de un pin para interactuar con el dispositivo.

Siempre que se necesite un máximo de 5 voltios, con la Raspberry Pi será suficiente para el suministro de energía, como es el caso del sensor DHT11.

- DHT11

El sensor DHT11 consta de 3 pines, uno de tierra (GND), uno de alimentación (VCC) y uno de datos (DATA). Para la alimentación necesita una tensión de 5V por lo que el pin VCC conectará con el pin 2 de la Raspberry PI, para la masa usará el pin 6 correspondiente a GND, y para los datos con el pin GPIO 4.

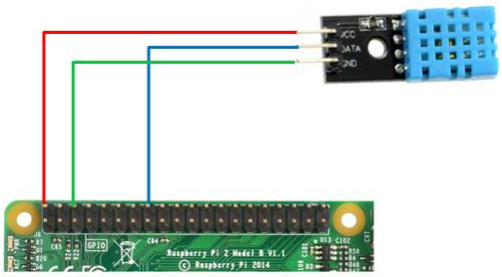


Figura 4.16 Conexionado DHT11

Para los dispositivos que necesitan un mayor voltaje será necesario el uso de elementos que proporcionen una fuente de alimentación mayor.

- Placa de relés

Una place de relés es una solución a este problema, debido a que puede conectarse a la Raspberry Pi ocupando los pines de los sensores a los que va a proporcionar un mayor voltaje, ya que para su funcionamiento solo necesita 5 voltios, y al mismo tiempo permite la conexión entre una fuente de alimentación mayor y el dispositivo que la necesita sin que la Raspberry Pi pierda el control sobre dicho dispositivo. Vamos a mostrar cómo quedaría dicha conexión poniendo el ejemplo del ventilador, que necesita una tensión de 220V.

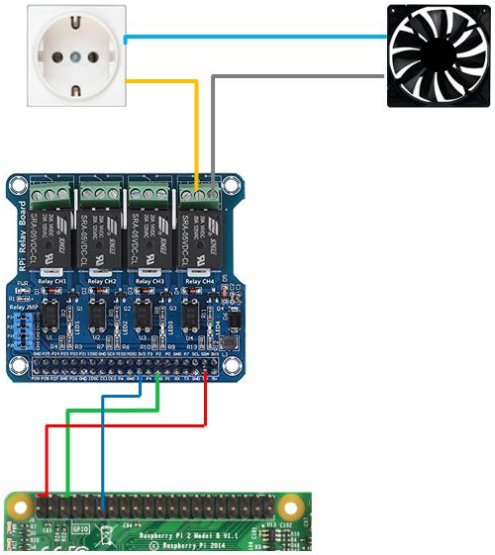


Figura 4.17 Conexionado empleando Relé

De esta manera hasta que el pin de control de la Raspberry no cambie de estado y active la bobina del relé, este no cerrará el circuito entre el ventilador y la fuente de alimentación, y por la tanto el ventilador no se activará.

- VEML6075

Determinados dispositivos necesitan emplear pines especiales de la Raspberry Pi para su interacción con esta, ya que utilizan protocolos de comunicación que no tienen todos los pines de la Raspberry. Es el caso del sensor VEML6075, el cual utiliza el protocolo de comunicación I2C. I2C [39] es un protocolo de comunicación síncrono el cual emplea dos líneas para la comunicación, una para la señal de reloj y la otra para los datos. Por esto debemos emplear dos pines de control para la obtención de datos de este sensor, en concreto dentro de los pines de control debemos emplear los dos que incluyen comunicación I2C, el pin GPIO 8 (SDA), para los datos, y el pin GPIO 9 (SCL), para el reloj. Por lo tanto, el sensor VEML6075 debe conectarse a 4 pines, los dos pines GPIO, al pin de alimentación de 3.3 voltios y al pin de tierra.

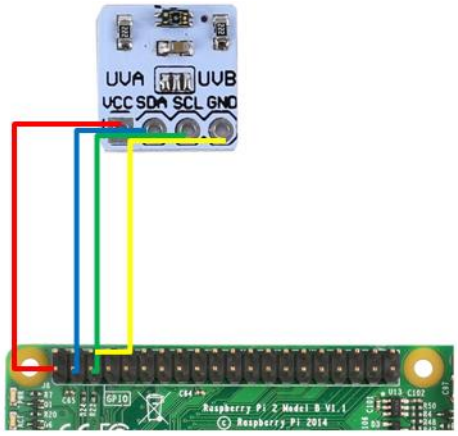


Figura 4.18 Conexionado sensor VEML6075

5.3.2 Distribución de pines

La distribución de pines GPIO quedaría de la siguiente manera:

- El pin GPIO 0, ubicado en la posición 11 será el encargado de controlar el ventilador.
- El pin GPIO 1, posición 12, se encargará de la manta eléctrica.
- El pin GPIO 2, posición 13, será el encargado del humidificador.
- El pin GPIO 3, posición 15, corresponderá a la bombilla UVB.
- El pin GPIO 4, posición 16, se encargará de la lectura del sensor DHT11.
- El pin GPIO 5, ubicado en la posición 18, corresponderá a la válvula de comida.

- El pin GPIO 6, ubicado en la posición 22, corresponderá a la válvula de agua.
- El pin GPIO 7, posición 7, se encargará del sensor de nivel de comida.
- El pin GPIO 21 y 22 corresponderán con los sensores de nivel alto y bajo de agua.
- Los pines GPIO 23 y GPIO 24 corresponderán con las válvulas de nivel de agua y comida de los depósitos.
- Los pines GPIO 8 y 9, se usarán para el sensor de radiación UVB, ya que estos dos pines corresponden con los pines SCL y SDA del bus I2C.

Para la alimentación se usarán los pines 1 y 2, que corresponden a la salida de 3.3 y 5 voltios respectivamente, y para la línea de masa o tierra se usara el pin 6.

5.3.3 Sistema final

Con el fin de simular la lógica del sistema, vamos reemplazar los dispositivos que nos faltan por otros. En el caso de los actuadores serán reemplazados por ledes, que suplantaran el estado de los dispositivos a los que reemplazan. De esta manera cuando el actuador debería activarse se encenderá el led. En el caso de los sensores de nivel de agua y comida se reemplazarán por un pulsador, de tal forma que la acción de pulsar equivaldrá a un cambio de estado en el sensor.

La conexión con estos dispositivos se realiza de la siguiente manera.

- **Ledes.** Tienen dos patas, una negativa o cátodo y otra positiva o ánodo, y debe conectarse a el pin GPIO, que controla su estado, con la pata positiva, y a el pin de tierra con la pata negativa. Se debe incorporar también a este circuito una resistencia de 220 Ω.

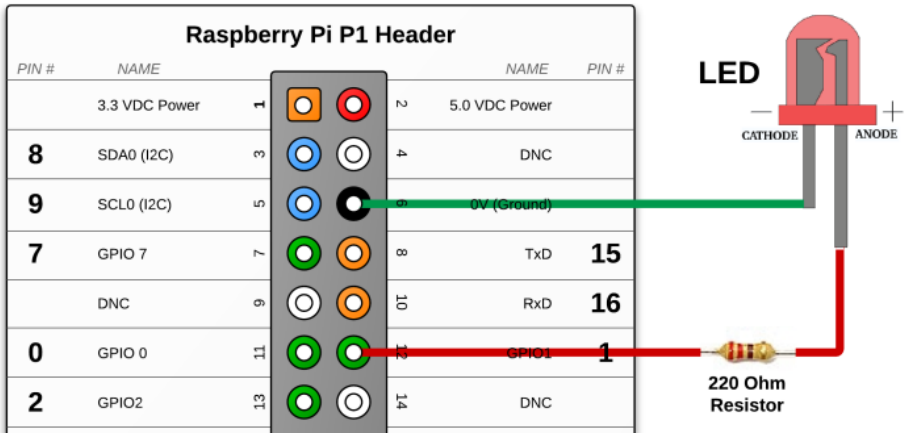


Figura 4.19 Cableado ledes [38]

- **Pulsadores.** En el caso de los pulsadores también únicamente se necesitan 2 conexiones, una a la tensión de 3.3V y la otra al pin controlador GPIO. Esto es debido al hecho de que

el funcionamiento del pulsador consiste en cerrar el circuito cuando este está pulsado, por lo que la conexión a tierra estará en la parte del actuador, en este caso el led.

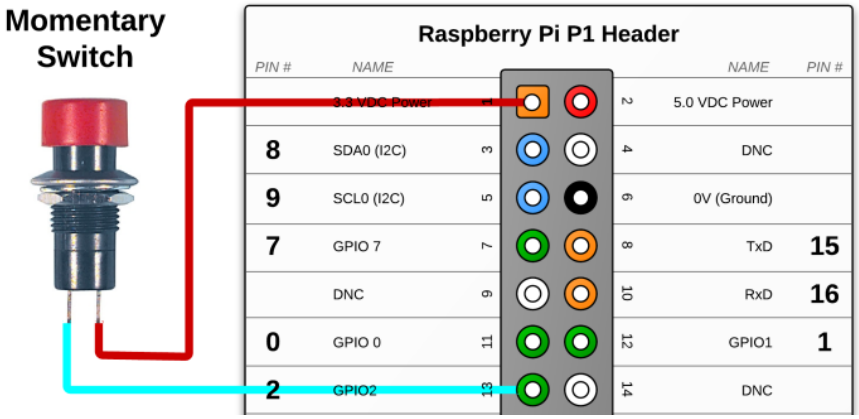


Figura 4.20 Conexión con pulsador [38]

6. Implantación

La configuración software que hace posible la realización de este proyecto consiste principalmente en la instalación de un sistema operativo en la Raspberry Pi y la configuración de esta para poder trabajar con ella correctamente. Existen otros elementos software importantes como Eclipse Oxygen y Android Studio, pero en ambos casos no se requiere una configuración especial para su instalación, únicamente se ha descargado la última versión de ambos. En este apartado vamos a detallar la configuración para la puesta en marcha del software que hace posible la implantación de la solución.

6.1 Sistema operativo

La instalación del sistema operativo en la Raspberry Pi pasa por una tarjeta micro-SD, en la cual se debe introducir dicho sistema operativo mediante un ordenador. Una vez tenemos la tarjeta SD conectada al ordenador, debemos descargar el sistema operativo deseado, en nuestro caso Raspbian, para ello vamos a la página principal de documentación de Raspberry Pi, al apartado de descargas [40] y, de entre las muchas alternativas de sistemas operativos compatibles con la Raspberry Pi, accedemos a la última versión de Raspbian. Actualmente le han cambiado el nombre a Raspbian por Raspberry Pi OS, pero el procedimiento es el mismo.

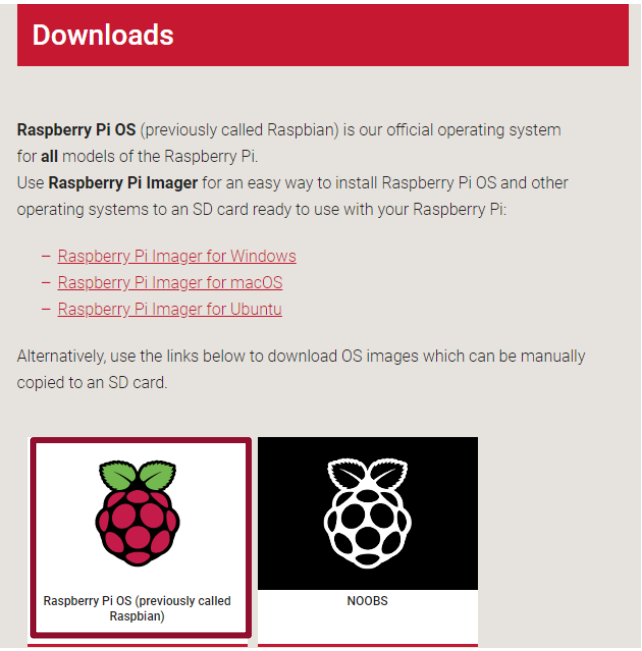


Figura 5.1 Página de descarga Raspbian (Raspberry Pi OS)

A continuación, se muestran tres opciones para la descarga de este sistema operativo, una sin interfaz gráfica y dos que sí la tienen. Para este proyecto se podría descargar cualquiera de las tres, pero para mayor facilidad en la interacción con nuestra Raspberry Pi se escoge una opción con interfaz gráfica. Los sistemas operativos están almacenados en archivos del tipo imagen (*.img*) utilizados para hacer copias de la totalidad de la información de un sistema en un volumen de unidad.

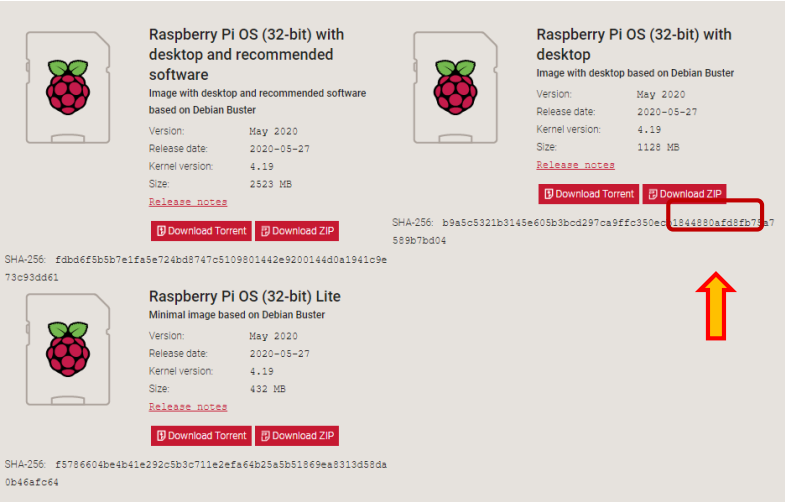


Figura 5.2 Opciones imagines Raspbian (Raspberry Pi OS)

Cuando tenemos descargado el archivo *.zip*, que contiene la imagen del sistema operativo, se podría descomprimir y obtener el archivo *.img* pero no es necesario, ya que para la escritura de este en la tarjeta SD se necesita un programa destinado a ello, el cual soporta archivos *.zip*. El programa que utilizamos para este propósito se llama **BalenaEtcher**, el cual está disponible para sistemas operativos Windows, macOS y Linux. Los pasos para la instalación y utilización de este programa son sencillos:

- En primer lugar, se descarga de la página oficial [41].

Figura 5.3 Descarga balenaEtcher

- Y en segundo se siguen los pasos que indica el programa, primero se selección la imagen, luego se selecciona la tarjeta donde se desea escribir la imagen y por último se ejecuta.

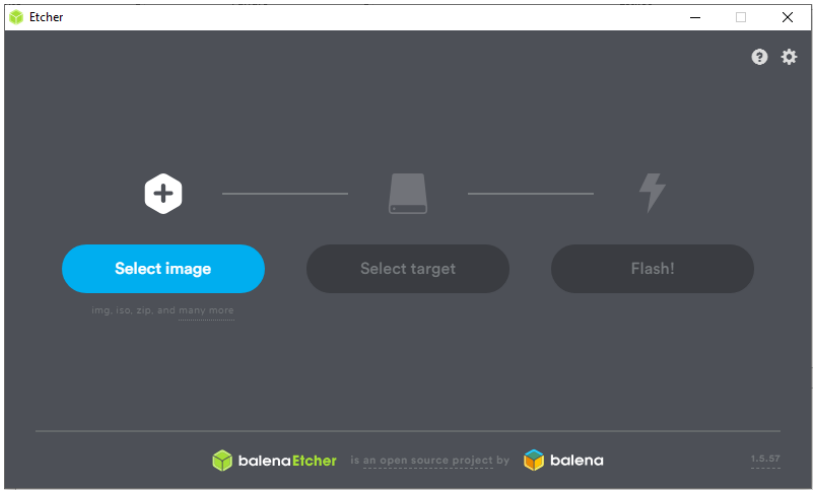


Figura 5.4 Indicaciones funcionamiento balenaEtcher

Una vez completado este procedimiento, el sistema operativo está correctamente escrito en la tarjeta micro-SD, la cual se puede desconectar del ordenador e introducir en la Raspberry Pi. Conectamos la Raspberry Pi a la corriente y esta se ejecuta con el sistema operativo Raspbian.

6.2 Configuración sistema operativo

En este punto vamos a mostrar los pasos para permitir la conexión desde otro ordenador y también mostraremos como se habilita o deshabilita la conexión con ciertos tipos de pines.

Hay dos formas de acceder a la configuración para modificar con las opciones de interfaz, la primera es clicando el botón desplegable, con el logotipo de Raspberry, que se encuentra en el escritorio de nuestro sistema operativo, y desde ahí acceder a *Preferencias/Coniguración Raspberry Pi*, como se muestra en la figura 5.5.

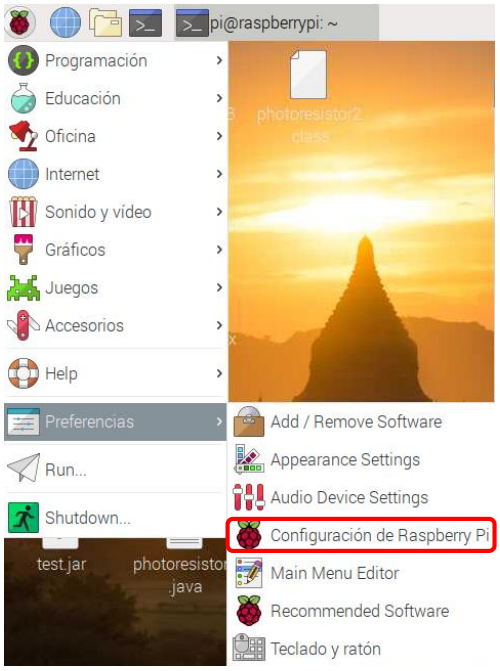


Figura 5.5 Menu Raspberry Pi

Figura 5.6 Configuracion Raspberry (Opción 1)

La segunda es a través de la consola con el comando *raspi-config*, con el que se accede a la siguiente pantalla.

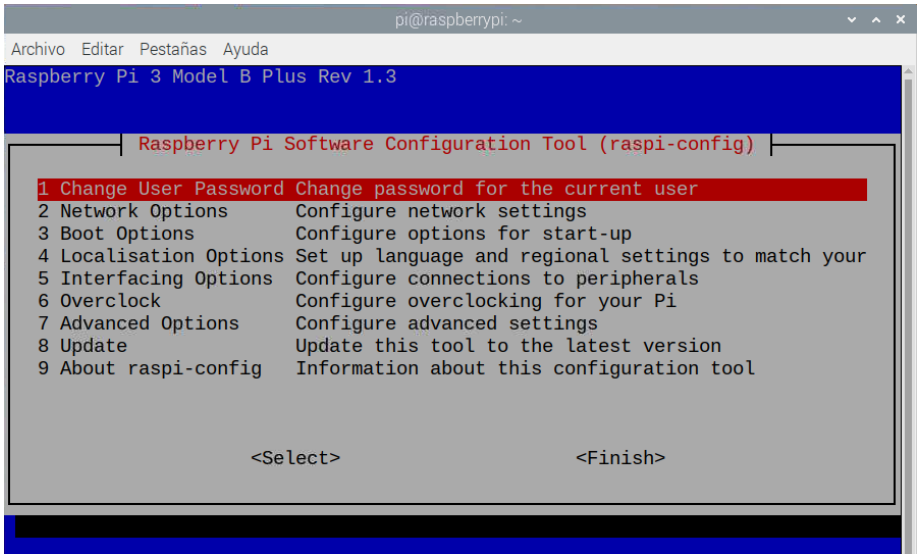


Figura 5.7 Configuracion Raspberry Pi (Opción 2)

Desde ambas se pueden modificar muchas particularidades de nuestro sistema, como por ejemplo la resolución de pantalla, el idioma, la hora, etc. Pero el aspecto más importante a modificar para la realización de nuestro proyecto son las interfaces, desde donde podemos habilitar y deshabilitar la cámara, el protocolo VNC, el protocolo SSH, o buses de comunicación de los pines de la Raspberry Pi, como SPI e I2C. En nuestro caso habilitaremos SSH y VNC para la comunicación con el ordenador, I2C para la conexión con el sensor VEML6075, y SPI para el posible uso de dispositivos analógicos.

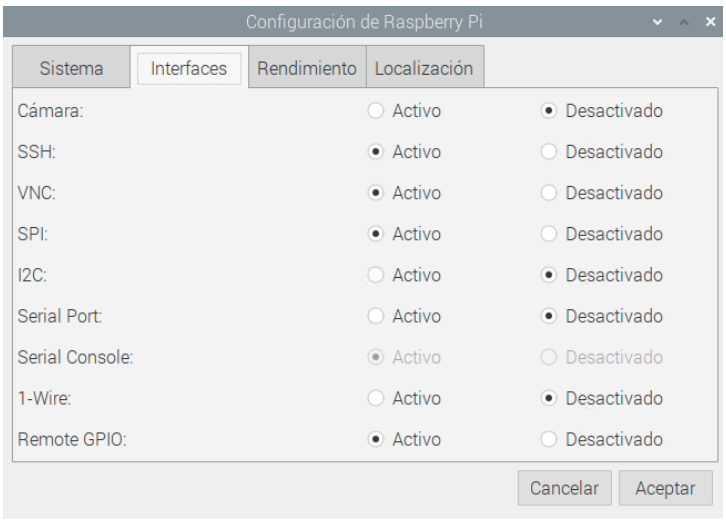


Figura 5.8 Configuración interfaces

7. Pruebas

En este punto vamos a registrar las pruebas que se han realizado para el testeo del proyecto, tanto por la parte del sistema de automatización comandado por la Raspberry Pi, como por la parte de la aplicación móvil.

7.1 Prueba aplicación móvil

Para comprobar el correcto funcionamiento de la aplicación, se han probado todas las pantallas y comprobado que la conexión con la Raspberry Pi se realizada de forma efectiva. Se ha observado que la aplicación no fallaba y que los datos mostrados en ella, así como los cambios realizados en el sistema de automatización, se guardaban y mostraban correctamente. Vamos a indicar los pasos seguidos.

Primero abrimos la aplicación sin conectar la Raspberry Pi y esto es lo que nos muestra:

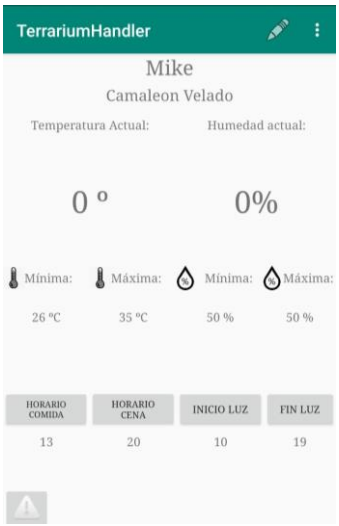


Figura 6.1 Aspecto aplicación sin conexón

A continuación, se conecta la Raspberry y se ejecuta el *.jar*, lo que inicia el proceso de obtención de los datos de temperatura y humedad antes de habilitar el puerto 9000 para la conexión del Socket. Una vez obtenidos ambos datos se permite la conexión con la aplicación.

```
^Cpi@raspberrypi:~/Desktop/PruebasJAR $ sudo java -classpath
conectado
Temperatura y humedad
0.0
30.0
55.0
Conexión con aplicación realizada
```

Figura 6.2 Consola de la Raspberry Pi al inicar la conexión

Cuando el sistema está activo, se abre la aplicación para observar si la conexión se ha realizado correctamente y se muestran los datos de la temperatura y humedad actual junto a los valores actuales del resto de parámetros modificables.

Figura 6.3 Aspecto aplicación con conexión

El siguiente paso sería comprobar la otra pantalla de la aplicación y el resto de funcionalidades. Clicamos sobre el botón superior derecho con forma de lápiz, para entrar en la pantalla de edición de las condiciones del terrario.

Figura 6.4 Pantalla de edición

Modificamos el contenido de dicha pantalla y le damos al botón de guardar, ubicado en la parte superior derecha con forma de disquete.

Figura 6.5 Modificación de valores

Dicho botón debe llevarnos a la pantalla principal, donde se mostrarán los valores modificados. En el caso de no querer guardar los cambios se le da al botón de la cruz, justo a la izquierda del botón de guardar.

Figura 6.6 Pantalla inicio tras modficación de datos

Observamos que todos los cambios se muestran de forma correcta, y accedemos a la base de datos para observar si la transmisión de datos entre la aplicación y el sistema se ha producido también correctamente.

```
temperaturaMinima: 29
temperaturaMaxima: 40
horaComida: 13: 00
horaCena: 20: 00
humadadMaxima: 60
humedadMinima: 40
inicioLuz: 10: 00
finLuz: 19: 00
```

Figura 6.7 Base de datos después de la modificación

Dentro de la aplicación solo nos quedaría comprobar si la implementación de la funcionalidad para modificar los valores de forma individual funciona correctamente. Para ello pulsamos en uno de los valores dentro de la pantalla principal, por ejemplo, pulsaremos el de la temperatura mínima.

Figura 6.8 Pantalla de modificación rápida valores

La función se ejecuta según lo esperado, y la modificación y comunicación para la actualización de la base de datos también se produce satisfactoriamente. Por último, para testear la fiabilidad de la aplicación se realizan múltiples modificaciones de valores, algunas de ellas con una hora de diferencia y sin parar la ejecución del sistema en la Raspberry Pi, y no se produce ningún error.

7.2 Prueba sistema automatización

En cuanto a las pruebas en el sistema de automatización, se ha comprobado el correcto funcionamiento de la lógica a través de una simulación realizada con sensores y ledes en lugar de actuadores, con la particularidad de que se emplea un pulsador para simular el sensor de nivel de los tanques de agua y comida. En dicha simulación se activa el sistema, se comprueba que la comunicación con la aplicación se realiza de forma correcta y sin necesidad de reinicio, ya que se queda permanentemente esperando a la apertura de nuevas comunicaciones por parte del cliente, y se observa que los ledes se encienden o apagan en función de las condiciones fijadas. A través de la aplicación variamos las condiciones con el fin de que el sistema apague y encienda todos los ledes y de esta forma se pueda comprobar el correcto funcionamiento de la lógica del sistema. Para verificar que las transmisiones entre la aplicación y la Raspberry Pi se realizan de forma correcta también se puede observar si la base de datos se modifica acorde con los datos editados, tal y como se ha mostrado en la imagen 6.7 del apartado 7.1.

Mostramos una imagen de la conexión, con un led, un sensor, una resistencia y la Raspberry Pi:

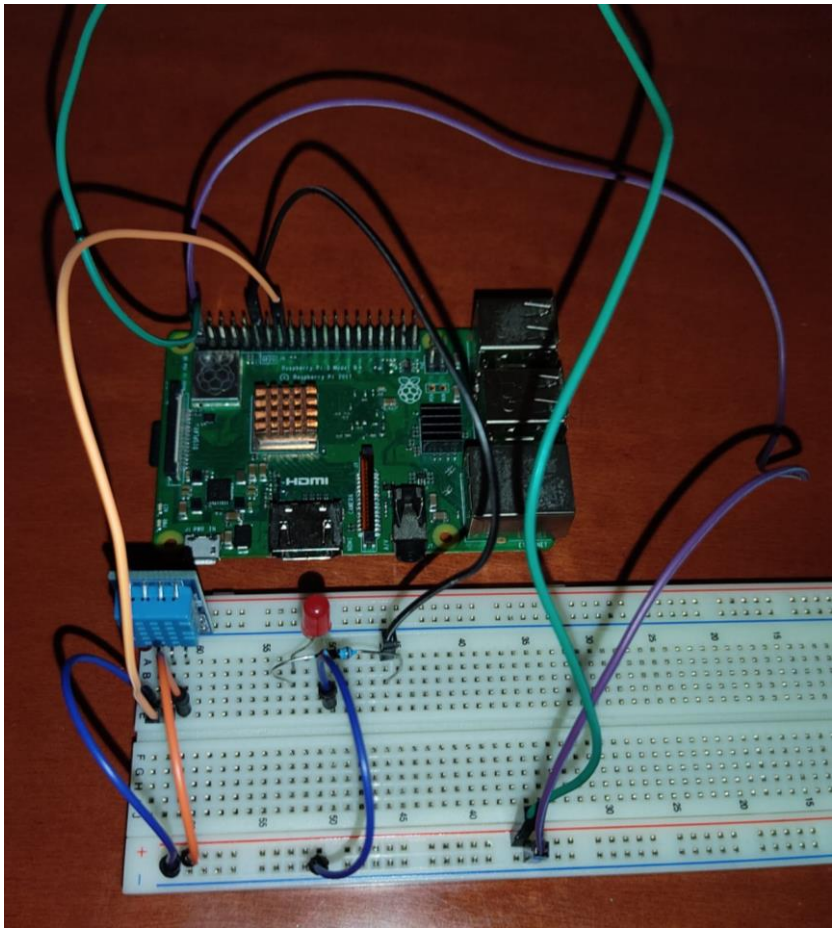


Figura 6.9 Conexionado simple

Y mostramos también, el aspecto del sistema de simulación total, aunque por el exceso de cables es complicado distinguir todas las conexiones.

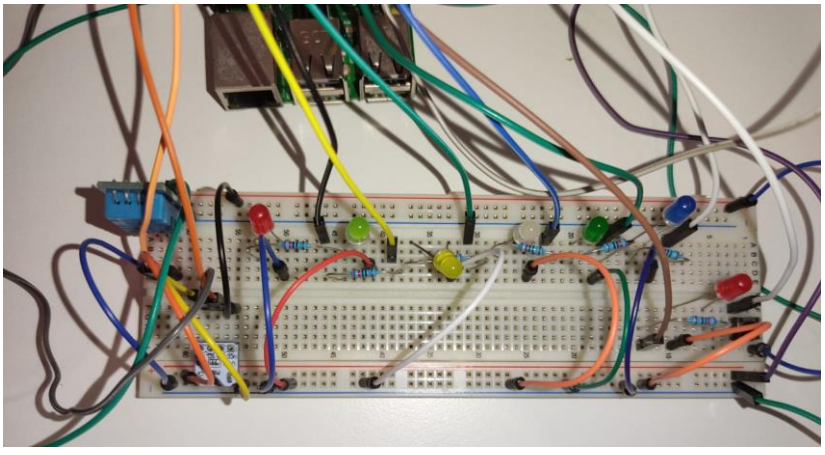


Figura 6.10 Conexionado sistema de simulación

8. Conclusiones

El objetivo principal de este proyecto era automatizar la temperatura y la humedad de un terrario de reptiles, empleando para ello sensores, actuadores y una placa base, y crear una aplicación móvil para el control de estos factores. Durante el periodo de investigación de cómo realizar dicha tarea, se creyó conveniente que se debía ir más allá y buscar la automatización de otros factores también importantes para la supervivencia y bienestar de los reptiles, consiguiendo así un sistema que generara un entorno adecuado para la vida del reptil y facilitándole el cuidado a los dueños.

Durante la realización nos hemos encontrado con diversos problemas como el tiempo de espera al simular la aplicación, que se solucionó conectado el móvil al ordenador y simulando la aplicación en el móvil; se realizó hasta tres veces la escritura del sistema operativo Raspbian en la SD pensando que no funcionaba y resulto ser que el cable HDMI conectado a la Raspberry Pi estaba roto. También se han cometido errores como la compra de un sensor analógico pensando que era digital; no declarar una clase de tipo *Activity* en el *scheduler* de la aplicación, la cual cosa provocaba que no funcionara; un fallo en el conteo de los pines, que provocó que se pensara que el código para el sensor DHT11 no funcionaba. Pero sin duda, el peor problema y error ha sido la no obtención de todos los dispositivos necesarios para la realización del proyecto.

Con todo ello y viendo que los objetivos desde un punto de vista más amplio eran la creación de un sistema de automatización de factores climáticos y factores relacionados con la nutrición de los reptiles, y la creación de una aplicación móvil para la gestión de dicho sistema. Se podría decir, por lo tanto, que los objetivos a nivel software, en cuanto a la creación de una aplicación funcional que conecta e intercambia datos con el sistema de automatización, y la creación de la lógica de dicho sistema, se han cumplido, pero desde el punto de vista de la funcionalidad total, nos hemos encontrado con la falta de dispositivos necesarios.

En cuanto a lo aprendido, en el ámbito personal me ha servido para no bajar los brazos ante los problemas que surgían y pelearme con ellos, buscando información y creyéndome capaz de solventarlos de forma efectiva por mí mismo, y también me ha servido para organizarme de una forma más eficiente el tiempo y el trabajo. En el aspecto profesional, con el fin de realizar el trabajo de la mejor manera posible, se ha aprendido a desarrollar aplicaciones móviles, lo cual no se ha impartido de forma directa en la carrera y se ha necesitado investigar profundamente, también se ha aprendido a trabajar con Raspberry Pi y demás dispositivos electrónicos, para lo cual también se ha dedicado bastante tiempo, y por último, se han aprendido nociones de un lenguaje de programación no impartido en el grado, llamado Kotlin, aprovechando la oportunidad de tener que realizar una aplicación en Android.

8.1 Relación del trabajo desarrollado con los estudios cursados

En el grado de ingeniería informática no se imparten asignaturas que comprendan todos los aspectos tocantes a este proyecto, pero por separado sí que existen asignaturas que proporcionan herramientas y nociones para saber enfrentarse a la realización del proyecto. Existen asignaturas en las que se enfoca la parte hardware como Diseño de sistemas digitales, donde trabajamos con una placa base; asignaturas orientadas a la automatización como Control por computador; asignaturas donde se muestra la utilización de sockets como Redes de computadores; y asignaturas para la creación de aplicaciones como Ingeniera del Software.

9. Trabajos futuros

Este proyecto tiene muchas vías para continuar trabajando en él y por supuesto mejorarlo. La mejora más evidente sería conseguir todos los dispositivos necesarios para cubrir la lógica realizada en este proyecto, es decir, todos los dispositivos expuestos en el apartado 3.2, e instalar el sistema en un terrario. Se van a presentar una serie de trabajos futuros, independientes al problema de la falta de sensores:

1. Mejorar el aspecto de la aplicación móvil, incorporando un diseño más llamativo y menos cuadrático.
2. Programar la aplicación para que, en caso de niveles de comida y agua bajos, llegara una notificación a la pantalla de inicio del móvil sin la necesidad de haber abierto la aplicación.
3. Contactar con nuestro proveedor de internet para que nos habilitara una dirección IP externa fija para la Raspberry Pi, y de esta manera poder acceder a la aplicación y gestionar el sistema, aunque no estemos conectados al mismo Wifi que la Raspberry Pi.
4. Instalar el sistema en un terrario de una marca dedicada a la fabricación de este tipo de artículos, y comprobar su funcionamiento.
5. Introducir en la aplicación una funcionalidad para añadir más terrarios, y que de esta manera desde el mismo dispositivo móvil se puedan gestionar las condiciones de tantos terrarios como se desee. Esto sería perfecto para amantes de los reptiles, los cuales tiene varios en casa o para tiendas de reptiles, ya que facilitarían el cuidado de estos.
6. Dar la opción de modificar el número de comidas que realizará el reptil durante el día o incluso durante la semana, ya que hay reptiles que no necesitan comer dos veces al día e incluso se alimentan tan solo un par de veces por semana, y por supuesto que el sistema se adapte a esta modificación.

10. Referencias

[1] Zoo Med Laboratories, Inc. https://zoomed.com/. [En línea] HygroTherm™ Humidity & Temperature Controller, 2018. [Último acceso: 20 de Noviembre de 2019.] https://zoomed.com/hygrotherm-humidity-temperature-controller/.

[2] Amazon. https://www.amazon.es/. [En línea] YCDJCS Terrarios. [Último acceso: 20 de Noviembre de 2019.] https://www.amazon.es/Terrarios-YCDJCS-Insectos/s?rh=n%3A13441944031%2Cp_6%3AAVADRJJ9JN8W5.

[3] Zoo Med Laboratories, Inc. http://es.zoomed.eu/. [En línea] Zoo Med Laboratories Terrariums, 2018. [Último acceso: 20 de Noviembre de 2019.] http://es.zoomed.eu/category/reptile/terrariums/.

[4] ESPECIESPRO. https://especiespro.es/. [En línea] La temperatura de los reptiles , 29 de Abril de 2016. [Último acceso: 12 de Enero de 2020.] https://especiespro.es/articulos/la-temperatura-de-los-reptiles/.

[5] Tiendanimal. https://www.tiendanimal.es/. [En línea] Que es la radiación UVB y quá lámpara es mejor para tu terrario, 2011. [Último acceso: 18 de Febrero de 2020.] https://www.tiendanimal.es/articulos/que-es-la-radiacion-uvb-y-que-lampara-es-la-mejor-para-tu-terrario/.

[6] Xataka. https://www.xataka.com/. [En línea] Qué es Arduino, cómo funciona y qué puedes hacer con uno, 3 de Agosto de 2018 [Último acceso: 19 de Diciembre de 2019.] https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno#:~:text=Arduino%20es%20una%20plataforma%20de,para%20los%20creadores%20y%20desarrolladores..

[7] INSTITUTO NACIONAL DE FORMACIÓN TÉCNICA PROFESIONAL. https://sistemaopertaivoi.wordpress.com/. [En línea] Arduino [Último acceso: 17 de Enero de 2020.] https://sistemaopertaivoi.wordpress.com/arduino/

[8] HZ Hard Zone. https://hardzone.es/. [En línea] Análisis: Raspberry Pi 3 Modelo B+, 2 de Julio de 2018. [Último acceso: 28 de Noviembre de 2019.] https://hardzone.es/reviews/perifericos/analisis-raspberry-pi-3-modelo-b/

[9] Omniblug. http://www.omniblug.com/index.html. [En línea] Sensor de temperatura y humedad DHT11 - DHT22, 24 de Julio de 2019. [Último acceso: 23 de Noviembre de 2019.] http://www.omniblug.com/sensor-temperatura-humedad-DHT11-DHT22.html

[10] GeekBot. www.geekbotelectronics.com. [En línea] Temperature and humidity module DHT11 Product Manual [Último acceso: 10 de Noviembre de 2019.] http://www.geekbotelectronics.com/wp-content/uploads/2014/09/DTH11_geekbot.pdf.

[11] Talos Electronics. https://www.taloselectronics.com/. [En línea] Sensor analógico de nivel de agua [Último acceso: 19 de Febrero de 2020.]

https://www.taloselectronics.com/products/sensor-analogico-de-nivel-de-agua#:~:text=Descripci%C3%B3n%3A,directamente%20placa%20de%20desarrollo%20Arduino..

[12] Rambal Automatización y Robótica. https://rambal.com/. [En línea] Sensor Nivel Liquidos Mejorado LQ-SENSOR NON-CONTACT. [Último acceso: 2 de Marzo de 2020.] https://rambal.com/presion-peso-nivel-flex/857-sensor-nivel-de-liquidos-sin-contacto-xkc-y25-v.html.

[13] Amazon. https://www.amazon.es/. [En línea] Sensor de Proximidad Capacitivo. [Último acceso: 8 de marzo de 2020.] https://www.amazon.es/Proximidad-Capacitivo-LJC30A3-H-Z-Interruptor-Normalmente/dp/B0827QTC7S.

[14] VISHAY. www.vishay.com. [En línea] VEML6075, 23 de Noviembre de 2016. [Último acceso: 18 de Febrero de 2020.] https://cdn.sparkfun.com/assets/3/c/3/2/f/veml6075.pdf.

[15] NOMOYPET. https://www.nomoypet.net/. [En línea] Nomo y Pet [Último acceso: 18 de Febrero de 2020.] https://www.nomoypet.net/.

[16] Aliexpress. es.aliexpress.com. [En línea] UVA + UVB lámpara para reptiles [Último acceso: 22 de Febrero de 2020.] https://es.aliexpress.com/item/4000611650091.html?src=google&src=google&albch=shopping&acnt=631-313-3945&isdl=y&slnk=&plac=&mtctp=&albbt=Gploogle_7_shopping&aff_atform=google&aff_short_key=UneMJZVf&&albagn=888888&albcp=9858092633&albag=99420998559&trgt=89.

[17] Meteoblue weather close to you . meteoblue.com. [En línea] Humedad. [Último acceso: 1 de Abril de 2020.] https://content.meteoblue.com/es/especificaciones/variables-meteorologicas/humedad#:~:text=Humedad%20relativa&text=Por%20lo%20tanto%2C%20la%20definici%C3%B3n,g%2F(m%5E3)).

[18] Aliexpress. es.aliexpress.com. [En línea] Mini humidificador USB DIYhttps://es.aliexpress.com/item/32864927823.html?spm=a2g0s.9042311.0.0.3da263c0L7wma4.

[19] Cetronic componentes eléctricos. cetronic.es. https://www.cetronic.es/. [En línea] ALMOHADILLA CALEFACTORA ELECTRICA. [Último acceso: 25 de Febrero de 2020.] https://www.cetronic.es/sqlcommerce/disenos/plantilla1/seccion/producto/DetalleProducto.jsp?idIdioma=&idTienda=93&codProducto=999334129&cPath=1343 http://descargas.cetronic.es/1481.pdf.

[20] Velleman. https://www.velleman.eu/. [En línea] VENTILADOR 12VDC. [Último acceso: 29 de Febrero de 2020. https://www.velleman.eu/products/view?id=380578&country=us&lang=es.

[21] tuaireacondicionado. https://tuaireacondicionado.net/. [En línea] La válvula solenoide [Último acceso: 30 de Mayo de 2020.] https://tuaireacondicionado.net/valvula-solenoide-funcionamiento/.

[22] Amazon. https://www.amazon.es/ [En línea] Válvula Magnética Solenoide Eléctrica Latón Normalmente Cerrada para Agua Control. [Último acceso: 18 de Febrero de 2020.] https://www.amazon.es/V%C3%A1lvula-Magn%C3%A9tica-Solenoide-El%C3%A9ctrica-Normalmente/dp/B07F366L6D/ref=asc_df_B07F366L6D/?tag=googshopes-21&linkCode=df0&hvadid=376199076429&hvpos=&hvnetw=g&hvrand=16142404702574605348&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvc.

[23] Aliexpress. https://es.aliexpress.com/ [En línea] Válvula de bola de latón motorizada eléctrica DN20 [Último acceso: 22 de Febrero de 2020.] https://es.aliexpress.com/item/32839550310.html.

[24] Aliexpress. es.aliexpress.com. [En línea] Módulo de relé de 5 V [Último acceso: 14 de Abril de 2020.] https://es.aliexpress.com/item/32735292917.html?spm=a219c.search0104.3.2.60583ba1tQczjG&ws_ab_test=searchweb0_0,searchweb201602_1_10152_10151_10065_10344_10068_10342_10547_10343_5722611_10340_10548_10341_10696_5722911_5722811_5722711_10084_10083_10618_103.

[25] Aliexpress. es.aliexpress.com. [En línea] Raspberry Pi 3 Modelo B [Último acceso: 13 de Octubre de 2019.] https://es.aliexpress.com/item/32626862737.html?spm=a2g0s.9042311.0.0.409063c0e5lpkf.

[26] Aliexpress. es.aliexpres.com. [En línea] DHT11 Sensor digital de humedad y temperatura para arduino y Raspberry Pi. [Último acceso: 18 de Septiembre de 2019.] https://es.aliexpress.com/item/32874373225.html?spm=a2g0s.9042311.0.0.409063c0e5lpkf.

[27] Amazon. https://www.amazon.es/ [En línea] Sensor de luz UV, VEML6075 UV Detecter. [Último acceso: 20 de Mayo de 2020.] https://www.amazon.es/gp/product/B084KWBNYX/ref=ppx_yo_dt_b_asin_title_o06_s00?ie=UTF8&psc=1.

[28] Aliexpress. https://es.aliexpress.com/. [En línea] XKC-Y25-V. [Último acceso: 8 de Marzo de 2020.] https://es.aliexpress.com/item/4000935529678.html?spm=a2g0o.productlist.0.0.15621a2fnnVObO&algo_pvid=f5454ee2-2c2c-4852-a50c-ca67e617007f&algo_expid=f5454ee2-2c2c-4852-a50c-ca67e617007f-0&btsid=0ab6f82115936756868352603e7788&ws_ab_test=searchweb0_0,search.

[29] CESPEDES Electrónica. https://www.cespedes.es/index.html. [En línea] BLS12/92 VENTILADOR 12VDC [Último acceso: 22 de Marzo de 2020.] https://www.cespedes.es/spa/item/1410585.html.

[30] Universidad d'Alacant. https://www.ua.es/es/. [En línea] Modelo-Vista-Controlador [Último acceso: 12 de Mayo de 2020.] https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-

mvc.html#:~:text=Modelo%20Vista%20Controlador%20(MVC)%20es,control%20en%20tres%20componentes%20distintos.&text=La%20Vista%2C%20o%20interfaz%20de,los%20mecanismos%20interacci%C3.

[31] edX - UPV. https://courses.edx.org/dashboard. [En línea] edX, Android: Introducción a la Programación.. [Último acceso: 15 de Noviembre de 2019.] https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/course/.

[32] Smartdraw. https://www.smartdraw.com/. [En línea] Símbolos de diagramas de flujo [Último acceso: 29 de Junio de 2020.] https://www.smartdraw.com/flowchart/simbolos-de-diagramas-de-flujo.htm.

[33] Wikipedia. https://es.wikipedia.org/. [En línea] Android Studio[Último acceso: 12 de Junio de 2020.] https://es.wikipedia.org/wiki/Android_Studio.

[34] Ubunlog. https://ubunlog.com/. [En línea] Eclipse Oxygen[Último acceso: 12 de Junio de 2020.] https://ubunlog.com/eclipse-oxygen-ide-ubuntu/.

[35] Programo ergo sum. https://www.programoergosum.com/. [En línea] ¿Qué es Raspbian? [Último acceso: 12 de Junio de 2020.] https://www.programoergosum.com/cursos-online/raspberry-pi/232-curso-de-introduccion-a-raspberry-pi/instalar-raspbian.

[36] The Pi4J Project. https://pi4j.com/1.2/. [En línea] The Pi4J Project, 5 de Marzo de 2019. [Último acceso: 13 de Diciembre de 2019. https://pi4j.com/1.2/install.html.

[37] PiMyLifeUp. https://pimylifeup.com/. [En línea] Raspberry Pi UV Sensor using the VEML6075, 3 de Junio de 2019. [Último acceso: 25 de Mayo de 2020.] https://pimylifeup.com/raspberry-pi-uv-sensor-veml6075/.

[38] ThePi4J Project. https://pi4j.com/1.2/. [En línea] Pin Numbering - Raspberry Pi 3B+ [Último acceso: 26 de Junio de 2020.] https://pi4j.com/1.2/pins/model-3b-plus-rev1.html.

[39] Aprendiendo arduino. https://aprendiendoarduino.wordpress.com/. [En línea] I2C, 9 de Julio de 2017. [Último acceso: 8 de Junio de 2020.] https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/.

[40] Raspberry Pi Forum. https://www.raspberrypi.org/. [En línea] Raspberry Pi. [Último acceso: 26 de Octubre de 2019. https://www.raspberrypi.org/downloads/.

[41] balenaEtcher. https://www.balena.io/etcher/. [En línea] Balena Etcher[Último acceso: 26 de Octubre de 2019.] https://www.balena.io/etcher/.