UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Monitorización remota de plantas mediante Raspberry Pi y Telegram

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Santiago Baidez Ayuste

Tutor: Carlos Miguel Tavares de Araujo Cesariny Calafate

Curso 2017-2018

Resum

`Telegram` és una aplicació de missatgeria instantània mòbil i d'escriptori basada en el núvol. Entre altres, `Telegram` permet intercanviar missatges i arxius de qualsevol tipus. En 2017 el nombre d'usuaris de `Telegram` supera els 100 milions d'usuaris actius i els seus servidors processen més de 15 mil milions de missatges al dia.

Una de les principals característiques de `Telegram` és la seua plataforma per a bots. Els usuaris poden usar l'API que oferix `Telegram` per a crear els seus propis bots i posar-los en funcionament, sent accessibles a tots els usuaris de l'aplicació sense cap cost per al desenvolupador o l'usuari. Aquests bots permeten dotar de noves funcionalitats a l'aplicació, com ara jocs, serveis d'informació o utilitats fent que `Telegram` siga molt més que una aplicació de missatgeria.

D'altra banda, tenim `Raspberry Pi`, un mini-computador basat en ARM amb infinitat de possibilitats. Els projectes que s'aborden amb `Raspberry Pi` van des del seu ús com a computadors funcionals de baix cost fins a clusters de processament de dades, passant per aplicacions domòtiques, recreatives i científiques de qualsevol tipus.

Gràcies a aquestes dues tecnologies, es poden dur a terme projectes molt variats i diversos en els que l'únic límit és la nostra imaginació. I amb aquesta premissa i tenint en compte les necessitats i inquietuds actuals de la societat de controlar i informatitzar tot el que ens rodeja, s'obri una porta per a monitoritzar elements a què ens agradaria prestar-los més atenció diàriament, però que els horaris actuals ens impedixen realitzar.

Ací doncs, s'oferix una solució per a monitoritzar un element tan quotidià com una planta, podent regar-la en la distància o automatitzar aquesta tasca. Aquesta solució ens oferix la possibilitat d'interactuar amb la nostra planta i comprovar que està en bon estat, facilitant-nos la tasca de l'atenció i de manteniment del nostre amic vegetal.

Paraules clau: Raspberry pi, Telegram, bot, monitorització remota

Resumen

Telegram es una aplicación de mensajería instantánea móvil y de escritorio basada en la nube. Entre otros, Telegram permite intercambiar mensajes y archivos de todo tipo. En 2017 el número de usuarios de Telegram superó los 100 millones de usuarios activos, y sus servidores procesaron más de 15 mil millones de mensajes al día.

Una de las principales características de Telegram es su plataforma para bots. Los usuarios pueden usar la API que ofrece para crear sus propios bots y ponerlos en funcionamiento, siendo accesibles a todos los usuarios de la aplicación sin coste alguno para el desarrollador o para el usuario. Estos bots permiten dotar de nuevas funcionalidades a la aplicación, tales como juegos, servicios de información o utilidades, haciendo que Telegram sea mucho más que una aplicación de mensajería.

Por otro lado, podemos encontrar en el mercado la Raspberry Pi, un mini-computador basado en ARM con infinidad de posibilidades. Los proyectos que se abordan con Raspberry Pi van desde su uso como computadores funcionales de bajo coste hasta clústeres de procesamiento de datos, pasando por aplicaciones domóticas, recreativas y científicas de todo tipo.

Gracias a estas dos tecnologías, se pueden llevar a cabo proyectos muy variados y diversos en los que el único límite es nuestra imaginación. Con esta premisa, y teniendo en cuenta las necesidades e inquietudes de la sociedad actual en el sentido de controlar e informatizar todo lo que nos rodea, se abre una puerta para monitorizar elementos a los que nos gustaría prestarles más atención a diario, pero que los horarios actuales nos impiden realizar.

Aquí pues, se ofrece una solución para monitorizar un elemento tan cotidiano como una planta, pudiendo regarla desde la distancia o automatizar esta tarea. Esta solución nos ofrece la posibilidad de interactuar con nuestra planta y comprobar que está en buen estado, facilitándonos la tarea del cuidado y de mantenimiento de nuestro amigo vegetal.

Palabras clave: Raspberry Pi, Telegram, bot, monitorización remota

Abstract

Telegram is a mobile and desktop instant messaging application that relies on the cloud computing paradigm. Among others, Telegram allows the user to exchange messages and files of any kind. In 2017, the number of Telegram users exceeded 100 million, and their servers process more than 15 billion messages per day.

One of the main features of Telegram is its bot's API. Users can use the bot's API of Telegram to create their own bots and deploy them, being accesible to all the users of the applicaction, and is free of charge to both developers and users. These bots extend the functionalities of the application, by offering games, information services or other utilities, making Telegram more than a simple messaging application.

On the other hand, in the market we can find the Raspberry Pi, a small ARM-based computer, with a large list of possibilites. Projects that rely on the Raspberry Pi include use as a low cost computer to big clusters of data processing, and also home automation applications, entertainment systems and scientific systems of all kinds.

Thanks to these two technologies, it is possible to carry out very different projects in where our imagination is the limit. With that premise, and noticing the current needs and concerns for controlling and computerizing all that surround us, a door is open to monitoring some elements that we would like to take care more often, but that we do not have enough time to do so.

In this project, a solution is offered to monitorize an element as current as a plant, making it possible to water it from the distance, or to automate that task. This solution offers us the possibility of interacting with our plant, and to check whether it is in a good condition, simplifying the task of taking care and mantaining our vegetal friend.

Key words: Raspberry pi, Telegram, bot, remote monitoring

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

1.1 Motivación

A diario vemos como el "Internet de las cosas", o IoT (por sus siglas en inglés, "Internet of Things"), cobra cada vez más importancia en nuestra vida diaria. De hecho, tendemos a tener todo más informatizado y conectado a Internet.

La primera inquietud de los usuarios es dotar de seguridad al hogar, y los sistemas de videovigilancia vinieron para cubrir esa necesidad, últimamente con el aliciente de poder visualizar el hogar desde el propio terminal móvil.

Una vez se cubrieron las necesidades de seguridad, se buscaron otras formas de poder hacer nuestra vida más confortable. Para ello se empezó por conectar a la red nuestros electrodomésticos más importantes. Si hablamos de confortabilidad, lo más esencial es la temperatura. Encontrar nuestro hogar a la temperatura ideal cuando llegamos a casa es uno de los lujos que se hacen posibles mediante el Internet de las Cosas.

De nuevo, cuando tenemos la seguridad y confortabilidad cubiertas, buscamos el control de otros electrodomésticos que nos hagan la vida más fácil. Un claro ejemplo es la nevera. No parece muy útil que una nevera tenga la capacidad de conectarse a Internet, pero si pensamos en que la nevera puede llegar a encargarse de realizar la compra por nosotros, este aspecto cobra otra perspectiva.

Se podría decir con casi total seguridad que, algún día, todos los electrodomésticos que tenemos en casa estarán conectados a Internet ya que, de un modo u otro, eso nos hará la vida más fácil.

Partiendo de esta base, se planteó que el ritmo ajetreado de nuestra vida actual puede llegar a hacernos descuidar nuestras plantas, nuestro pequeño jardín, o quizás nuestro huerto urbano. Por eso, lo que se viene a ofrecer aquí es una solución que, usando tecnologías actuales, es capaz de cubrir esa necesidad. Con esta solución podremos controlar de manera remota nuestras plantas sin dedicarles mucha atención, y convirtiendo la tarea del cuidado de una planta en algo divertido.

Atendiendo a que "Sistema de monitorización remota de plantas mediante `Raspberry Pi` y `Telegram`" es un nombre largo a la par que difícil de recordar, a lo largo de esta memoria se hará referencia al sistema como **Cactus Pi**. Se decidió usar este nombre porque

las primeras pruebas que se realizaron con este sistema, se hicieron con un cactus, que no requería demasiados cuidados. Además, ofrece una analogía relacionada con las espinas de los cactus y los pines GPIO de `Raspberry Pi`.

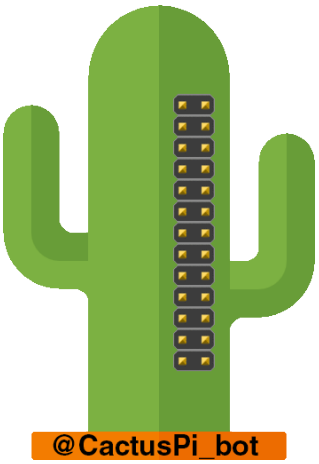

Figura 1.1: Logotipo escogido para Proyecto Cactus Pi. (Photo credit: Iconexperience.com)

Para la realización del logotipo, se empleó una imagen encontrada en Internet, la cual se modificó para el propósito del proyecto.

Se usará este logotipo como imagen del *bot* de `Telegram`, y en cualquier documento relacionado con el proyecto.

Como se trata de un proyecto en el que se trabaja con diferentes tecnologías, al final de este documento se adjunta un glosario de términos técnicos o específicos.

1.2 Alcance

El proyecto que aquí se desarrolla, busca cubrir la monitorización remota de una planta, así como la instalación de actuadores que puedan interactuar con la misma. Todo esto controlado desde un *bot* desarrollado para la plataforma `Telegram Messenger`.

Por lo tanto, el principal objetivo es desarrollar un *bot* que sea capaz de comunicarse con un servidor, en nuestro caso una placa `Raspberry Pi`, y que sea capaz de recibir información de sensores conectados a la placa y enviar órdenes a los actuadores que interactuarán con la planta supervisada.

Asimismo, se desarrollará un sistema de gestión de usuarios para permitir al administrador del *bot* que solo hagan uso del mismo aquellos usuarios legítimos y con los permisos que se les hayan concedido. Por lo tanto, será necesario desarrollar esta capa de seguridad.

Para desarrollar el proyecto **Cactus Pi**, será necesario dominar diferentes tecnologías entre las que, inicialmente, se encuentran:

- Telegram Messenger: habrá que estudiar cómo se realizan *bots* en esta plataforma, y averiguar cómo se ponen en funcionamiento y a disposición de los usuarios.
- `Raspberry Pi`: será necesario familiarizarse con el entorno `Raspberry Pi` para poder llegar a usar esta placa para los fines que se han descrito.

1.3 Estructura de la memoria

A continuación, siguen seis capítulos cuyo contenido es el siguiente:

- **Capítulo 2: Estado del arte.** En este capítulo se indagará en otras soluciones a problemas similares al propuesto, se verá cómo los han enfrentado y cómo se han resuelto, señalando sus características, y viendo de qué tecnologías hacen uso.
- **Capítulo 3: Análisis.** Una vez estudiada la situación actual desde la que se parte, se evaluará cómo se puede afrontar el problema que se ha definido, y se presentará una solución para llegar al objetivo propuesto. Para establecer una métrica que indique si se alcanza este objetivo, se analizarán los requisitos que tendrá que cumplir el sistema.
- **Capítulo 4: Tecnologías.** Aquí se expondrán las tecnologías que se emplearán para desarrollar el proyecto que van a permitir cubrir los requisitos definidos.
- **Capítulo 5: Planificación.** En este capítulo se establecerá la planificación que se seguirá para desarrollar el proyecto.
- **Capítulo 6: Desarrollo.** En este capítulo se aborda cómo se implementa la solución propuesta, teniendo en cuenta los requisitos definidos, las tecnologías recogidas para su desarrollo, y la planificación establecida.
- **Capítulo 7: Conclusiones.** Este capítulo presentará una conclusión del trabajo realizado, servirá para analizar qué ha ido bien, qué se podría mejorar, y presentará las posibles líneas de trabajo que pudieran derivarse del aprendizaje adquirido.

CAPÍTULO 2
Estado del arte

Antes de comenzar con el desarrollo del *bot*, conviene pararse a estudiar otras soluciones que estén cubriendo las mismas necesidades que se pretenden cubrir. Aquí tienen cabida tanto sistemas artesanales, como sistemas profesionales desarrollados por empresas que los comercializan.

Para comenzar, se han buscado proyectos similares en Internet. Se ha podido encontrar alguno basado en el cuidado automatizado de plantas, sin tener en cuenta la monitorización remota. En estos casos, lo que prima es automatizar procesos sin la acción del usuario.

Como ejemplo, se encuentra este proyecto de riego automatizado de tomates en un macetero casero con publicación en `Twitter` cada vez que se riega. El proyecto `Tomaatit` [1] es de 2009, y está realizado empleando una placa `Arduino`. El riego está programado con una cierta frecuencia, y tiene conectados sensores de temperatura, humedad y presión atmosférica. Todo esto se publica en dicha red social y, aunque es un proyecto ya sin uso, todavía se pueden consultar las publicaciones que realizaron en su momento.

Figura 2.1: Proyecto Tomaatit.

Se puede apreciar que se trata de un sistema muy artesanal, donde prima la funcionalidad. El usuario no puede controlar nada más allá de lo que está programado en la placa `Arduino`. Cada modificación se tiene que hacer conectando la placa al ordenador. Por lo

tanto, no es un proyecto que permita a todo tipo de usuarios trabajar con él. Podemos decir que es un proyecto para usuarios avanzados o expertos en el uso de tecnologías.

También vemos que el usuario no puede conocer el estado de sus plantas de manera remota, si están a una temperatura alta, o con baja humedad, salvo cuando la placa riega y lo publica en `Twitter`. Esto no sería el escenario ideal para un usuario que necesita saber cómo está su planta o su invernadero en un momento puntual.

Ahora, vamos a ver un sistema de cuidado de plantas comercial aunque no profesional. Se trata del sistema `AeroGarden` [2], un sistema de cuidado de plantas hidropónico, es decir, sin tierra, donde los nutrientes de las plantas se encuentran en el agua.

Figura 2.2: Sistema de riego hidropónico `AeroGarden`.

Este sistema llama la atención porque está monitorizando la planta e indica al usuario cuándo tiene que añadir los nutrientes, ofrece consejos del cuidado, etc. También vemos que incluye una lámpara de luces LED en la parte superior, que garantizan la iluminación necesaria para la planta con un bajo consumo eléctrico.

Básicamente, se trata de un sistema donde los usuarios pueden cuidar de sus plantas favoritas y, con ayuda del panel informativo, conseguir que crezcan con garantías.

Si bien no se trata de un sistema como el que se desea desarrollar, aporta un enfoque interesante, el de ofrecer un panel informativo que el usuario pueda visualizar al llegar a casa y donde pueda ver si tiene que realizar alguna acción con la planta.

No habilita ningún tipo de actuación externa pero, como tampoco necesita de riego, ya que se trata de un sistema hidropónico, es razonable para el uso al que está enfocado.

En su web, vemos que han desarrollado una aplicación móvil para visualizar el mismo panel informativo en el terminal mientras se está conectado a la misma red.

Parece importante y necesario, antes de lanzarse a desarrollar un producto nuevo, ver qué tipo de soluciones aportan otros, para así ofrecer algo que se ajuste mejor a un potencial mercado y diferenciarse de otros con propuestas nuevas y no explotadas todavía.

Ahora, se van a analizar algunos sistemas desarrollados con las mismas tecnologías que se emplearán en este proyecto, para ver cómo las han aprovechado otros en otras circunstancias, y aprender de estos sistemas para enfocar el desarrollo del proyecto actual.

2.1 Raspberry Pi

2.1.1. Introducción a `Raspberry Pi`

En el capítulo de tecnologías se definirá qué es `Raspberry Pi`, y por qué ha alcanzado tanta popularidad en la actualidad. A modo de resumen, indicar que se trata de una placa de bajo coste que permite abordar proyectos de todo tipo a un coste muy reducido, ofreciendo una eficiencia muy alta atendiendo a su consumo de energía.

A continuación, veremos algunos proyectos para los que se suele destinar la `Raspberry Pi` en busca de ideas que se pueden aprovechar para este proyecto. Se busca en la red [3] y, entre otros, se encuentran los que se detallan a continuación.

2.1.2. Proyectos con `Raspberry Pi`

`Raspberry Pi` como servidor web

Uno de los usos más extendidos que se encuentran en la red, es el de emplear una `Raspberry Pi` como un servidor web. Es lógico, dado que `Raspberry Pi` tiene suficiente potencia para cumplir este cometido, y su bajo consumo eléctrico permite mantener la placa encendida 24x7 con un consumo similar al de un teléfono móvil.

En este caso, la placa se puede emplear a modo de servidor de pruebas, para un proyecto en desarrollo que el usuario quiere tener accesible desde cualquier punto, o incluso para poner en el proyecto en producción, si se trata de un proyecto pequeño que no va a ser accedido masivamente, ya que el ancho de banda de la conexión del usuario sería un cuello de botella, y la placa `Raspberry Pi` posiblemente no podría responder a muchas solicitudes.

`Raspberry Pi` en domótica

Otro de los usos que más popularidad está adquiriendo, es el empleo de la placa para proyectos de domótica. Y es que `Raspberry Pi` es perfecta para todo tipo de proyectos domóticos, ya que permite programar, en pocas líneas de código, aplicaciones que nos ayuden a controlar nuestro hogar.

Como vemos en la figura anterior, las posibilidades son inmensas. Desde un simple monitor de temperatura hasta un sistema de alertas de correo en nuestro buzón. Sin olvidarnos de los aspectos más importantes, como detector de incendios o sistemas de seguridad, o incluso fugas de agua. También podemos conectar detectores de presencia e

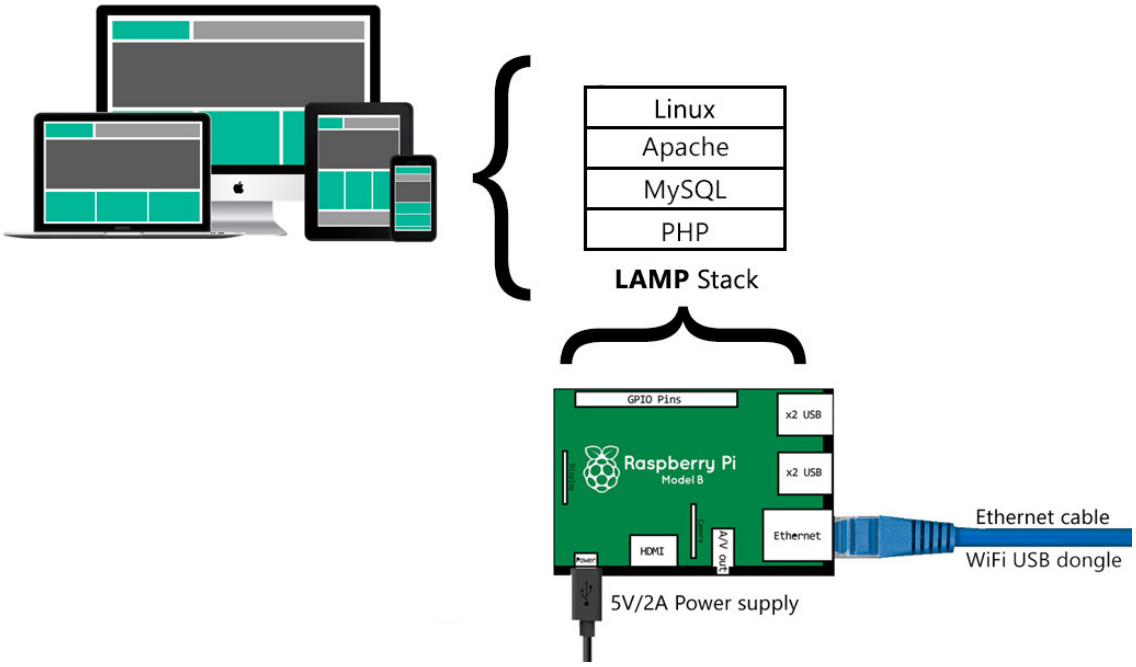

Figura 2.3: `Raspberry Pi` como servidor web

incluso un notificador de ladridos de perro.

Lo más sorprendente es que la `Raspberry Pi` es capaz de gestionar todo esto con un coste muy reducido en comparación con sistemas domóticos convencionales.

`Raspberry Pi` en sistemas de seguridad

Finalmente, un uso que podríamos englobar en la categoría anterior, porque cada vez se asocian más, pero no por ello tienen que ir de la mano. Se trata de sistemas de videovigilancia basados en `Raspberry Pi`. Puesto que `Raspberry Pi` ofrece la opción de conectar una cámara y discos duros por USB, se han desarrollado aplicaciones que permiten emplear la `Raspberry Pi` como un sistema de videovigilancia.

Estos sistemas permiten, entre otros, grabar vídeo cuando se activa un sensor de luz o de presencia, alertar al usuario cuando se activa un detector, gestionar las grabaciones, configurar las horas de grabación, ofrece acceso remoto vía web para consultar las grabaciones o ver el vídeo en tiempo real, y un sinfín de opciones que hacen de la `Raspberry Pi` un perfecto gestor de la seguridad del hogar o de un negocio, aunque muchos también lo usan para ver a sus mascotas, niños pequeños, personas mayores o enfermas, etc.

2.2 Telegram Messenger

2.2.1. Introducción a Telegram Messenger

Actualmente, `Telegram` es el tercer sistema de mensajería instantánea más utilizado en 2018, después de Whatsapp y Facebook Messenger. `Telegram` es software libre en su

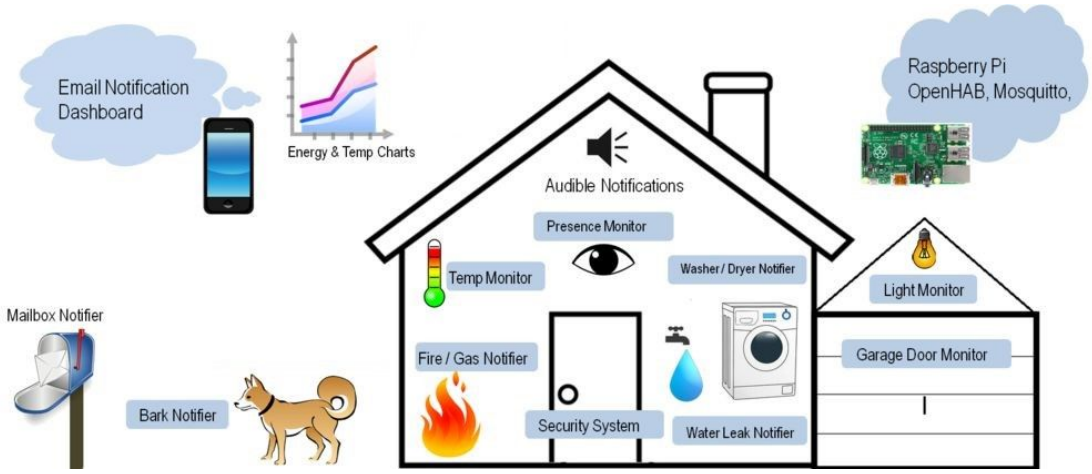

Figura 2.4: Usos de `Raspberry Pi` en domótica

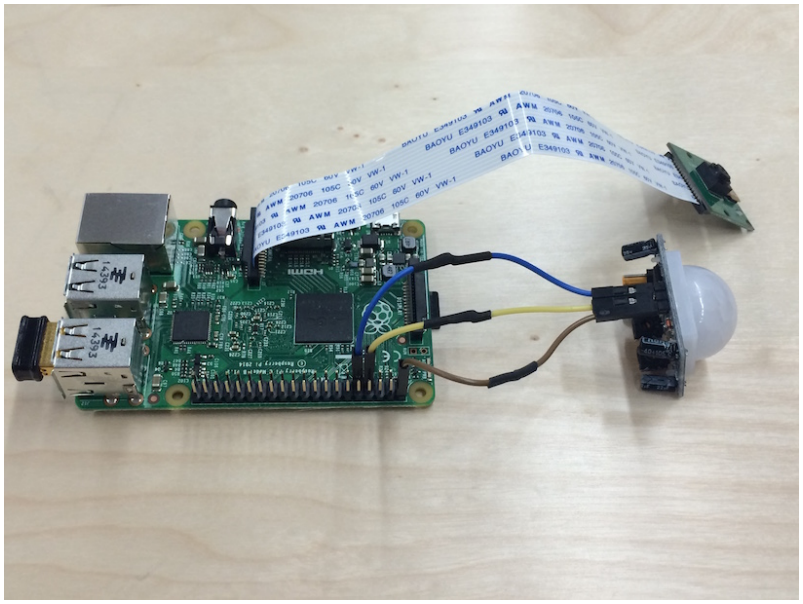

Figura 2.5: `Raspberry Pi` con cámara y sensor de presencia conectados.

parte de cliente (frontend) y además es multiplataforma, teniendo clientes para los sistemas más importantes.

Más adelante se explicará de dónde viene `Telegram`, y qué permite hacer a los desarrolladores de software, pero antes de continuar hay que indicar que `Telegram` ha habilitado una *API* para la creación de *bot*s en su plataforma, permitiendo que pueda utilizarse para una infinidad de objetivos y proyectos.

Existen *bot*s para todo tipo de necesidades, como juegos, alarmas, descarga de música, creación de encuestas, productividad, moderación de grupos, etc.

A continuación, veremos algunos *bot*s relevantes para el proyecto que nos ocupa.

2.2.2. Bots de Telegram

TeleMonBot

TeleMonBot [4] es un script de código abierto, desarrollado por `Egor Koshmin`, que permite monitorizar sistemas Windows y Linux recibiendo la información en un *bot* de `Telegram`.

Para ello, el usuario tiene que crear su propio *bot* y ponerlo a funcionar con el código que está en el repositorio de Koshmin.

Este *bot*, habilita al usuario a recibir la siguiente información sobre su servidor:

- El porcentaje de memoria RAM utilizada.
- El porcentaje de CPU en uso.
- La disponibilidad del sistema.
- La hora del servidor.
- Tomar y enviar una captura de pantalla del servidor.
- Consultar la ubicación geográfica del servidor.

Y muchas más opciones que va ampliando la comunidad a diario.

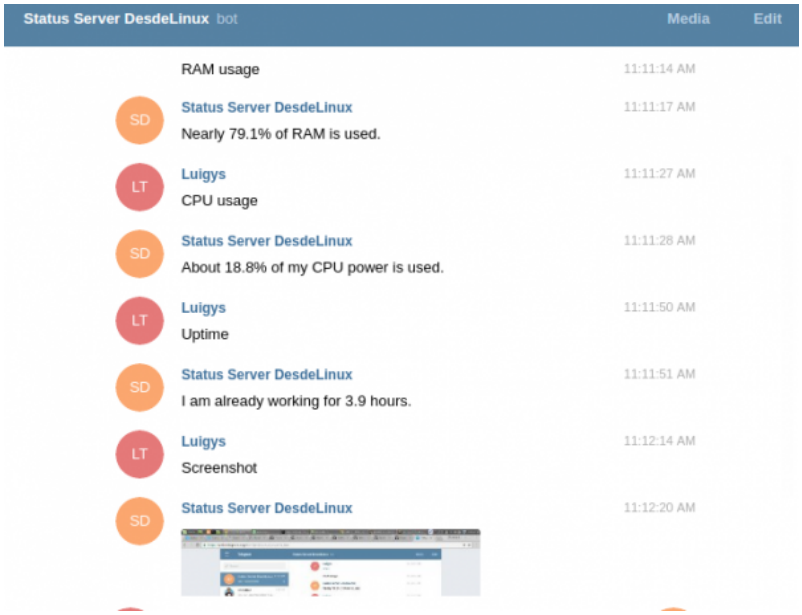

Figura 2.6: Telegram como sistema de monitorización de servidores.

En este caso, el propio servidor aloja el *bot* de `Telegram` y utiliza las métricas del sistema para comunicarlas al *bot*, y éste al usuario. Por lo tanto, no necesita ningún tipo de sensor añadido.

Downtime Bot

En la línea de este *bot*, se encuentran otros casos como `DowntimeBot` [5], un servicio de pago, que, mediante un *bot* de `Telegram`, permite activar alertas de caídas de los sitios web de los que el usuario es propietario. El servicio todavía está en fase de desarrollo, pero ya ofrece funcionalidades tales como:

- Monitorizar tantas URL como el usuario desee.
- Monitorización cada 3 segundos.
- Alertas ante caídas, sin importar si son cortas.
- Sin necesidad de instalar aplicaciones de terceros, sólo `Telegram`.

Por lo tanto, encontramos otra utilidad para la que `Telegram` es verdaderamente útil. Los usuarios buscan conocer de inmediato si su web está caída para reaccionar cuanto antes, y este *bot* consigue ese cometido a muy bajo coste gracias a `Telegram`.

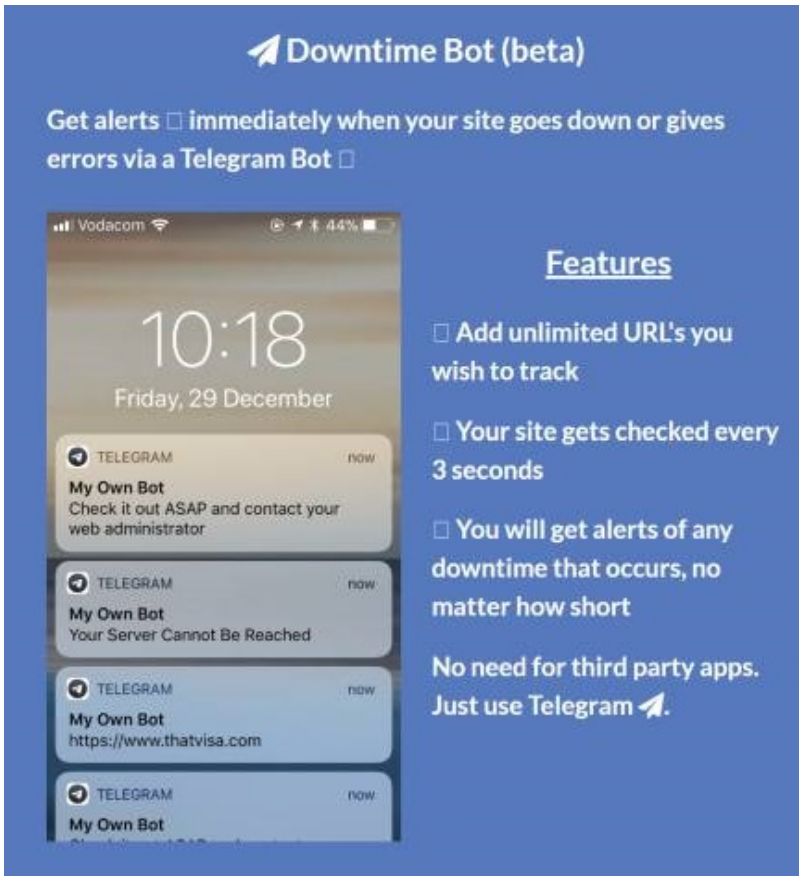

Figura 2.7: DownTime Bot. *Bot* de alertas ante caídas en sitios web.

Bot de consulta de condiciones para el surf

Como curiosidad, este último *bot* muestra un interesante uso de `Telegram` para cubrir una necesidad: conocer las condiciones locales para practicar surf. El desarrollador de `BotSurf` [7] quería poder consultar de una manera fácil las condiciones climatológicas para poder practicar su deporte favorito y, para ello, nada mejor que usar un *bot* de

`Telegram` y una *API* pública donde consultar la meteorología.

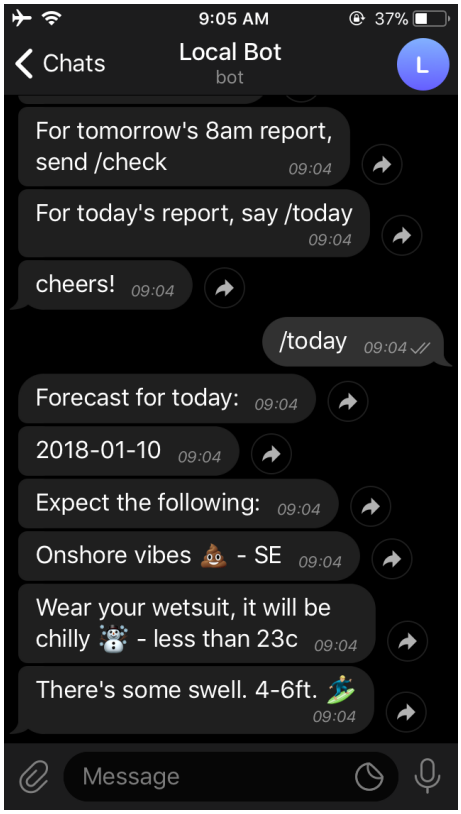

Figura 2.8: *Bot* de `Telegram` para consultar las condiciones climatológicas locales.

El desarrollador necesitaba conocer rápidamente los datos relativos a la dirección del viento, el oleaje y el tamaño de las olas. Usando la *API*, el *bot* puede realizar la consulta que solicita el usuario. En la figura 2.8 vemos un ejemplo. El usuario envía el comando "/today" y el *bot* realiza la consulta a la *API* externa, recibe los datos, extrae los que necesita el usuario y se los devuelve en forma de mensaje.

En este ejemplo, vemos cómo se hace uso de un *bot* para recibir la información que necesita el usuario. Esto es lo que se quiere conseguir con el proyecto **Cactus Pi**. En este caso no se contará con una *API* a la que consultar, sino que se tendrá que obtener los datos de sensores que estarán conectados a una `Raspberry Pi`.

2.3 Sistemas domóticos

2.3.1. Introducción

Para finalizar, se muestran a continuación algunas de las ventajas que aporta la domótica, y cómo lo hace para facilitarnos la vida mediante el uso de la tecnología.

Como sabemos, la domótica [8] está cada vez más integrada en los hogares, y es seguro que esta tendencia irá aumentando.

Algunos ejemplos de las funciones que ofrecen los sistemas de domótica actuales para mejorar la confortabilidad y seguridad del hogar son:

- Regulación de la temperatura en cada estancia del hogar
- Control y automatización de la iluminación, incluso generando diferentes ambientes en función de la intensidad de la luz y el color.
- Gestión de alarmas y control de entrada de intrusos.
- Subida y bajada de persianas o plegado y desplegado de toldos.
- Control de sistemas multimedia e hilo musical.
- Gestión de la temperatura mediante aire acondicionado o calefacción.
- Gestión de horarios para diferentes usos: cargar un coche eléctrico, subir o bajar el ascensor a una planta determinada, apagado de luces programado, etc.

2.3.2. Sistemas de monitorización

Cuando se habla de domótica, una de las razones más interesantes para los usuarios es monitorizar el consumo eléctrico, a fin de poder reducirlo y obtener un ahorro energético.

Existen en el mercado multitud de equipos de gestión y control inteligente de la energía para permitir a los usuarios reducir su consumo mediante la monitorización cuando no se está en el hogar, programar electrodomésticos para su uso en franjas de energía más económica y de este modo incrementar el confort, así como el valor final de la vivienda.

Smappee

Uno de estos monitores comercializados es *Smappee* [9], un monitor inteligente que reconoce el consumo eléctrico de cada uno de los electrodomésticos del hogar.

El producto ofrece al usuario conocer el consumo de energía en tiempo real, recibir alertas en caso de detectar fugas o peligros, conocer qué ocurre en el hogar en cada momento, y un ahorro energético de hasta un 30 %.

Figura 2.9: Presentación del monitor energético Smappee.

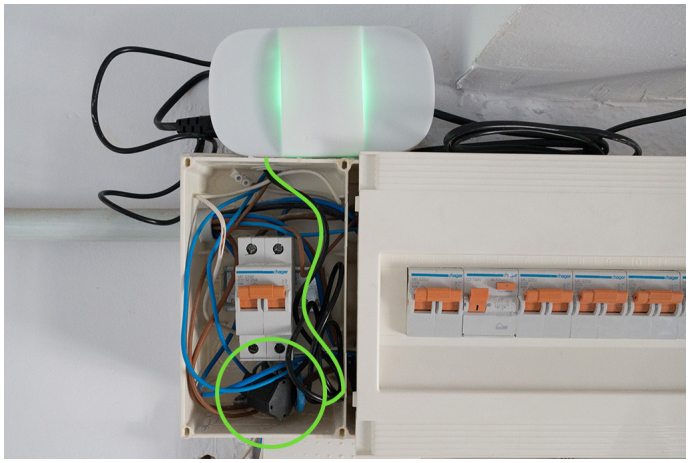

Figura 2.10: Monitor energético Smappee en instalación eléctrica.

Este dispositivo utiliza el protocolo MQTT, cuyas siglas pertenecen a "Message Queue Telemetry Transport", protocolo ideado por `IBM` y liberado para que cualquiera pueda usarlo enfocado a la conectividad "Machine-to-Machine" (M2M).

`Smappee` permite, además, la monitorización remota mediante aplicaciones nativas para los sistemas Android e iOS.

Broadlink A1

Sin ir a dispositivos tan complejos, encontramos otras opciones más sencillas que también pueden aportar a los hogares confortabilidad y bienestar.

A modo de ejemplo destacar el dispositivo Broadlink A1 [10], un monitor de calidad del aire, temperatura, humedad, luminosidad y ruido. En muy poco espacio es capaz de agrupar todas estas funcionalidades y con una interfaz muy cuidada. Por ejemplo, para visualizar la calidad del aire, el usuario solo tiene que fijarse en el color de la luz que transmite.

Figura 2.11: Monitor de calidad del aire, temperatura, humedad, luminosidad y ruido.

Para la comunicación con el usuario, también ponen a disposición del mismo una aplicación que, una vez registrado el dispositivo, permite consultar todos los parámetros que recogen sus sensores.

Es muy importante tener en cuenta este tipo de ejemplos al desarrollar un *bot* y ver cómo lo están haciendo empresas que se dedican al sector del *IoT*.

2.3.3. Sistemas de actuación

Centro de control de domótica

Para finalizar, se muestra un centro de control domótico convencional [11]. Este tipo de centros permiten agrupar desde un único punto todo el control de los sistemas domóticos instalados en el hogar.

Como vemos, desde un único punto, se podría establecer comunicación con cualquier actuador que estuviera conectado a la instalación y ponerlo en funcionamiento. Por contra, la empresa no ofrece una aplicación móvil que permita a los usuarios realizar esas mismas acciones desde sus terminales móviles. No se puede decir que sea un fallo, pero es una funcionalidad cada vez más solicitada.

2.4 Crítica al estado del arte

Como se ha podido comprobar, este proyecto se va a sustentar en ideas existentes, pero que aun no han sido empleadas para el mismo propósito.

Existen en el mercado distintos productos para cuidar de nuestras plantas, se utilizan tanto `Raspberry Pi` como `Telegram` para infinidad de proyectos y, sin embargo, no se ha encontrado un proyecto en que haga uso de ambas tecnologías con el objetivo de éste.

Figura 2.12: Centro de control domótico.

Con este proyecto, se pretende cubrir ese espacio, el de usar dos tecnologías aparentemente independientes para un fin único. En este caso, el cuidado de una planta.

Es lógico pensar que este objetivo no dista mucho de otros, como podrían ser la monitorización de animales domésticos, como perros o gatos, monitorizando los niveles de alimento y agua a disposición del mismo, y su visualización mediante cámara web. O animales de pecera, midiendo los niveles de pH del agua, y dispensando alimento de manera remota.

Por lo tanto, las posibilidades que se abren son inmensas, y muchas de ellas están aun por explotar.

2.5 Propuesta

Según lo que se ha visto en este capítulo, resta ver qué ideas se van a tomar de lo aprendido, y qué se propone mejorar de las tecnologías existentes.

Por un lado, se ha visto el riego de las plantas, usando para su control las medidas de temperatura y humedad; esto será básico para el proyecto.

Del sistema hidropónico se va a tomar la idea de la iluminación LED para las plantas. Es interesante poder iluminar las plantas, y esto también será de gran utilidad si se pretende capturar una instantánea en la oscuridad. También es interesante la existencia de un panel LCD que muestre información al usuario cuando se encuentra frente a la planta.

Se estudiará si tiene viabilidad y una utilidad práctica para el proyecto.

De los proyectos para los que se suele emplear `Raspberry Pi`, se puede ver que su desempeño como servidor es ideal, consiguiendo una eficiencia muy alta para un consumo eléctrico muy bajo.

Además, el hecho de permitir la conexión de distintos sensores va a permitir no tener que contar con distintos microprocesadores para cada uno de los sensores que se quieran emplear en el proyecto.

De su uso en sistemas de videovigilancia se adquiere la idea de conectarle una cámara web para poder visualizar la planta a monitorizar. Se deberá de estudiar cómo hacerlo.

De `Telegram Messenger`, en adelante simplemente `Telegram`, se pone énfasis en el cómo se crean los *bots*, más que en el qué hacen los mismos. Habrá que ver cómo utilizar el API que ofrece `Telegram` para su creación e implementación.

Se ha visto que es muy simple usar comandos con un *bot*, y que este responda a la petición del usuario. Así es como se pretende proceder.

Finalmente, de los sistemas domóticos, se intenta extraer ideas de elementos que se pueden conectar que ayuden a monitorizar y actuar con una planta. Se ven, por ejemplo, sensores de temperatura, humedad, gestión de la temperatura y luminosidad.

En el siguiente capítulo se realizará un análisis de lo que se ha extraído en esta modesta investigación para definir qué se puede hacer con las tecnologías a nuestro alcance.

CAPÍTULO 3
Análisis

3.1 Identificación del problema

Antes de planificar el trabajo a realizar, se van a evaluar las opciones que se presentan para afrontar el proyecto.

En cuanto al uso de `Raspberry Pi`, tenemos diferentes opciones. Por un lado, se puede tratar de conectar los sensores directamente a la placa `Raspberry Pi` con el riesgo de no encontrar una librería para un sensor en concreto. También se podría conectar la `Raspberry Pi` a una placa `Arduino`, que generalmente tienen mayor compatibilidad con todo tipo de sensores.

La primera solución parece más interesante al evitar añadir una capa más de comunicación que requeriría de un trabajo adicional; por otro lado, si hay que conectar muchos dispositivos a la `Raspberry Pi`, se corre el riesgo de quedar sin alimentación suficiente para todos, o incluso quedarse sin espacios disponibles en los pines GPIO de la placa.

Por otro lado tenemos `Telegram`, aplicación que nos permite realizar *bot*s en prácticamente cualquier lenguaje de programación, dada la naturaleza abierta de su API. Aquí el reto será encontrar el lenguaje que más convenga para llevar a cabo el proyecto.

También se puede especular acerca de la arquitectura a implantar. Por un lado se puede usar un *bot* externo en un servidor web que se comunique mediante un servicio web con la propia `Raspberry Pi`, que tendrá conectados los sensores.

Otra posible solución, sería tener el *bot* funcionando en la propia `Raspberry Pi`, evitando tener que programar la capa del servicio web.

En el primer caso, lo más lógico sería usar un lenguaje como PHP, más propio de servidores web, y las consultas con la `Raspberry Pi` se podrían hacer compartiendo ficheros JSON.

En el segundo caso, `Raspberry Pi` nos ofrece alta compatibilidad con `Python`, lenguaje muy versátil que podemos emplear para programar un *bot* directamente.

3.2 Solución propuesta

Vistas las opciones disponibles, se decide emplear la estrategia más sencilla en términos de implementación, dejando abierta la puerta a posibles modificaciones y ampliaciones si se viera necesario.

Por lo tanto, en la parte hardware, se usarán los sensores directamente conectados a la placa `Raspberry Pi` mediante los pines GPIO y conexiones USB, ya que se estima que no se ocuparán todos y, si fuera necesario, se añadiría una placa `Arduino` para las necesidades adicionales.

En el lado de `Telegram`, se contará con el *bot* alojado en la propia `Raspberry Pi`, ya que facilitará la comunicación directa con el sistema, los sensores y demás componentes del proyecto. Debido a esto, el lenguaje de programación escogido para la realización del *bot*, será `Python`.

3.3 Análisis de requisitos

Para determinar si se ha alcanzado con éxito el objetivo que se pretende abordar, se van a analizar los requisitos que tendrá que cubrir el *bot*.

Antes de realizar un análisis más exhaustivo, los objetivos generales a cubrir con el *bot* serán los siguientes:

- Comprobar la temperatura de la planta.
- Poder ventilar la planta.
- Comprobar la humedad ambiental
- Poder regar la planta.
- Controlar el acceso al *bot* así como los permisos que se asignan a los distintos usuarios.
- Poder visualizar la planta de manera remota.

No obstante, para planificar el correcto desarrollo del *bot*, hay que pensar en los usuarios que van a utilizarlo. Para ello, se va a hacer un pequeño modelado de usuarios y se verá qué características tendrán los mismos.

Se verá también qué requisitos serán necesarios cumplir para que el *bot* cubra el alcance del proyecto y permita interactuar con la planta, tal y como se había planificado.

3.3.1. Modelado

Para modelar el tipo de usuarios que harán uso del *bot*, hay que pensar en qué tipo de personas que suelen cuidar plantas, y además pueden estar interesadas en monitorizar y controlar acciones como el riego a distancia.

Se plantea que los usuarios tendrán que ser personas de edades comprendidas entre los 18 y 50 años, tanto hombres como mujeres, y con conocimientos en el cuidado de plantas y en el uso de terminales móviles y aplicaciones de mensajería instantánea, así como aplicaciones de redes sociales.

Eventualmente, movidos por la curiosidad, podrá haber usuarios de otras edades que también deseen utilizar el *bot* para ver cómo pueden colaborar en el cuidado de la planta.

Esto tiene un pequeño inconveniente, sobre todo si los que controlan la planta son muchos, puesto que pueden darse circunstancias de sobre riego, o falta del mismo por falta de comunicación, así como otros posibles malentendidos que puedan perjudicar la salud de la planta.

Por lo tanto, diferenciaremos entre dos tipos de usuarios: los administradores del *bot* y los usuarios autorizados. Los demás usuarios (usuarios no autorizados) no se tendrán en cuenta, ya que no podrán interactuar con el *bot*.

Los usuarios autorizados podrán interactuar con la planta en función de los permisos que se les hayan asignado.

Los permisos, los podrán asignar los usuarios administradores desde el propio *bot*, e irán desde simples consultas del estado de la planta, hasta tener el control total de la misma.

Por lo tanto, se han definido:

- Usuarios no autorizados: no pueden interactuar con el *bot*.
- Usuarios autorizados: pueden actuar con el *bot* en función de sus permisos.
- Administradores: son usuarios autorizados con todos los permisos y pueden gestionar a los usuarios.

3.3.2. Requisitos del bot

Ahora se pasa a ver qué tipo de requisitos necesita cubrir el *bot* tanto en la parte del cuidado de la planta como en la gestión de usuarios que se ha comentado.

Si queremos conseguir que los usuarios sean capaces de recibir toda la información de los sensores conectados a la `Raspberry Pi`, y poder actuar con la planta mediante los actuadores, tendremos que cubrir una serie de requisitos funcionales básicos.

Los requisitos funcionales del *bot* se detallarán a continuación.

Requisitos relacionados con los sensores:

- Ver estado de la planta
- Visualizar temperatura
- Visualizar humedad

- Solicitar una fotografía de la planta

Requisitos relacionados con los actuadores:

- Encender/apagar luz
- Encender/apagar ventilación
- Regar la planta

Y en cuanto a la gestión de usuarios, será necesario gestionar estas funcionalidades:

- Solicitar permisos de uso
- Admitir usuarios que han solicitado permiso
- Restringir usuarios
- Cambiar permisos a usuarios

Ahora, se va a estudiar los perfiles de usuario que se pueden crear, y qué requisitos asignar a cada uno de ellos.

Usuario sin permisos, podrá:

- Solicitar permiso para utilizar el *bot*

Y se definen 3 tipos de usuarios con permisos:

Usuario "supervisor", para supervisar la planta, podrá:

- Solicitar más permisos
- Ver estado de la planta
- Visualizar temperatura
- Visualizar humedad ambiental

Usuario "jardinero", además de los anteriores permisos, podrá:

- Solicitar una fotografía de la planta
- Encender/apagar luz
- Encender/apagar ventilación
- Regar la planta

Usuario "administrador", tendrá todos los permisos anteriores, así como los relativos a la gestión de usuarios. Los propios de gestión de usuarios permitirán:

- Conceder permisos de uso
- Denegar permisos de uso
- Ascender a un usuario a un perfil superior
- Degradar a un usuario a un perfil inferior

Con todo esto, ya se tiene una imagen más clara de todas las acciones que permitir realizar el sistema. En base a ellas, se desarrollarán tanto las soluciones hardware como software.

CAPÍTULO 4
Tecnologías

Tal y como se ha definido previamente, se emplearán sensores y actuadores conectados directamente a la placa `Raspberry Pi` que, a su vez, alojará tanto el *bot* que se desarrollará en lenguaje `Python` como la base de datos necesaria para su funcionamiento. Todas estas tecnologías y las subyacentes se explican en este apartado.

Se comenzará hablando del hardware, con mayor incidencia en `Raspberry Pi`, y después se pasará al software, con mención especial para `Telegram` y la base de datos.

4.1 Hardware

4.1.1. `Raspberry Pi`

`Raspberry Pi` es un ordenador de placa reducida de bajo coste, desarrollado en Reino Unido por la fundación `Raspberry Pi`, con el objetivo principal de incitar tanto a niños como a adultos a que aprendan sobre ordenadores y todo lo relacionado con los mismos.

La idea de desarrollar algo así surgió en 2006 cuando Eben Upton, Rob Mullins, Jack Lang y Alan Mycroft, del laboratorio de informática de la Universidad de Cambridge, constataron cómo habían cambiado los conocimientos de los niños sobre la informática. En la década de los 90, surgió mucha afición por la programación entre la juventud, pero a partir del año 2000 la tendencia fue disminuyendo y esa inquietud se dirigió, principalmente, a la programación web, de mucho más alto nivel.

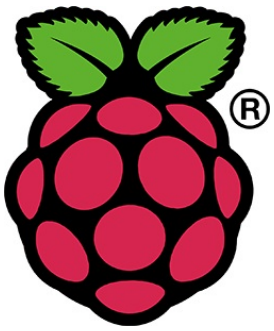

Figura 4.1: Logotipo de `Raspberry Pi`.

La fundación `Raspberry Pi` surgió con un objetivo en mente: Desarrollar el uso y entendimiento de los ordenadores en los niños. Su idea es conseguir ordenadores muy

baratos que permitan a los niños usarlos sin miedo, abriendo su mentalidad, y educándolos en la ética del "ábrelo y mira cómo funciona".

En este proyecto se trabajará con una placa `Raspberry Pi` modelo B. Las características de este modelo son las siguientes:

- Chip: Broadcom BCM2835
- Procesador: ARM 1176JZFS a 700 MHz
- Procesador gráfico: Videocore 4
- Memoria RAM: 512 MB SDRAM 400 MHz
- Salidas de vídeo: HDMI y RCA
- Resolución: 1080p
- Audio: HDMI y 3.5 mm
- USB: 2 x USB 2.0
- Pines GPIO: 26
- Red: Ethernet 10/100
- Almacenamiento: Tarjeta SD
- Alimentación: micro USB, 750mA hasta 1.2A a 5V

Figura 4.2: `Raspberry Pi` modelo B.

4.1.2. Sensores y actuadores

Sensores

En este apartado, veremos las características de los sensores que se emplearán para monitorizar la planta.

En primer lugar, se nombran todas las funciones que cubren los sensores:

- Medida de temperatura
- Medida de humedad
- Toma de fotografías

Estas funciones se traducirán en un sensor de temperatura y humedad y una cámara web.

Para el sensor de temperatura y humedad ambiental, se va a contar con el sensor DHT11. Se trata de un sensor digital de alta fiabilidad y estabilidad, por su señal digital calibrada. Se ha adquirido un sensor con placa PCB, donde va insertado para facilitar su uso.

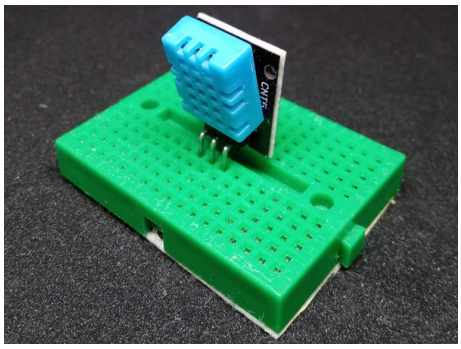

Figura 4.3: Sensor DHT11 de temperatura y humedad.

Como vemos en la figura 4.4, se tiene un pin para tierra, otro para la alimentación y otro que devuelve las medidas de temperatura y humedad tomadas.

Modelo	DHT11
Alimentación	de 3,5 V a 5 V
Consumo	2,5 mA
Señal de salida	Digital

Temperatura	
Rango	de 0ºC a 50ºC
Precisión	a 25ºC ± 2ºC
Resolución	1ºC (8-bit)

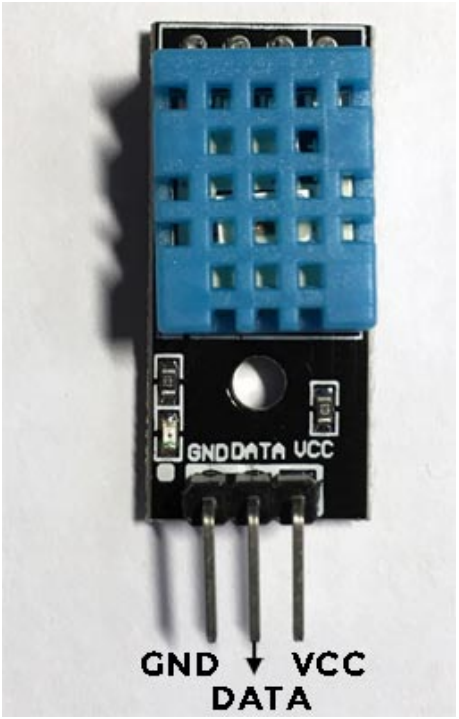

Figura 4.4: Identificación de pines en el sensor DHT11.

Humedad	
Rango	de 20 % RH a 90 % RH
Precisión	entre 0ºC y 50ºC ± 5 % RH
Resolución	1 % RH

Transmisión de datos del sensor DHT11

Aunque se ha dicho que el sensor es digital, en realidad esto no es del todo correcto, ya que realmente el sensor es analógico, pero internamente se realiza la conversión a señal digital, y la devuelve en una trama como la que se muestra a continuación.

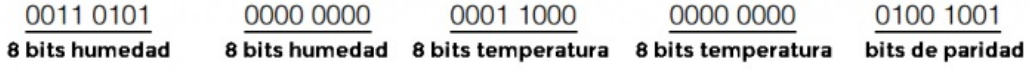

Figura 4.5: Trama de datos que devuelve el sensor DHT11.

Como se puede ver, el primer grupo de 8 bits es la parte entera de la humedad, y el segundo la parte decimal. Lo mismo para la temperatura, donde la parte entera sería el tercer grupo, y la parte decimal el cuarto. Por último se encuentran los bits de paridad para confirmar que no hay datos corruptos.

Actualmente existen librerías que recogen estas tramas y devuelven los valores en decimal. Una de estas librerías es `Adafruit`.

Para poder utilizar la librería `Adafruit`, se deberá instalar desde su repositorio en línea y, una vez hecho, ya se podrá solicitar los datos al sensor, y así recibirlos en consola.

Cámara web

Para la toma de fotografías se va a emplear una cámara web. Se consulta en Internet el listado de cámaras compatibles con `Raspberry Pi` para comprobar si se tiene alguna de las listadas o, en su defecto, buscar una compatible para comprar [22].

Se encuentra que la cámara web "Sony EyeToy for PS2", con la que actualmente ya se cuenta, es compatible con la placa, aunque se indica una observación: "Occasional 'mangled frame' directly connected to Rev 2 Raspberry Pi". Esto quiere decir que, eventualmente, algunas instantáneas tomadas se mostrarán descompuestas. Se considera como un problema asumible, achacado al modelo de la cámara, que no va a impedir el normal funcionamiento del sistema.

Figura 4.6: Cámara web Sony EyeToy para Ps2.

Actuadores

Para actuar con la planta, se contará con una serie de componentes para:

- Ventilación del invernadero
- Riego de la planta
- Iluminación del invernadero

Para ello se precisará de un ventilador, un sistema de riego mediante una pequeña bomba de agua e iluminación de tipo LED.

Ventilador

Para llevar a cabo la ventilación del invernadero, se va a contar con un pequeño ventilador diseñado para refrigerar ordenadores portátiles que, debido a su tamaño, es sufi-

ciente para el cometido, y que tan solo requiere un potencial de 5 voltios para su alimentación.

Figura 4.7: Ventiladores para Cactus Pi.

Datos técnicos:

- Tensión: 5V
- Potencia nominal: 0,75 W

Bomba de agua

En cuanto al riego, existen en el mercado opciones funcionales muy elaboradas, pero se pretendía crear un sistema más artesanal, tomando como ejemplo el proyecto `Tomaatit` visto en el capítulo 2. Para ello, se contará con una pequeña bomba de agua que será capaz de regar las plantas del invernadero.

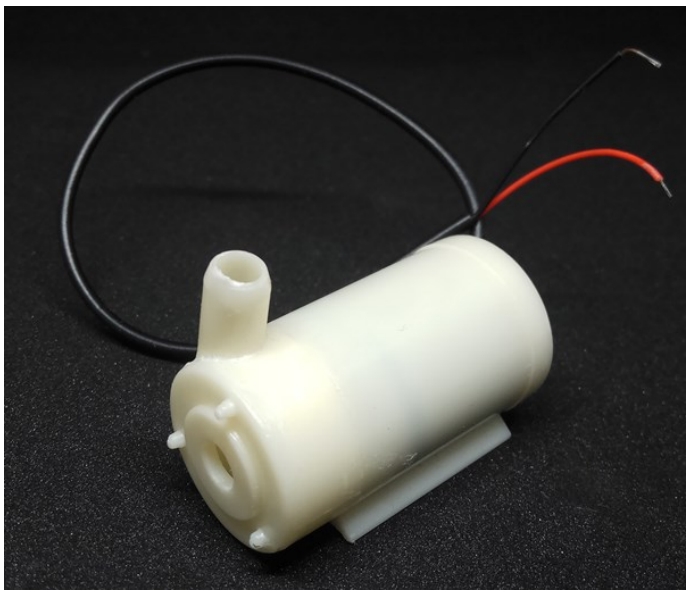

Figura 4.8: Bomba de agua para Cactus Pi.

Los datos técnicos de la bomba son:

- Corriente de funcionamiento: 130-220 mA
- Diámetro de salida: 7,5 mm
- Flujo: de 80 a 120 l/h
- Potencia: de 0.4 a 1.5 W

Iluminación LED

Se va a contar con iluminación LED para el invernadero. De las opciones posibles, se ha optado por adquirir una tira de LED con adhesivo que, aunque en principio viene alimentada mediante conexión USB, se modificará para conectarla directamente a la placa `Raspberry Pi`.

Figura 4.9: Tira de luces LED.

Datos técnicos:

- Tipo de LED: SMD3528
- Corriente: 5V
- Consumo de energía: 2.88 W/m
- Longitud tira de LED: 1m

Pantalla LCD

Finalmente, con referencia al producto `Aerogarden` que se vio en el capítulo 2, se decide colocar una pantalla LCD con información sobre la temperatura y la humedad que se pueda consultar en el propio invernadero de la planta sin tener que hacer uso del *bot*.

La pantalla LCD escogida para este propósito es el módulo I2C LCD1602 sobre una placa de la marca `Sunfounder`.

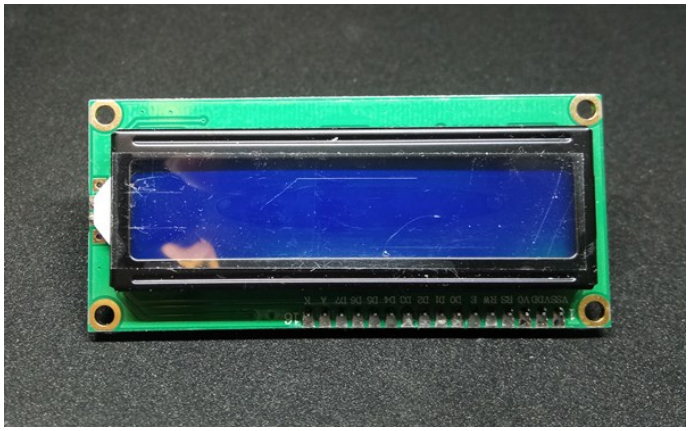

Figura 4.10: Pantalla LCD. Parte frontal.

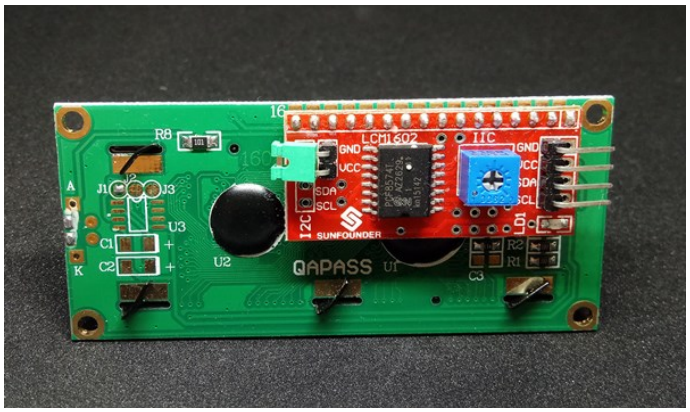

Figura 4.11: Pantalla LCD. Parte trasera.

Detalles técnicos de la pantalla LCD:

- Tensión: 5V
- Tipo LCD: I2C LCD1602
- Potenciómetro para ajustar luz y contraste.
- Conexión mediante puertos: SDA y SCL

4.2 Software

A continuación, se verán las tecnologías software que se van a emplear para el desarrollo del sistema.

4.2.1. Python

Desde su origen, la placa `Raspberry Pi` está íntimamente ligada al lenguaje de programación `Python`, viniendo éste integrado en el sistema operativo Raspbian. Un ejemplo de esto es la librería RPi.GPIO que nos permite utilizar `Python` para configurar los pines

de las conexiones GPIO que trae la placa Raspberry Pi.

Uno de los puntos fuertes de este lenguaje de programación es su sencillez y sintaxis clara, ya que ayuda a profanos en la programación a iniciarse de una manera rápida y fácil.

Figura 4.12: Logotipo de Python.

Python es un lenguaje de programación multiparadigma, lo cual significa que permite tanto la orientación a objetos como programación imperativa, e incluso programación funcional. Se trata de un lenguaje interpretado, de tipado dinámico y multiplataforma. Está registrado bajo una licencia de código abierto, y es mantenido por la Python Software Foundation.

Todo esto hace de Python un lenguaje ideal para el desarrollo tanto de pequeños scripts con los que los usuarios puedan realizar pruebas de concepto en poco tiempo, como proyectos grandes y bien estructurados, como sucede en otros lenguajes como Java o C#.

4.2.2. Telegram Messenger

Telegram Messenger es un servicio de mensajería desarrollado por los hermanos Nikolai y Pave Durov que se estrenó en 2013. Para el envío y recepción de mensajes, desarrollaron la arquitectura llamada MTProto.

Telegram es software libre en su versión *frontend*, y tiene versiones de cliente para todos los sistemas operativos móviles y de escritorio, además de ofrecer una versión web.

Figura 4.13: Logotipo de Telegram Messenger.

Actualmente, Telegram es la única plataforma de mensajería que ofrece una API para el desarrollo de *bot*s. Esto la convierte en la primera opción en la que piensan los desarro-

lladores cuando quieren llevar a cabo un proyecto de estas características.

Además, la comunidad de desarrolladores crece constantemente, y empieza a haber gran cantidad de información disponible en la red, de modo que facilita, en gran medida, la tarea de buscar información y soluciones a problemas recurrentes.

API *Bots*

Para crear un *bot* de `Telegram`, la primera tarea será conocer las posibilidades a nuestro alcance, y estas posibilidades nos las ofrece su API.

Esta se encuentra en: `https://core.telegram.org/bots/api`, y nos ofrece toda la información necesaria para poder crear un *bot*. No obstante, esto se refiere a la parte de implementación, pero primero tendremos que crear nuestro *bot*, y para ello será necesario usar otro *bot*: *@BotFather*.

BotFather

@BotFather es un *bot* creado por `Telegram` para permitir a los desarrolladores crear y administrar sus *bot* de una manera sencilla. Como su nombre indica, es el "padre de todos los *bot*s".

Para crear un *bot* nuevo, la primera tarea es iniciar una conversación con él.

Figura 4.14: Bot que gestiona la creación de bots para Telegram Messenger.

Nos ofrece distintas opciones:

- **/newbot** - crear un nuevo *bot*
- **/mybots** - editar tus *bot*s
- **/mygames** - editar tus juegos

Editar *bot*s:

- **/setname** - cambiar el nombre de un *bot*
- **/setdescription** - cambiar la descripción de un *bot*
- **/setabouttext** - cambiar la ”información sobre” de un *bot*
- **/setuserpic** - cambiar la foto de perfil del *bot*
- **/setcommands** - cambiar los comandos de un *bot*
- **/deletebot** - eliminar un *bot*

Ajustes de *bot*s:

- **/token** - generar un token de autenticación
- **/revoke** - revocar un token
- **/setinline** - modificar el comportamiento del *bot* en mensajes “inline”.
- **/setinlinegeo** - modificar el comportamiento del *bot* en mensajes “inline”.
- **/setinlinefeedback**
- **/setjoingroups** - indica si el *bot* se puede o no unir a grupos
- **/setprivacy** - indica si el *bot* tendrá acceso a los mensajes privados o sólo a los que vayan dirigidos a él

Se ingresa el comando ”/newbot” y pide un nombre para el *bot*. Se introduce un nombre descriptivo y ahora pide un usuario que debe terminar con la cadena ”bot”.

Una vez terminados estos pasos, `@BotFather` facilita un enlace al *bot* y lo más importante, un token para acceder a la API vía HTTP, que será necesario para el *bot* a desarrollar.

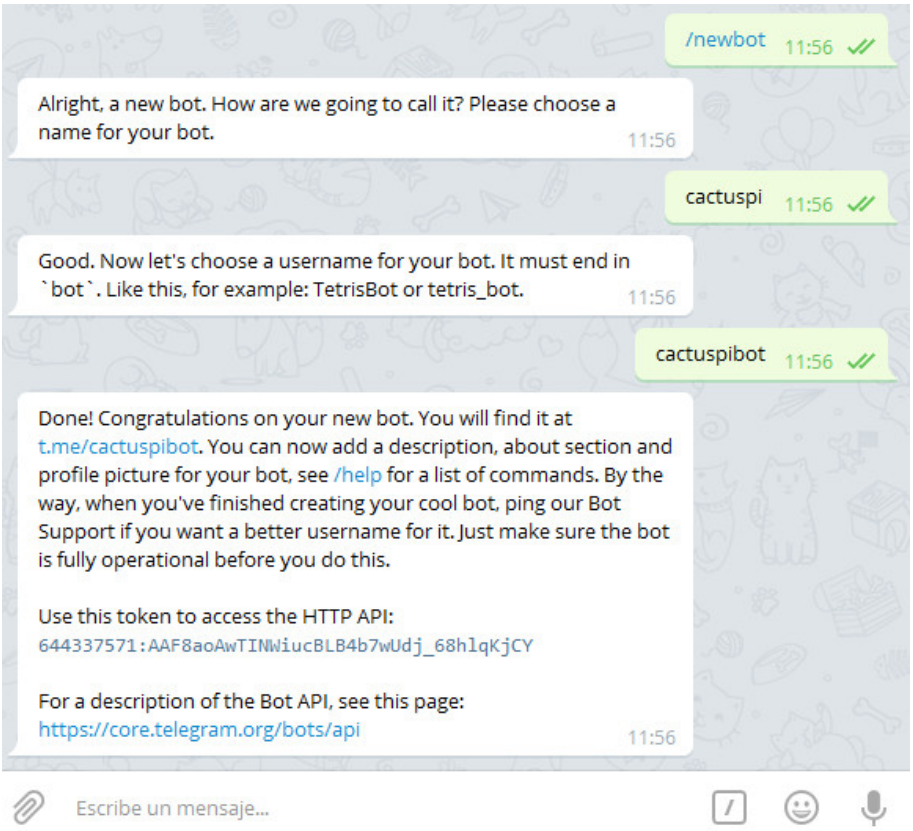

Figura 4.15: Ejemplo de creación de bot usando @BotFather.

pyTelegramBotAPI

Para trabajar con la API de `Telegram`, se va a utilizar una implementación para `Python` llamada `pyTelegramBotAPI`. Esta implementación facilita la tarea de crear bots con el lenguaje de programación `Python` sin tener que lidiar con los ficheros JSON que se envían en las comunicaciones con `Telegram`.

De esta manera, se va a facilitar la creación del *bot* para poder fijar la atención en los aspectos funcionales en lugar de los correspondientes a la comunicación con el programa de mensajería.

La API de esta implementación se encuentra en el repositorio de Github: `https://github.com/eternnoir/pyTelegramBotAPI` donde está mantenido por su creador y la comunidad de desarrolladores.

4.2.3. SQLite

En `Python` existe una propuesta de API estándar para la gestión de bases de datos, de manera que el código sea prácticamente el mismo sin tener en cuenta la base de datos que está funcionando de manera subyacente. Esta especificación se conoce como `Python Database API` o `DB-API`.

Para el propósito que nos ocupa, se trabajará con `SQLite`, una implementación ligera de base de datos relacional que viene integrada con `Python`. Esta ventaja permite trabajar con `SQLite` de una manera rápida y sin tener que instalar librerías externas.

Figura 4.16: Logotipo `SQLite`.

Además, como se verá, trabajar con `SQLite` es sumamente sencillo debido a su integración en `Python`, y va a permitir escribir código rápido y eficiente en muy pocas líneas.

CAPÍTULO 5
Planificación

En este apartado, se van a establecer las fases necesarias para llevar a cabo el proyecto teniendo en cuenta los requisitos y las tecnologías vistas anteriormente.

5.1 Fase inicial: Pruebas y aprendizaje previo

Antes de lanzarse a programar la primera línea de código, se tendrá que probar el funcionamiento de los componentes, buscar información al respecto, y procurar que todo esté en orden para el desarrollo del proyecto.

Primero, será necesario instalar un sistema operativo en la `Raspberry Pi`. A priori, parece que el sistema operativo `Raspbian` es el que ofrece mayor versatilidad a la hora de trabajar con los pines GPIO, y de hacer servir la placa como un servidor. Se valorará la posibilidad de otros sistemas, aunque parte como favorito por toda la información al respecto en la red.

Habrá que analizar si los sensores que se han adquirido para llevar a cabo el trabajo son compatibles con `Raspberry Pi` de manera nativa, o hay que trabajar con librerías de terceros para hacerlos funcionar. Se verá si hay que modificar las librerías para garantizar la compatibilidad, o ya ofrecen todo lo necesario para poder trabajar con ellas.

Será necesario ver qué cámaras web son compatibles con el sistema operativo elegido, y se verá si hay que adquirir una específicamente para este propósito, o si se puede reutilizar una que se tenga previamente.

La placa `Raspberry Pi` se ubicará junto a las plantas, y el modelo B no cuenta con conectividad inalámbrica, por lo que habrá que ver cómo se soluciona este inconveniente. Se puede contar con un cable de red suficientemente largo para tal propósito, o adquirir un adaptador inalámbrico para dotar de dicha conectividad a la placa.

Para la conexión con la placa, tampoco se va a contar con monitor y periféricos dedicados, por lo que habrá que tener esto en cuenta, y establecer una conexión remota con la misma. Se valorarán conexiones tipo VNC, escritorio remoto, SSH, etc.

Antes de comenzar a crear un *bot*, se partirá de *bot*s sencillos, y de ahí se irá incrementando la complejidad del mismo hasta conseguir el que se desea. Para ello, el servidor

tendrá que estar funcionando en los términos que se han definido previamente, ya que será necesario que esté operativo para tener un entorno de desarrollo.

Para no perder los cambios realizados y llevar un cierto control de versiones, se probarán distintas alternativas, desde repositorios GIT hasta opciones más sencillas como `Dropbox`, que permitan tener salvaguarda de los ficheros y recuperar versiones anteriores.

5.2 Fase de montaje

El primer paso constará en preparar la placa `Raspberry Pi` para trabajar como un servidor, y para ello será necesario instalarle un sistema operativo. Una vez el sistema esté funcionando, habrá que establecer una dirección IP estática para la `Raspberry Pi` a fin de tenerla localizada en la red, y acto seguido establecer un canal de comunicación con la misma.

En cuanto al montaje del invernadero, será un proceso incremental, añadiendo los elementos necesarios a cada avance del *bot*. Esto quiere decir que se comenzará ubicando la placa `Raspberry Pi` en el invernadero y, una vez se tenga la conectividad establecida y el *bot* de prueba funcionando, se irán añadiendo elementos como sensores o actuadores, se programará en el *bot* el código necesario para interactuar con los mismos, y se añadirá otro elemento en el invernadero hasta completar el conjunto de elementos que se han previsto conectar al mismo.

De esta forma, cuando se acabe con este proceso, tendremos todos los elementos conectados y operativos a disposición del usuario del *bot*.

Este proceso permitirá localizar fácilmente errores y posibles incompatibilidades entre componentes, o problemas que puedan surgir derivados de la adición de nuevos componentes al sistema.

5.3 Fase de programación

Una vez se tenga establecida la comunicación con el servidor, se instalará en el mismo la API del *bot*, y se creará un directorio para el entorno de desarrollo. En dicho directorio se copiará el *bot* de prueba que ofrece la propia API.

En base a ese *bot* se comenzará el desarrollo de nuestro *bot*. Habrá que ver cómo separar el código para estructurarlo adecuadamente y poder trabajar con el mismo de una manera sencilla y eficiente.

Se irá ampliando el *bot* con todas las funcionalidades que se desean obtener del mismo mientras se irán conectando al *bot* los elementos necesarios con los que habrá que interactuar.

El código del *bot* se dará por finalizado cuando interactúe con todos los sensores y actuadores, y ofrezca al usuario una interfaz clara y sencilla.

CAPÍTULO 6
Desarrollo

Ya hemos visto qué requisitos tiene que cumplir el *bot* para alcanzar los objetivos marcados, se han analizado las tecnologías que se van a emplear, y se ha planificado el proceso que hay que seguir.

Ahora, por lo tanto, se va a ver cómo se desarrolla el proyecto desde la instalación del sistema operativo en la `Raspberry Pi` hasta la configuración del último sensor en el invernadero.

Se da por entendido que ya se han realizado las pruebas necesarias con los elementos definidos como para poder abordar el desarrollo.

6.1 Arquitectura general

En primer lugar, se analiza la arquitectura del sistema que se desea implementar, la cual se puede ver en la siguiente figura.

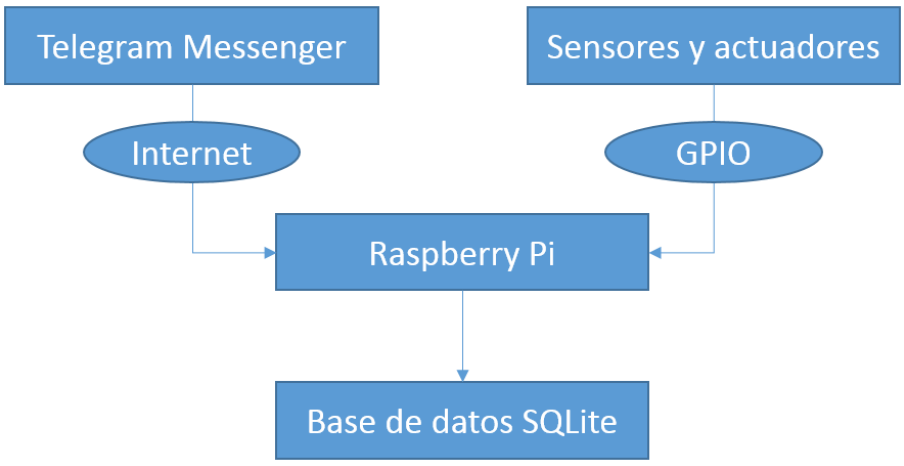

Figura 6.1: Arquitectura del sistema Cactus Pi.

Por un lado se tiene la plataforma de `Telegram`, que gestiona la comunicación con el usuario a través de Internet. Por otro lado, vemos los sensores y actuadores que se conectan a `Raspberry Pi` mediante pines GPIO y conexiones USB. Todo ello se procesa en la placa `Raspberry Pi`, y a su vez, se almacena información en la Base de datos SQLite que

gestiona el *bot*.

6.2 Preparación del servidor

Como se ha planificado, se comienza a trabajar con la placa `Raspberry Pi`. Lo primero será instalar el sistema operativo en la tarjeta de memoria que se inserta en la placa.

En concreto, se cuenta con una tarjeta Micro-SD de 8GB de espacio con un adaptador de tarjeta SD.

Figura 6.2: Tarjeta Micro-SD con adaptador.

El sistema operativo elegido para hacer funcionar la placa como un servidor apto para el proyecto es `Raspbian`, el sistema recomendado por la fundación `Raspberry Pi`.

Este sistema se basa en una distribución de GNU/Linux llamada Debian, y está optimizado para aprovechar al máximo las características de la placa. Para este proyecto, se utilizará la versión `Raspbian Stretch` publicada el 7 de septiembre de 2017.

Para la instalación del sistema operativo en la tarjeta SD, se hace uso del programa Etcher [18], que permite "flashear" la imagen del sistema operativo en la tarjeta SD, formateándola previamente.

Una vez está instalado el sistema operativo en la tarjeta, la introducimos en la placa y la iniciamos por primera vez conectándola a la corriente.

La primera configuración al iniciar el sistema operativo es establecer una dirección IP estática para la `Raspberry Pi`. Para ello, se edita el fichero `interfaces` que se encuentra en la ruta:

```
/etc/network/interfaces
```

Y se reserva una dirección IP que permitirá tener localizada la placa en la red donde se va a conectar [17].

Se ha optado por emplear un adaptador inalámbrico para dotar de conectividad a la placa. Este tipo de conector es Plug&Play, y no requiere de controladores específicos para el sistema `Raspbian`.

Figura 6.3: Adaptador inalámbrico USB.

Con la placa conectada a la red en una dirección específica de manera inalámbrica, se puede situar ya en el invernadero. En concreto, se localizará en la parte exterior, para evitar contacto con agua o humedad, y aun así tenerla cerca de todos los sensores y dispositivos que se van a conectar.

Ahora se detalla cómo establecer una conexión remota para conectar con la placa y preparar el entorno de desarrollo.

Se empleará el programa `MobaXterm` (figura 6.4), que es un cliente de conexión remota por terminal SSH que ofrece funcionalidades interesantes como conexión SFTP, que permitirá realizar las actualizaciones que se realicen al *bot* en desarrollo.

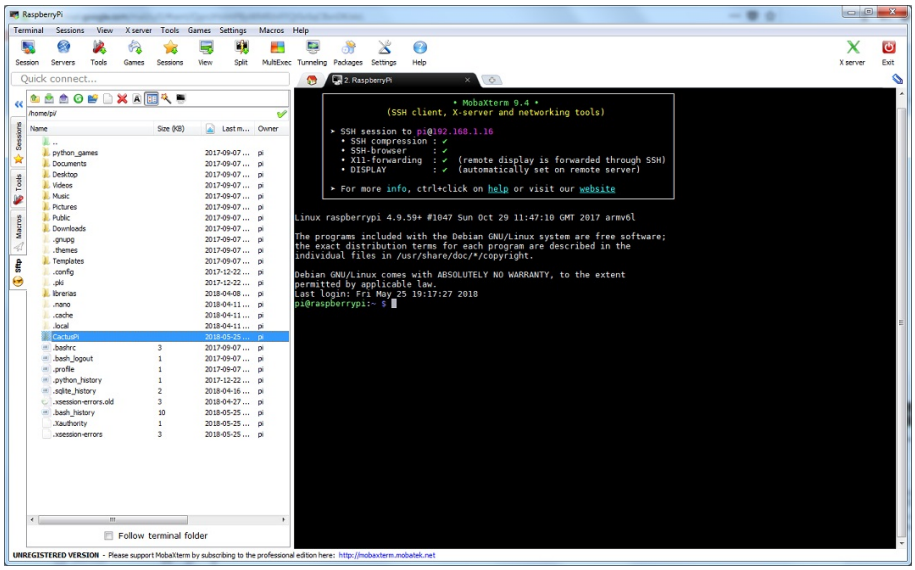

Figura 6.4: Programa MobaXterm para establecer conexión SSH con la Raspberry Pi.

Para su funcionamiento en la placa `Raspberry Pi`, es necesario, previamente, permitir las conexiones mediante el protocolo SSH.

Una vez establecida la conexión con la placa, se pasa a instalar la librería `pyTelegramBot` [20], que instalará todos los ficheros necesarios para poder ejecutar el *bot* que se va a desarrollar.

Se crea una carpeta "CactusPi" donde se alojarán todos los archivos del *bot*, y se procede a instalar la librería pyTelegramBot. Para ello, es tan sencillo como introducir en el terminal:

```
$ pip install pyTelegramBotAPI
```

y la instalación se realiza automáticamente gracias a *pip*, un gestor de paquetes de `Python`.

Antes de poner ningún *bot* en funcionamiento, será necesario crearlo en la plataforma de `Telegram`, como ya se vio en el capítulo de tecnologías, haciendo uso de @Botfather y configurarlo para que muestre un mensaje de bienvenida a los usuarios que quieran utilizarlo.

Figura 6.5: Bienvenida del bot.

Una vez se tiene el *bot* creado, @Botfather ofrece un `TOKEN` que servirá para autenticar el *bot* en ejecución, y poder realizar la comunicación de mensajes entre usuarios y el mismo.

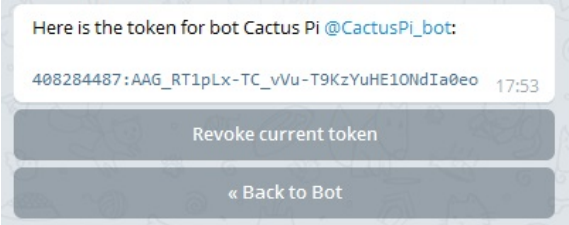

Figura 6.6: Token del bot `Cactus Pi`.

Para probar el funcionamiento de la librería recién instalada, se va a copiar el *bot* básico que ofrece la misma y se pondrá en funcionamiento. El código del *bot* básico es el siguiente:

```
import telebot

bot = telebot.TeleBot("TOKEN")

@bot.message_handler(commands=['start', 'help'])
def send_welcome(message):
        bot.reply_to(message, "Howdy, how are you doing?")

@bot.message_handler(func=lambda message: True)
def echo_all(message):
        bot.reply_to(message, message.text)

bot.polling()
```

Como se puede observar, se trata de un *bot* sencillo que contesta a los comandos /start y /help con un mensaje: "Howdy, how are you doing?". Actualmente no realiza ninguna acción más, pero en base a este se desarrollará el *bot* de `Cactus Pi`.

Se modifica el string "TOKEN" por el que devuelve @BotFather y se lanza la ejecución del *bot*:

```
$ python echo_bot.py
```

Se comprueba que funciona correctamente.

Con esta primera fase superada, se puede proseguir con el desarrollo del sistema. Se van a explicar los pasos que se dan para desarrollar al completo el sistema `Cactus Pi`, sin tener en cuenta el orden cronológico, a fin de organizar de una manera más estructurada todo el proceso.

Primero se explicará cómo se conectan todos los sensores y actuadores, y posteriormente se detallará el desarrollo del *bot* y se mostrarán capturas de pantalla del mismo en funcionamiento.

6.3 Conexión de sensores y actuadores a Raspberry Pi

Para el montaje del sistema `Cactus Pi` se ha decidido adquirir un pequeño invernadero que contenga todo lo que se ha planificado instalar.

Figura 6.7: Invernadero para plantas empleado en el sistema `Cactus Pi`.

Se trata de un pequeño invernadero de paredes de metacrilato, que va a permitir trabajar fácilmente e instalar todos los componentes necesarios para cubrir los requisitos previstos.

A continuación, se va a explicar cómo se conectan los sensores y actuadores que se han presentado en el apartado de tecnologías.

Como se ha indicado, se conectarán de manera incremental, mientras se va programando la parte software que tiene que ver con los mismos.

6.3.1. Sensor de temperatura y humedad ambiental

El sensor de temperatura y humedad se va a instalar en un punto medio del invernadero, que permita recoger valores característicos.

El esquema de conexión con la placa `Raspberry Pi` se ve en la figura 6.8

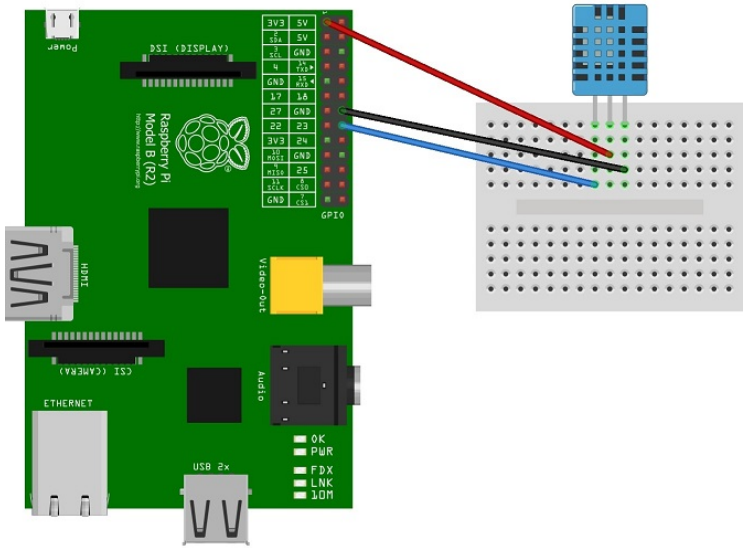

Figura 6.8: Conexión del sensor de temperatura y humedad a la Raspberry Pi.

Para obtener valores característicos, se ha instalado en el invernadero sobre el metacrilato, a media altura, alejado de la ventilación, tal y como se ve en la figura 6.9.

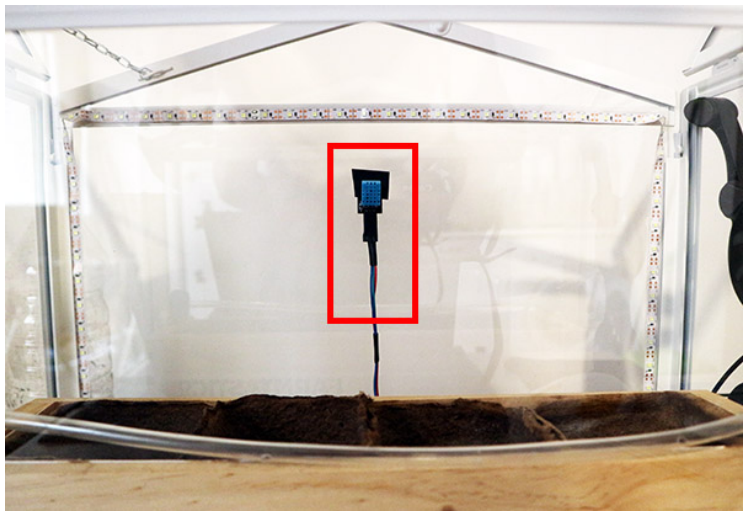

Figura 6.9: Sensor de temperatura y humedad instalado en invernadero de Cactus Pi.

6.3.2. Ventilación

Para la ventilación se han valorado distintas opciones, una de ellas era proporcionar ventilación directa a las plantas, pero se ha podido comprobar que la ventilación directa a las plantas de interior puede provocar la pérdida de hojas. [21]

Por ello, se decide que el ventilador realice una tarea de extracción de aire caliente del invernadero. De esta forma, no va a generar corrientes de aire que puedan perjudicar a las plantas, y va a conseguir el objetivo que se persigue.

El esquema de conexión con la placa `Raspberry Pi` se ve en la figura 6.10.

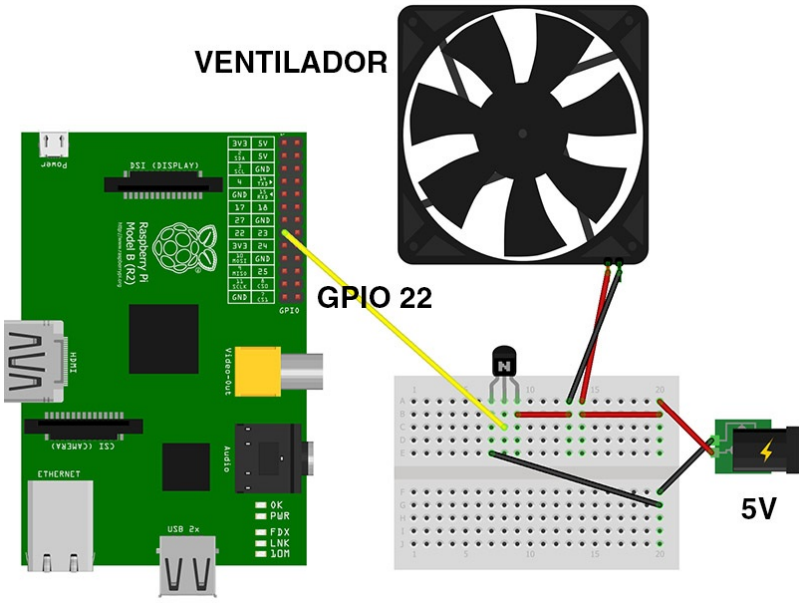

Figura 6.10: Esquema de la conexión del ventilador con la Raspberry Pi.

El ventilador se ha instalado en el invernadero de la manera que vemos en la figura 6.11.

6.3.3. Iluminación

Para la instalación de la tira de LED que va a iluminar el invernadero, se emplea el propio adhesivo que incluye la tira, y se coloca en una de las paredes de metacrilato del invernadero, aprovechando que se adhiere a la perfección.

El esquema de conexión con la `Raspberry Pi` corresponde con la figura 6.12.

El resultado es el que vemos en la figura 6.13.

6.3.4. Sistema de riego

En cuanto al sistema de riego, ya se ha visto la bomba de agua que se va a emplear. Sólo faltaría definir cómo se instala en el invernadero para hacer funcionar el sistema.

El esquema de la conexión con la `Raspberry Pi` se puede ver en la figura 6.14.

En este caso, no se ha usado alimentación externa; puesto que se dispone de alimentación suficiente en la placa. Hay que indicar que, la placa no es capaz de alimentar simultáneamente la bomba de agua, la ventilación y la iluminación. Es por ello que, se ha separado la manera de alimentar cada dispositivo.

Como depósito de agua se utilizará un frasco de vidrio reciclado con suficiente capacidad como para permitir más de diez riegos. La bomba de agua se fija con silicona en la base del recipiente (ver figura 6.15).

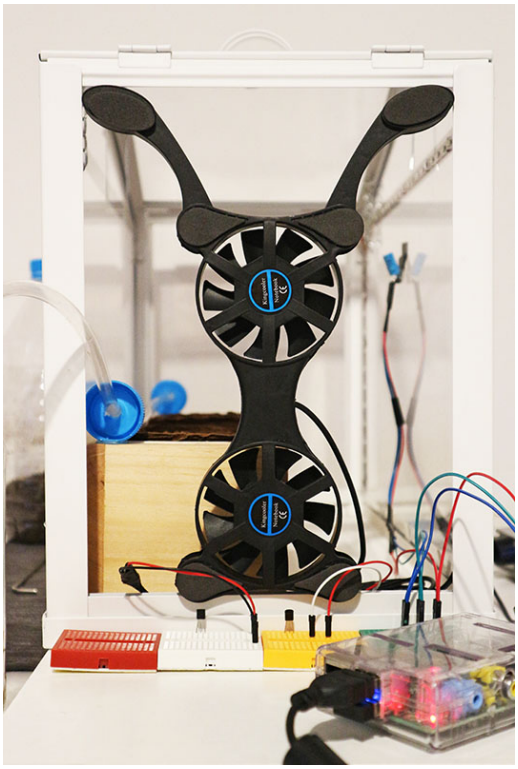

Figura 6.11: Ventilación instalada en invernadero.

Para transportar el agua, se empleará un tubo flexible de PVC de 7mm de diámetro, al que se le practicarán una serie de agujeros en los puntos de riego.

De esta forma, el usuario podrá ir rellenando el recipiente cuando lo considere oportuno con previsión de las necesidades de riego de la planta.

En la figura 6.16, se puede ver el montaje en el invernadero.

6.3.5. Cámara web

La cámara web, se conectará mediante USB a la propia `Raspberry Pi`. Esto va a facilitar mucho la toma de fotografías, pero será necesario instalar una librería que permita realizar fotografías y almacenarlas en el servidor para poder enviarlas a los usuarios.

Se valoran distintas opciones de librerías que manejan la adquisición de imágenes mediante cámara web y, en las pruebas realizadas, se decide optar por `fswebcam`. Esta librería se puede instalar directamente desde el repositorio de `Raspbian` con la orden:

```
$ sudo apt-get install fswebcam
```

Con esto, queda instalado el software que va a gestionar la cámara web. Se define el siguiente script que se llamará cuando un usuario solicite una imagen:

```
# -*- coding: utf-8 -*-
import threading
```

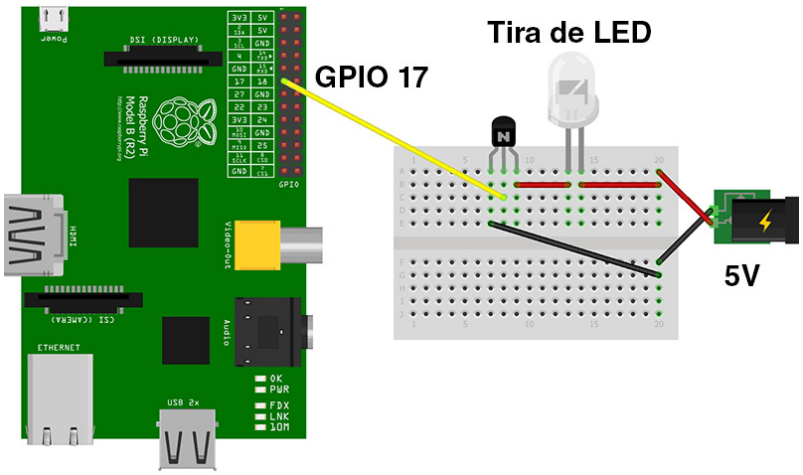

Figura 6.12: Esquema de la conexión de la tira de LED con la Raspberry Pi.

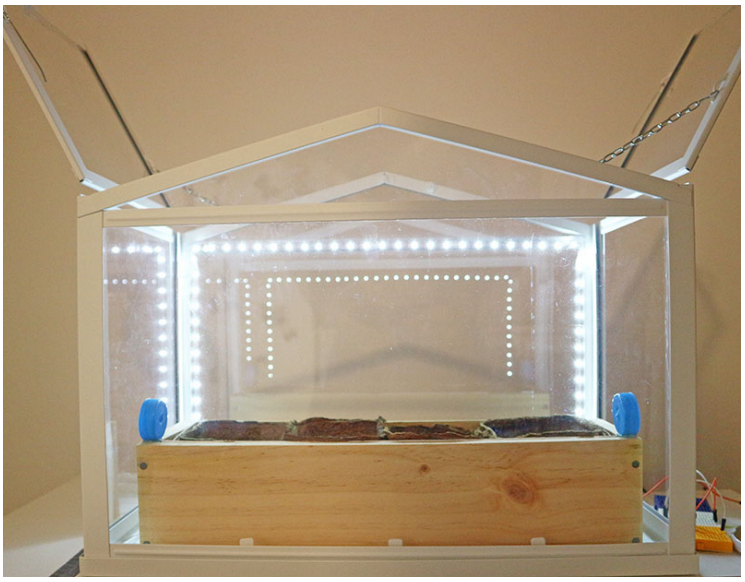

Figura 6.13: Iluminación LED instalada en invernadero.

```
import os

def takePhotoThread(bot,cid):
        print(threading.current_thread().getName())
        try:
                bot.send_message(cid, msg['user_msg']['wait_photo'])
                os.system('fswebcam -r 640x480 img/foto.jpg --title "Cactus
                 ↪  Pi Bot"')
                photo = open('img/foto.jpg', 'rb')
                bot.send_photo(cid, photo)
                os.system('rm img/foto.jpg -f')
        except:
                bot.send_message(cid, 'No se pudo realizar la fotografía')

def start(b,cid):
```

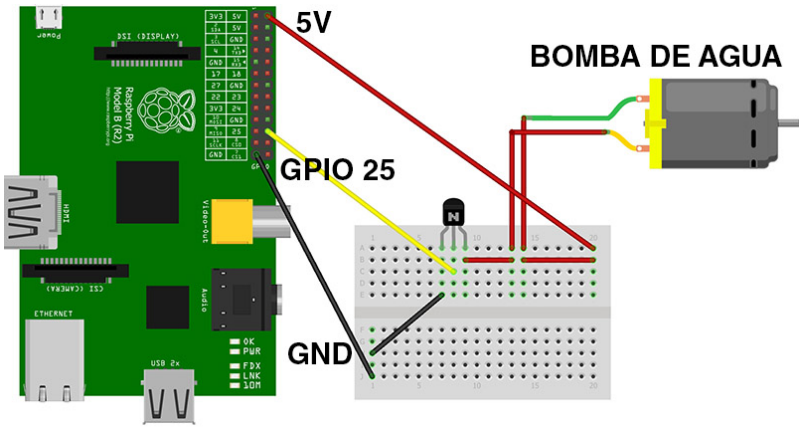

Figura 6.14: Esquema de conexión de la bomba con la placa.

```
photoThread = threading.Thread(target=takePhotoThread,
 ↪  args=(b,cid))
photoThread.start()
```

Se realizará una fotografía de tamaño 640x480 píxeles, y se almacenará en el directorio /img con el nombre foto.jpg y título "Cactus Pi *Bot*". Esta imagen se enviará por mensaje al usuario empleando su id de `Telegram` y, acto seguido, se borrará para no almacenar "basura" en el sistema, dado que no aporta ningún tipo de interés almacenar la imagen.

6.3.6. Pantalla LCD

La conexión de la pantalla LCD (módulo I2C LCD1602) a la `Raspberry Pi` no supone muchas dificultades. El módulo facilita su conexión y el trabajo a alto nivel con la misma.

El esquema de conexión con la `Raspberry Pi` se puede ver en la figura 6.17

Para su funcionamiento, se crea el siguiente script que consulta la base de datos y muestra la temperatura y humedad en pantalla.

```
#!/usr/bin/env python
import LCD1602
import time
import cactus_functions as cfun

def setup():
        LCD1602.init(0x27, 1)
        temp,hum = cfun.getTempHum()
        LCD1602.write(0, 0, '@CactusPi_bot')
        LCD1602.write(1, 1, 'T:'+str(temp)+'g'+' H:'+str(hum)+'%')
        time.sleep(2)

def destroy():
        pass

if __name__ == "__main__":
        try:
```

Figura 6.15: Bomba de agua instalada sobre base de recipiente.

```
        setup()
        while True:
                pass
except KeyboardInterrupt:
        destroy()
```

El resultado final se puede ver en la figura 6.18.

A modo de resumen, se muestra en la figura 6.19 un esquema de uso de los pines ocupados en la `Raspberry Pi` por los diferentes dispositivos.

6.4 Desarrollo del bot de Telegram

6.4.1. Arquitectura software

Antes de desarrollar el *bot*, se busca crear una arquitectura estructurada que permita trabajar de una manera eficiente y organizada; para ello se propone hacer una división de ficheros según su función dentro del *bot*. El esquema sería el que vemos en la figura siguiente.

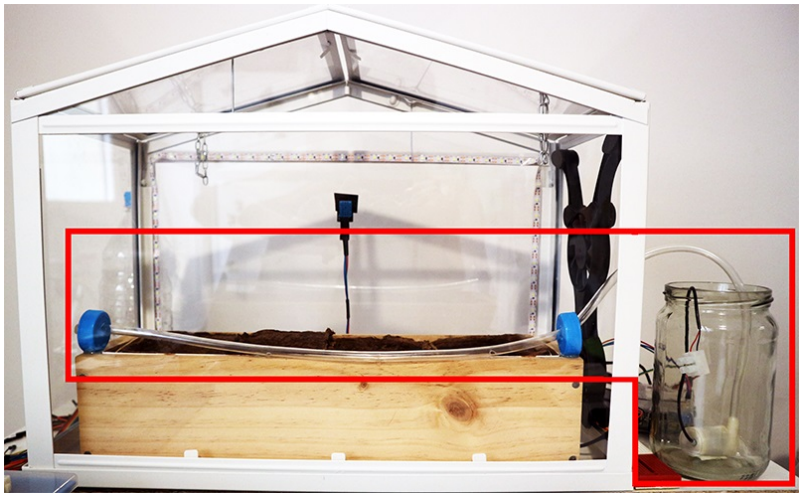

Figura 6.16: Sistema de riego instalado en invernadero de Cactus Pi.

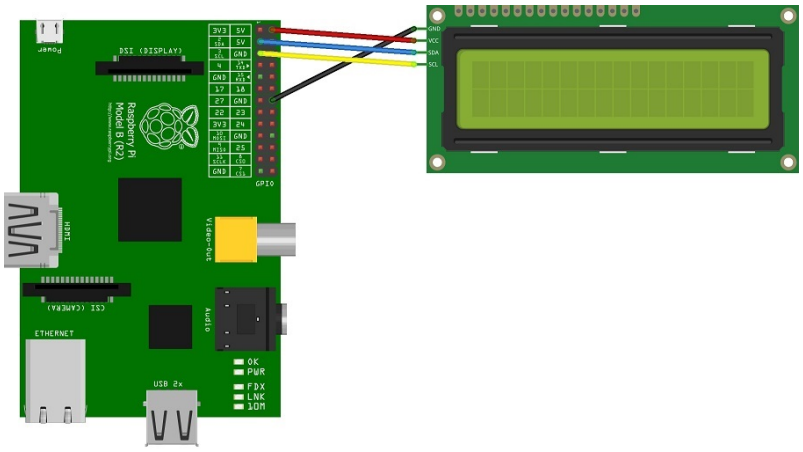

Figura 6.17: Conexión de la pantalla LCD con la Raspberry Pi.

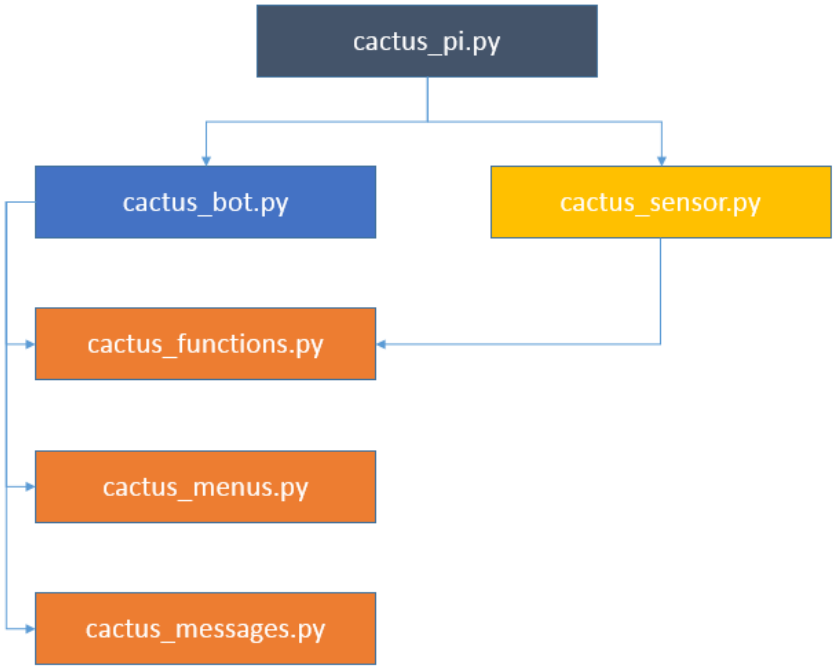

Figura 6.20: Arquitectura del *bot* Cactus Pi.

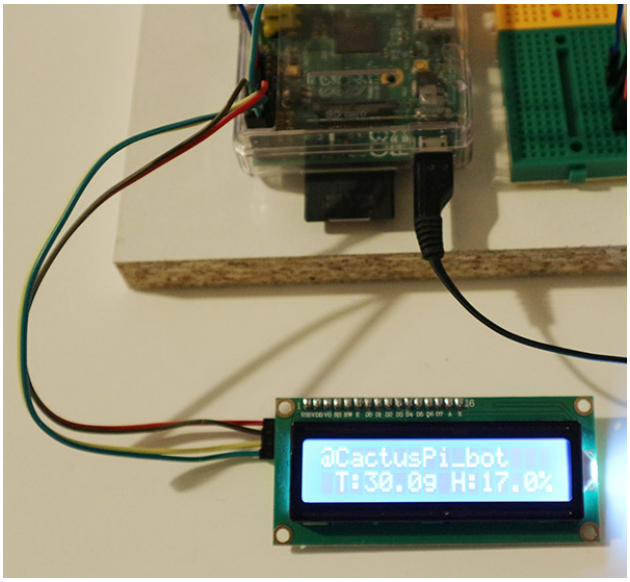

Figura 6.18: Pantalla LCD conectada a la Raspberry Pi.

Se tiene el fichero "cactus_pi.py" cuya función será lanzar la ejecución tanto del *bot* como del *script* responsable de las lecturas de los sensores de temperatura y humedad.

Después tendremos el fichero "cactus_bot.py" que actúa como *bot* propiamente dicho. Este script, se encargará de recibir los mensajes de usuario y procesarlos correctamente.

En el fichero "cactus_functions.py" se escribirán todas las funciones de las que se haga uso en el *bot*, para permitir la reutilización de código.

Finalmente, se tienen los ficheros "cactus_menus.py" y "cactus_messages.py" en los que se han escrito los menús que se le devuelven al usuario dependiendo de las acciones que seleccione, y los mensajes que recibe en consecuencia.

Se puede consultar el código completo en el anexo, donde está el resultado de este proceso de implementación.

6.4.2. Módulos del bot

Ya se ha visto la estructura en la que se va a dividir el *bot*. Ahora, se verán los distintos módulos definidos con un poco más de detalle.

Módulo principal: fichero cactus_bot.py

El fichero cactus_bot.py define el propio *bot*. Es el encargado de recibir, procesar y responder a los mensajes que le llegan. Para ello, en primer lugar, se define una función "listener" que recibe cada uno de los mensajes, extrae el chat_id de cada uno de ellos, y procesa el texto que contiene el mensaje.

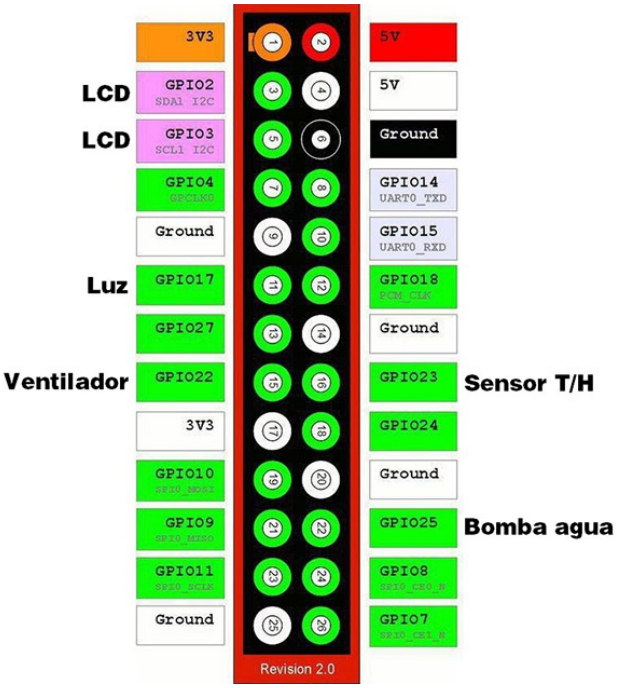

Figura 6.19: Esquema de uso de los pines GPIO de la Raspberry Pi.

El "chat_id" es el número identificador que otorga `Telegram` a cada uno de los usuarios de la aplicación de mensajería. Es único y no se puede modificar.

Para procesar los mensajes, existen diferentes métodos. Se diferencian por el tipo de manejador a emplear.

Para responder a los comandos, se usa un manejador del estilo:

```
@bot.message_handler(commands=['start'])
```

En este caso, el *bot* procesaría un mensaje que llegase con el comando "/start".

Si, por el contrario, se desea responder a un mensaje de texto concreto, habría que utilizar un manejador con función lambda:

```
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") == 'Main menu')
```

De esta manera, cualquier mensaje que reciba el *bot* y que coincida con la función lambda definida, servirá para activar el manejador.

Como se puede ver en el código, en muchos casos se combinan un comando y una función lambda en el mismo manejador, a fin de ofrecer mayor flexibilidad al usuario. Por ejemplo, para el manejador que lleva al usuario al menú principal, se han definido tanto el comando "/main" como la función lambda vista en el ejemplo anterior.

Dentro del *bot* se han definido manejadores de usuario y de administrador. Cada uno de ellos define cómo va a responder el *bot* a los mensajes que reciba de los usuarios. En los manejadores de administrador se comprueba que el "chat_id" que adjunta el JSON

corresponde al de un administrador. En caso contrario, el *bot* no responderá ante el comando o mensaje recibido.

Finalmente, también se han definido manejadores que responden ante expresiones regulares. Como ejemplo tenemos las que permiten a un administrador gestionar los permisos de un usuario:

```
@bot.message_handler(regexp="/user_")
```

Este manejador se activará cuando reciba un comando que contenga la cadena "/user_". En el *bot* se crea este comando concatenando a esta cadena el número de id de los usuarios. Y esto permite al administrador del *bot* editar un usuario concreto. Por ejemplo, un usuario con id 123456 registrado en el *bot*, generaría el comando "/user_123456" para que el administrador pueda editar sus permisos.

Para reutilizar código, todas las funciones que se emplean dentro del *bot* se han definido en un fichero a parte, tal y como se ha visto en la arquitectura software.

Módulo de funciones: fichero cactus_functions.py

Como se ha comentado previamente, este módulo contiene todas las funciones que se emplean en el *bot*, permitiendo así reutilizar código a lo largo del programa.

También ayuda a separar el *bot* de la base de datos. Todas las interacciones con la misma se realizarán desde este módulo, y eso ayuda a la localización de errores.

Como se puede apreciar, internamente el módulo se divide en funciones de los sensores, de actuadores, usuarios y administradores.

Todas las funciones están comentadas, lo que permiten entender fácilmente lo que realiza cada una de ellas.

Módulo de mensajes: fichero cactus_messages.py

El módulo de mensajes es un módulo diseñado para separar la parte lógica del *bot* de los mensajes que se van a enviar al usuario. Esto facilitaría, en un futuro, la traducción del *bot* a otros lenguajes, o la corrección de estos mensajes de una manera más eficiente.

Se trata de código `Python`, pero podría no parecerlo, ya que es únicamente un diccionario con dos entradas principales: user_msg y admin_msg, o lo que es lo mismo: mensajes de usuario y mensajes de administrador.

Así, cuando se quiere hacer referencia a un mensaje concreto desde el módulo principal, es tan fácil como escribir msg[user_msg][start]. De esta manera se tienen todos los mensajes concentrados en un único punto, facilitando así la gestión de los mismos.

Módulo de menús: fichero cactus_menus.py

Del mismo modo que los mensajes se han separado en un módulo a parte, se crea también un módulo para alojar los menús que se van a mostrar al usuario.
Es fácil entender la razón para separar los menús del código principal cuando se ve un ejemplo. Si se quiere que el usuario vea el menú principal, se llama al siguiente código:

```
def main_menu(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Plant status')
        itembtn2 = types.KeyboardButton('Actions')
        itembtn3 = types.KeyboardButton('Options')
        itembtn4 = types.KeyboardButton('Help')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)
```

Y el usuario verá lo siguiente:

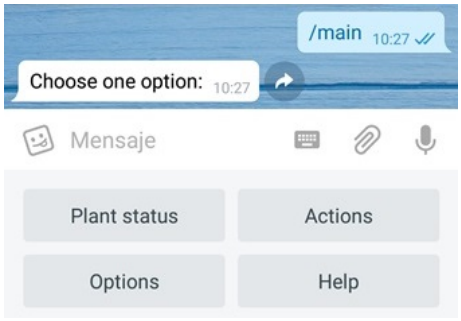

Figura 6.21: Menú principal del bot.

Y de idéntica manera ocurrirá con el resto de menús definidos en este módulo.

Es evidente la ventaja que supone tener este módulo separado del *bot* principal, ya que permite definir los menús de manera independiente y poder trabajar con ellos para su edición y posibles mejoras de forma más rápida.

Módulo del sensor: fichero cactus_sensor.py

Para finalizar, se comentará el módulo que gestiona el sensor. Este módulo, no depende del *bot* principal, sino que se ejecuta de manera paralela al mismo, y totalmente independiente.

Este módulo tampoco requiere de conexión a Internet, puesto que su cometido es registrar los datos de temperatura y humedad, y registrarlos en la base de datos.

Este módulo también se ayuda del módulo de funciones para hacer gestiones sobre la base de datos. En concreto, comprueba los valores máximos y mínimos de temperatura, para compararlos con los actuales y ver si es necesario modificarlos.

Como se puede ver en los comentarios del código, lo que realiza este módulo es:

1. Obtener la humedad y la temperatura desde el sensor.
2. Almacenar los datos recogidos en la base de datos.
3. Se recuperan los valores máximos y mínimos del día actual.
4. Si no hay registro del día, se crea uno nuevo.
5. Se actualiza si varían el máximo o el mínimo.
6. Y espera 5 minutos para volver a comprobar la temperatura y humedad.

6.4.3. Base de datos

Antes de lanzar la ejecución del *bot*, hay que crear la base de datos que alojará toda la información recogida por el *bot* y los sensores. Para ello se realiza el script que vemos a continuación.

```
import sqlite3
#Creamos la conexion con la base de datos
conn = sqlite3.connect("cactuspi.sqlite3")
#Creamos el cursor
cur=conn.cursor()

#Tabla de usuarios
cur.execute("DROP TABLE IF EXISTS Users")
cur.execute("CREATE TABLE Users (id INTEGER NOT NULL, first_name
 ↪  varchar(255) NOT NULL, username varchar(255), user_type INTEGER NOT
 ↪  NULL DEFAULT '0', language_code varchar(25))")

#Tabla de solicitudes
cur.execute("DROP TABLE IF EXISTS Requests")
cur.execute("CREATE TABLE Requests (id INTEGER NOT NULL, text TEXT)")

#Tabla de historico de temperatura y humedad
cur.execute("DROP TABLE IF EXISTS hist_Temp_Hum")
cur.execute("CREATE TABLE hist_Temp_Hum (day varchar(255), temp_min REAL,
 ↪  time_temp_min varchar(255), temp_max REAL,  time_temp_max varchar(255),
 ↪  hum_min REAL, time_hum_min varchar(255), hum_max REAL, time_hum_max
 ↪  varchar(255))")

#Tabla de temperatura y humedad
cur.execute("DROP TABLE IF EXISTS Current_Temp_Hum")
cur.execute("CREATE TABLE Current_Temp_Hum (time varchar(25), temp REAL,
 ↪  hum REAL)")

#Tabla de temperatura actual
cur.execute("DROP TABLE IF EXISTS Current_Temp")
cur.execute("CREATE TABLE Current_Temp (temp REAL)")

#Tabla de humedad actual
```

```
cur.execute("DROP TABLE IF EXISTS Current_Hum")
cur.execute("CREATE TABLE Current_Hum (hum REAL)")

#Tabla de humedad del suelo actual
cur.execute("DROP TABLE IF EXISTS Current_ground_Hum")
cur.execute("CREATE TABLE Current_ground_Hum (ghum REAL)")
cur.close()
conn.commit()
```

Se ejecuta el script:

```
$ python crearbasedatos.py
```

Con esto, ya se habría creado la base de datos y sólo faltaría lanzar la ejecución el *Bot*.

Se ejecuta el comando:

```
$ python cactus\_pi.py
```

Este comando lanzará la ejecución del fichero cactus_pi.py, que llamará a su vez al *bot* y al script de los sensores. Con esto, el *bot* comenzará a procesar mensajes, y los sensores a almacenar la información en la base de datos. El *bot* estará así listo para trabajar.

```
pi@raspberrypi:~ $ cd CactusPi/
pi@raspberrypi:~/CactusPi $ ls
cactus_bot.py          cactus_lcd.py    cactus_menus.pyc     cactus_pi.py      img          scripts
cactus_functions.py    cactus.log       cactus_messages.py   cactuspi.sqlite3  LCD1602.py
cactus_functions.pyc   cactus_menus.py  cactus_messages.pyc  cactus_sensor.py  LCD1602.pyc
pi@raspberrypi:~/CactusPi $ python cactus_pi.py
pi@raspberrypi:~/CactusPi $
Sensor running

Bot running
```

Figura 6.22: *Bot* en ejecución.

6.4.4. Cactus Pi en ejecución

Con el *bot* en ejecución, los usuarios podrán interactuar con el mismo e ir monitorizando y actuando con la planta a través de su terminal móvil.

A continuación, se van a mostrar capturas de pantalla del *bot*, ya en entorno de producción, respondiendo a las peticiones de usuario.

Cuando un usuario inicia un *bot*, se lanza automáticamente el comando /start, lo que aprovecharemos para indicar al usuario la existencia de dos comandos adicionales (figura 6.23). Por un lado, el comando /main que lleva a la pantalla principal y, por otro, el comando /help que indica los permisos que tiene el usuario, y si no se está autorizado, indicará cómo solicitar permisos al *bot*. (ver figura 6.24)

Figura 6.23: Respuesta al comando /start.

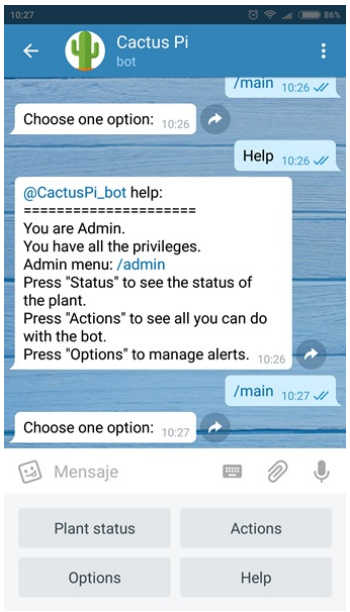

Figura 6.24: Vista de ayuda y menú principal del *bot*.

Como se ve en el menú principal, hay 4 posibles acciones para los usuarios: ”Plant status”, ”Actions”, ”Options” y ”Help”.

Si se pulsa sobre ”Plant status” el *bot* devuelve información básica sobre el estado de la planta.

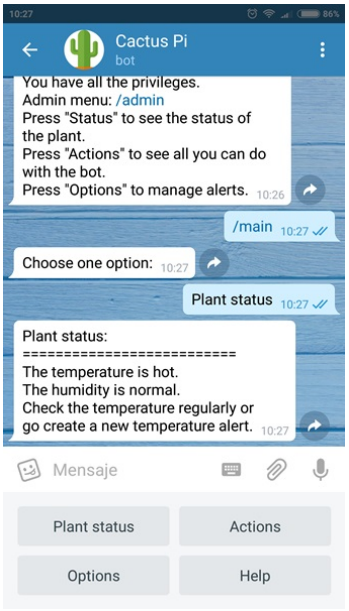

Figura 6.25: Respuesta del bot sobre el estado de la planta.

Las acciones disponibles a los usuarios con permisos, se pueden ver cuando el usuario pulsa sobre el botón "Actions".

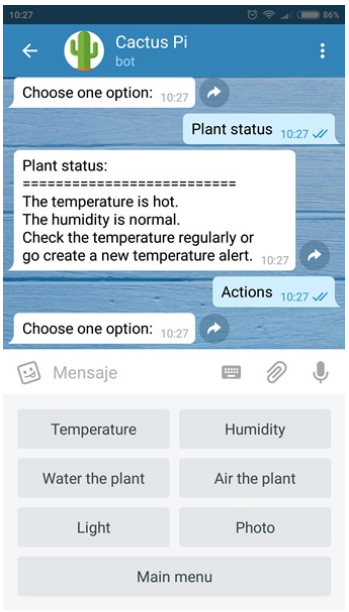

Figura 6.26: Menú de acciones disponibles en el bot.

Ahora, se va a ver cómo trabaja el *bot* en las diferentes acciones disponibles para el usuario.

El menú "Temperature" es siguiente:

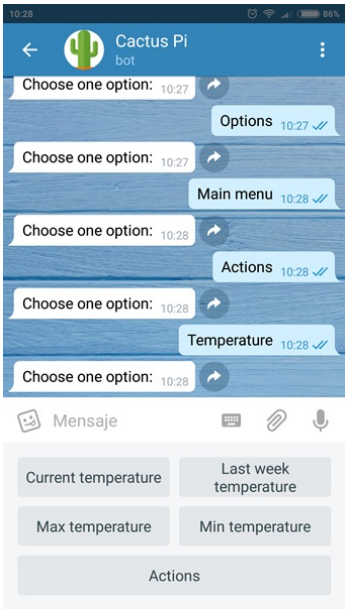

Figura 6.27: Respuesta del bot al comando "Temperature".

Se pueden consultar la temperatura actual, la temperatura máxima, la mínima, o los registros de temperatura de la última semana. En este caso se ve la solicitud de temperatura máxima:

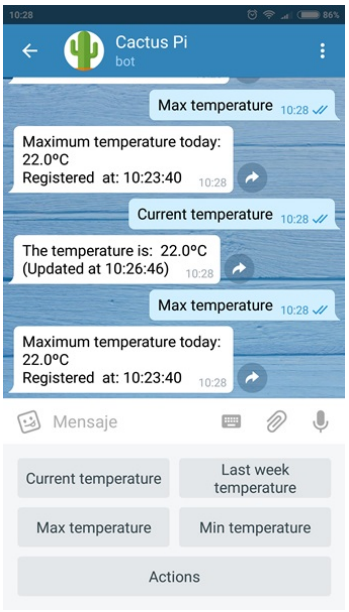

Figura 6.28: Respuesta del bot al comando "Max temperature".

Mismo caso para la opción de humedad.

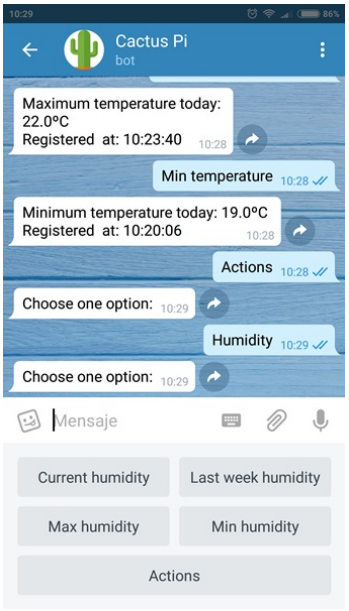

Figura 6.29: Menú de opciones sobre humedad en el bot

Se puede consultar la humedad actual, la humedad máxima, la mínima y las humedades registradas en los últimos 7 días. Para el ejemplo se muestra la humedad actual:

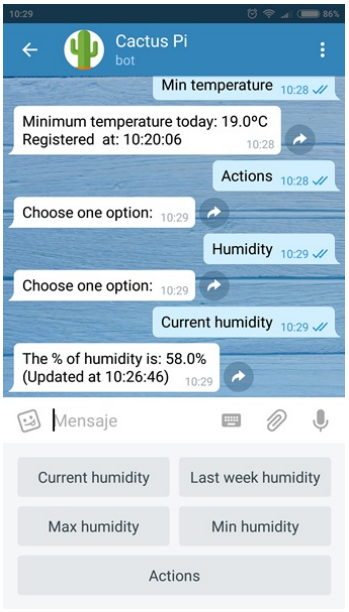

Figura 6.30: Respuesta del bot al comando "Current humidity".

En este otro ejemplo, se ve cómo se se solicita la humedad máxima:

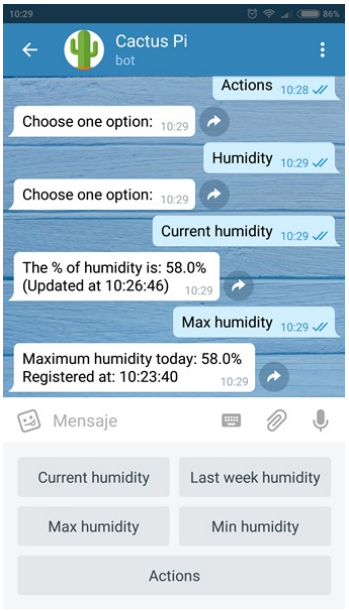

Figura 6.31: Respuesta del bot al comando "Max humidity".

En esta captura, se ve la opción de riego de la planta. Permite consultar el último riego de la planta, que indicará quién la ha regado y cuándo. Se puede realizar un bloqueo, para evitar que otros usuarios rieguen la planta de más, o desbloquearlo para volver a permitir el riego.

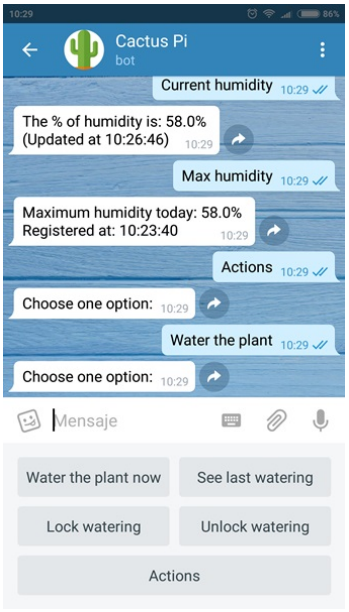

Figura 6.32: Menú de riego de la planta en el bot.

En este caso, se le solicita a **Cactus Pi** que riegue la planta. Si la orden se puede llevar a cabo, el *bot* indica que la planta ha sido regada.

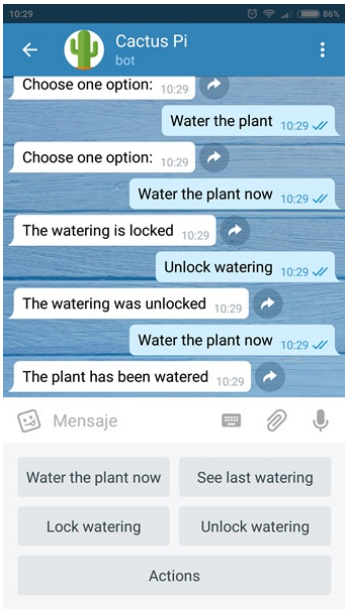

Figura 6.33: Solicitud de riego de la planta y respuesta del bot.

Como se ha indicado antes, se puede consultar la información relativa a la última vez que tuvo una acción de riego.

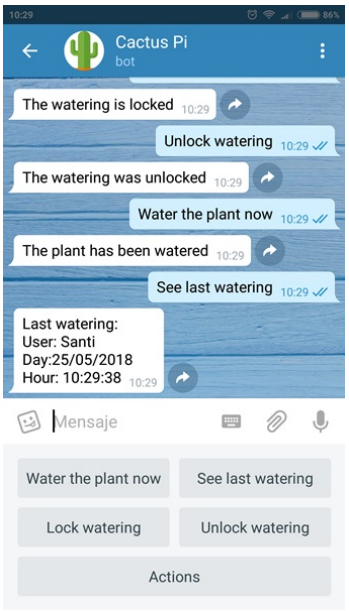

Figura 6.34: Respuesta del bot al comando "See last watering".

Volviendo al punto de la figura 6.26, y entrando en la opción "Air the plant", se puede controlar todo lo relativo a la ventilación del invernadero. En la siguiente captura, se puede ver cómo se usan todas las opciones disponibles.

Figura 6.35: Ejemplo de uso de todas las opciones del menú de ventilación.

A continuación se puede ver el mismo ejemplo anterior en el caso de pulsar sobre la opción ”Light”. En este caso, se cuenta con opciones para encender y apagar la luz y ver el estado de la misma.

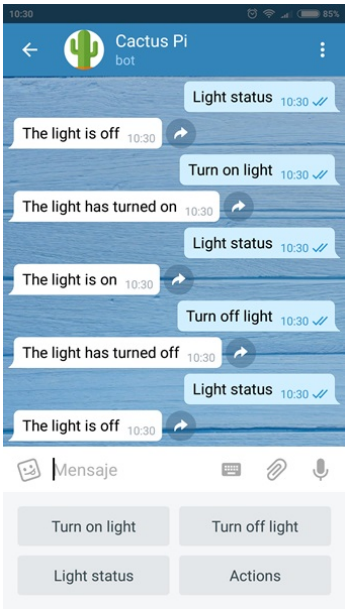

Figura 6.36: Ejemplo de uso de todas las opciones del menú de iluminación.

Y de idéntica forma, en el caso de pulsar sobre la opción ”Photo”. Las opciones disponibles son: tomar una fotografía de la planta y, para mejorar la iluminación, se permite también comprobar el estado de la luz, encender la luz, o apagarla.

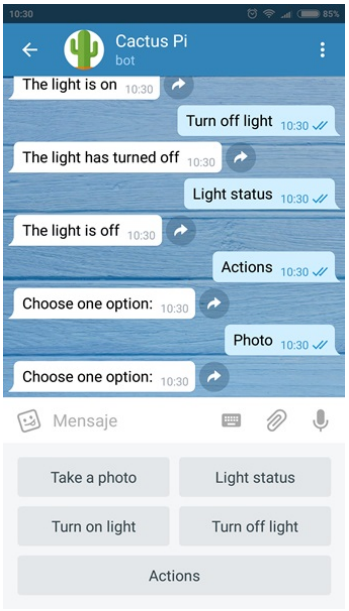

Figura 6.37: Opciones disponibles de sistema Cactus Pi para el comando "Photo".

En el menú de opciones, el usuario puede optar entre gestionar sus privilegios o las alertas.

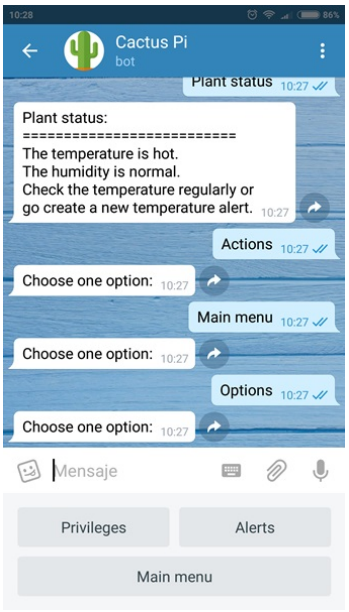

Figura 6.38: Menú de opciones disponibles.

La opción de privilegios habilita al usuario a pedir permiso de uso o solicitar un perfil con mayores privilegios.

Figura 6.39: Respuesta de Cactus Pi al comando "Privileges".

En el caso de las alertas, el *bot* permite ver las alertas que ha creado el usuario o configurar una nueva alerta.

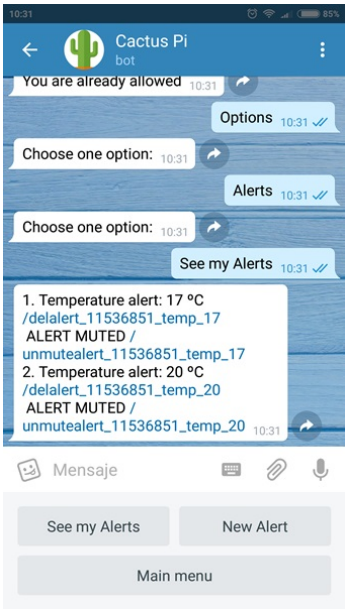

Figura 6.40: Respuesta de sistema Cactus Pi al comando "See my Alerts".

Cuando el usuario desea crear una nueva alerta, el *bot* pregunta de qué tipo de alerta se trata.

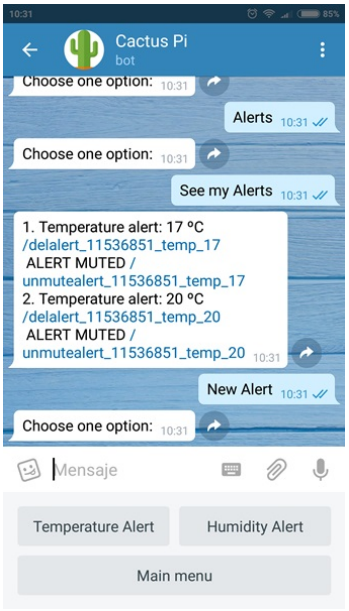

Figura 6.41: Creación de alerta en el sistema Cactus Pi.

En el ejemplo, se ve la creación de una alerta de humedad. Para ello, solicita al usuario un valor de humedad para el cual se va a lanzar la alerta.

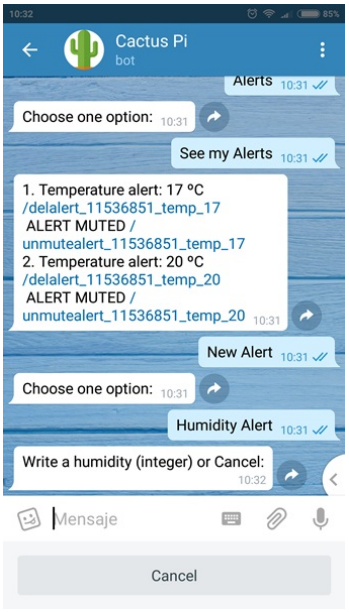

Figura 6.42: Creación de alerta de humedad en el sistema Cactus Pi.

El *bot* confirma la creación de la alerta.

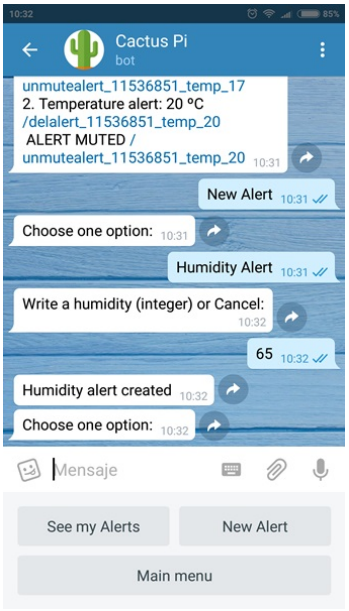

Figura 6.43: Confirmación de creación de alerta de humedad en el sistema Cactus Pi.

Y si se vuelve a consultar las alertas creadas por el usuario, se puede ver cómo aparece esa nueva alerta en el listado.

Todas las alertas se podrían borrar pulsando el comando /delalert que hay bajo cada alerta. Ese comando, se crea realizando la concatenación del número de usuario de `Telegram`, indicando si es una alerta de temperatura o humedad (temp, hum), y concatenando el valor de la alerta.

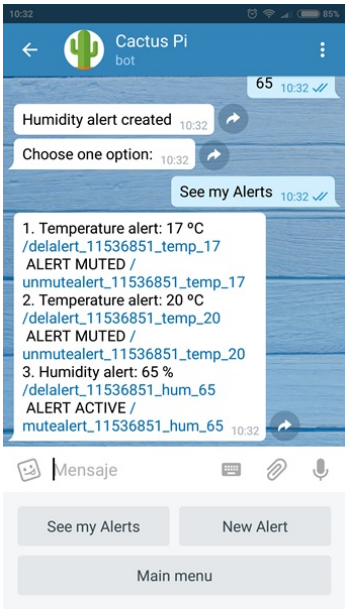

Figura 6.44: Respuesta de Cactus Pi al comando "See my Alerts".

Ahora, se ve la respuesta del *bot* cuando se le pide ayuda.

En este caso, el *bot* ofrece información sobre los permisos que tiene el usuario, y da información sobre las opciones disponibles al usuario.

Como vemos en el ejemplo, muestra la ayuda para un usuario tipo "Admin" y, por eso, le muestra todas las opciones disponibles. Si se tratase de un usuario sin privilegios, le indicaría cómo puede solicitarlos.

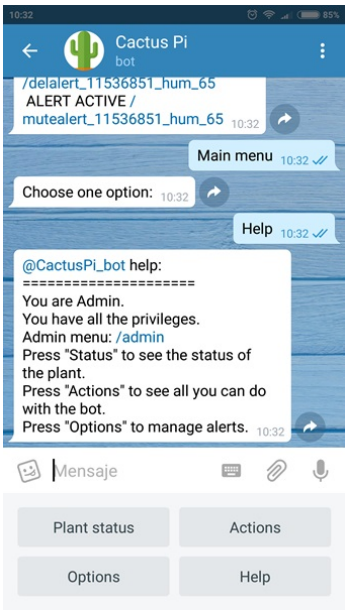

Figura 6.45: Respuesta de Cactus Pi al comando "Help".

Para los administradores, como se ha visto en la captura anterior, existe un menú privado que les permite administrar otros usuarios y ver algunas opciones extra.

En concreto, a continuación se ven las opciones para administrar usuarios, ver solicitudes de uso, opciones de administrador, y volver al menú principal.

Como es lógico, un usuario sin permisos de administrador, no puede acceder a este menú.

Figura 6.46: Respuesta de Cactus Pi al comando "/admin".

Si el administrador pulsa sobre "Manage users", se le permite visualizar los usuarios con permisos y los usuarios sin permisos. Ambas opciones listarán los usuarios que forman parte de estos conjuntos.

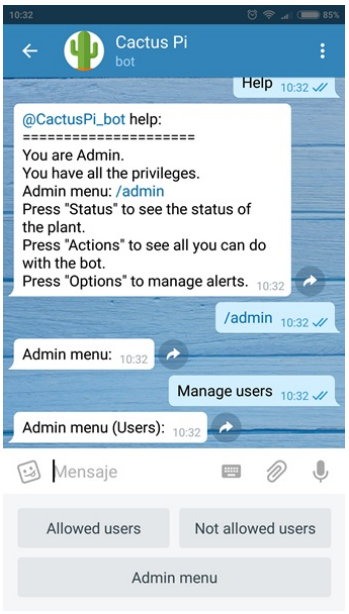

Figura 6.47: Respuesta de Cactus Pi al comando "Manage users".

En este caso, se pueden ver los usuarios con permisos sobre el sistema.

Como se puede ver, se indica la ID de usuario, el nombre registrado en `Telegram`, su alias (si lo ha configurado), el perfil de usuario, y un comando para poder editar el usuario.

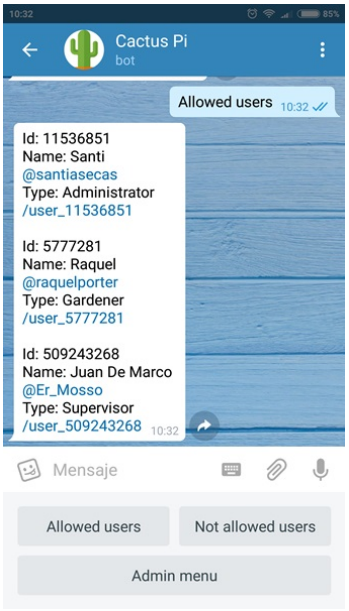

Figura 6.48: Respuesta de Cactus Pi al comando "Allowed users".

De igual forma, se pueden listar los usuarios que no tienen permisos. En este caso, no se muestra el tipo de perfil, obviamente.

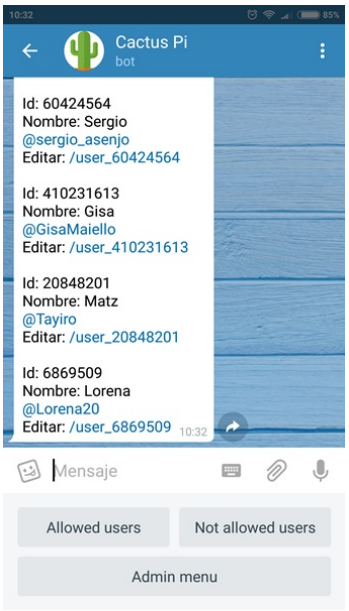

Figura 6.49: Respuesta de Cactus Pi al comando "Not allowed users".

En la siguiente captura, se muestra un listado de usuarios que han solicitado permisos para utilizar el *bot*. Se indica el día en que se ha solicitado los permisos, y se ofrece la opción de ignorar esa solicitud o de aceptarla.

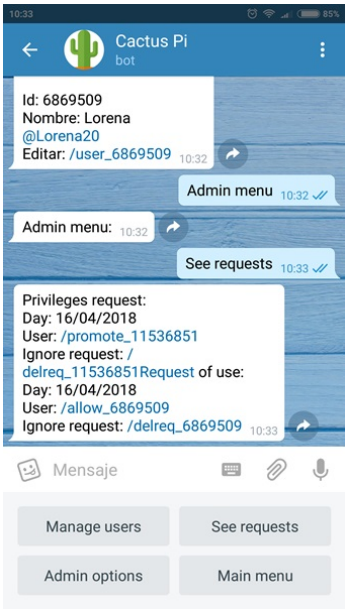

Figura 6.50: Respuesta de Cactus Pi al comando "See requests".

El usuario administrador, tiene permisos para ver las alertas creadas por el resto de usuarios y editarlas.

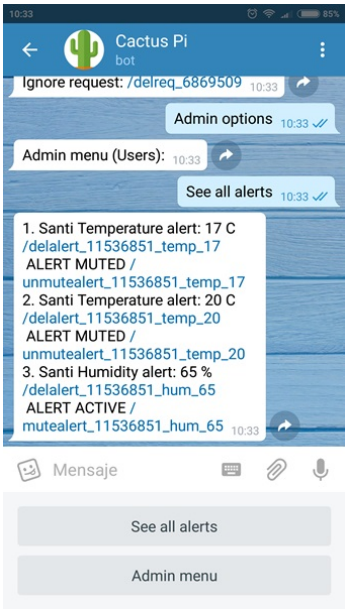

Figura 6.51: Respuesta de Cactus Pi al comando "See all alerts".

Volviendo atrás, recordemos que se podía editar un usuario. A continuación, se muestra un ejemplo. Si se pulsa sobre el comando "/user" de un usuario concreto, el sistema devuelve la información del usuario y permite promocionarlo a un perfil superior, o degradarlo a uno inferior. En el caso de editar un usuario con el perfil más bajo, la opción de degradarlo se sustituye por denegarle el permiso a utilizar **Cactus Pi**.

Figura 6.52: Edición de un usuario en Cactus Pi.

Con esto, se podría administrar los diferentes usuarios que utilizan el *bot*.

Cabe decir, que el usuario que crea el *bot*, es una especie de ”Superadministrador”. Un administrador normal puede administrar los perfiles de otros usuarios, salvo el de un ”Superadministrador”. Sin embargo, un ”Superadministrador” sí puede administrar el de un administrador normal.

CAPÍTULO 7
Conclusiones

Después de todo el trabajo realizado, queda plantearse qué ha ido bien y qué se podría mejorar si se volviese a afrontar un proyecto de este tipo.

En primer lugar, hay que tener en cuenta que es un trabajo un tanto ecléctico, puesto que se utilizan diferentes tecnologías y ha requerido investigar en diferentes disciplinas. Esto tiene su parte positiva, puesto que hace que sea muy dinámico enfrentarse a un proyecto de estas características, pero no se puede ahondar todo lo que uno quisiera en un campo en concreto.

7.1 Relación del trabajo desarrollado con los estudios cursados

Desde que decidí realizar los estudios de Grado en Ingeniería Informática, tenía claro que la rama que mayor interés me despertaba, era la de "Tecnologías de la información". En esta rama, se estudia cómo un ingeniero hace uso de las tecnologías para transmitir, almacenar y proteger la información.

Considero que en este proyecto se ponen de manifiesto muchos de los conocimientos adquiridos en los estudios cursados.

Para alcanzar con éxito este proyecto ha sido necesario aunar dos tecnologías que en principio no tienen por qué ir de la mano, y creo que ese es uno de los retos de los ingenieros que se enfrentan a problemas reales, hacer trabajar la tecnología para un objetivo concreto.

Por otro lado, tenemos la programación que, aunque sea de manera básica, todos los informáticos deberían de conocer y saber aplicar a proyectos diversos. Para mí ha sido un placer encontrarme con un lenguaje de programación que no conocía previamente, y aprender de manera rápida y sencilla la sintaxis gracias a la base que he adquirido en los estudios cursados.

Además, ha sido un reto volver a enfrentarme a la electrónica básica que estudiamos en los primeros cursos de la carrera. Es curioso comprobar cómo los conocimientos que pensaba que nunca usaría, se hacen necesarios y se afianzan cada vez que hay que afrontar problemas nuevos.

Una de las competencias que más he tenido que trabajar, por falta de hábito, ha consistido en la lectura paciente de manuales, antes de lanzarme a conectar y probar todo. De igual modo, he tenido que leer mucha información sobre la API de texttttTelegram para programar el *bot*. Creo que es importante tomarse tiempo para la lectura de manuales e información antes de comenzar un proyecto.

7.2 Qué ha ido bien

Uno de los objetivos secundarios al realizar este proyecto era aprender a realizar un *bot* desde cero y a usar `Raspberry Pi` como un centro de control domótico. Se puede decir que he aprendido las bases para la realización de *bots* más complejos, y que he aprendido cómo trabajar con la `Raspberry Pi`. Por lo tanto, la realización del proyecto ha sido muy positiva para mí.

En cuanto al alcance inicial que se propuso, también se puede considerar que se ha alcanzado, por lo que el proyecto habría sido terminado a tiempo y cubriendo los requisitos necesarios.

Una de las claves para haber conseguido llevar a cabo este proyecto, considero que sería la organización en el tiempo. Este proyecto se inició a principio de curso, cuando se acordó el mismo, y se ha ido realizando poco a poco y de manera muy escalada.

Esto ha beneficiado a la corrección de errores y el poder realizar numerosas pruebas con usuarios externos. Y esta creo que ha sido otra de las claves que han permitido llevarlo a cabo. La opinión de usuarios externos ha ayudado mucho a la creación de menús y funciones que tenía que cubrir el *bot*.

7.3 Puntos a mejorar

Uno de los errores que he detectado después de llevar a cabo este proyecto es no haber investigado previamente sobre el cuidado de plantas, y haber pasado directamente a la parte más técnica del proyecto.

Cuando afrontamos proyectos de este tipo, las personas con un perfil técnico, tendemos a preocuparnos por la parte técnica del proyecto y dejar de lado los aspectos fundamentales. En el caso de este proyecto, se dejaron de lado los aspectos sobre el cuidado de una planta. Fue ya avanzado el proyecto cuando leí que no es bueno que una planta reciba una corriente de aire directa, y por ello, tuve que modificar el planteamiento inicial en la parte de la ventilación, que originalmente ventilaba la planta de manera directa, para usar la ventilación de manera a extraer aire caliente del invernadero.

Además, se dio por hecho que todas las plantas se cuidan de la misma manera, lo cual es un grave error. Hay muchos tipos de plantas, y cada una tiene sus características propias. Por ejemplo, unas necesitan más agua en el riego que otras. Ha sido al final del proyecto, cuando he sido consciente de que se podría haber regulado, de una manera muy sencilla, la cantidad de agua que se vierte en cada riego simplemente controlando con una variable el tiempo de riego.

Además, he considerado las temperaturas y humedades de manera subjetiva como bajas, medias o altas, sin tener en cuenta el tipo de planta que se fuera a cuidar. Hay plantas para los que una temperatura es baja y esa misma temperatura puede ser alta para otra planta distinta.

A estos aspectos es a los que me refería al decir que había dejado de lado aspectos fundamentales. Creo que si se va a trabajar con plantas, aunque se trate de un proyecto enfocado a automatizar la monitorización y el cuidado de las mismas, primero hay que dedicarse al cuidado de plantas de forma manual, y ver qué necesitan y cómo hay que cuidarlas y, después de un tiempo, cuando se sepa cómo cuidarlas de una manera correcta, automatizar ese proceso usando la tecnología a nuestro alcance.

Pretender automatizar un proceso que no se conoce es un error, y esto es una lección que he aprendido gracias a este proyecto.

7.4 Líneas de trabajo futuras

Como líneas futuras de este proyecto, se pueden plantear diferentes alternativas.

Una posible opción sería desarrollar un sistema basado en **Cactus Pi** que ampliara el enfoque del mismo, para llevarlo a invernaderos y la agricultura en general. En este caso, habría que realizar numerosas mejoras de cara a ofrecer garantías al usuario final que, buscando lo mejor para su negocio, va a requerir los mejores productos y servicios.

Nos podemos hacer una idea de todo lo que habría que mejorar. Seguramente, habría que sustituir la `Raspberry Pi` por un servidor convencional, y `Telegram` por una aplicación nativa de terminal móvil o incluso web, con un sistema de seguridad mucho más avanzado. Además, habría que desarrollar muchas más funciones de las que hemos visto en este proyecto para monitorizar, entre otros, el pH de la tierra, humedades en diferentes puntos, contaminación atmosférica, etc.

Y por supuesto, habría que adaptar nuestro sistema a los sistemas de riego que estuvieran empleando, utilizar sensores distintos y de mejor calidad y precisión. En definitiva, habría que realizar mejoras en todo el sistema.

Una posible alternativa, sería un punto más cercano al que se encuentra el proyecto actualmente. Se podría comercializar un sistema de "mini invernadero" que ofreciese el sistema a clientes particulares que deseen un invernadero de bajo coste con opción de monitorización y actuación de sus plantas. Este producto, sería competencia directa del sistema que se ha visto en el capítulo 2 `AeroGarden`.

Esta alternativa, requeriría también de algunos cambios. Entre otros, habría que adaptar el *bot* en `Telegram`, para que cada usuario pudiera registrar su invernadero con algún tipo de identificador y facilitar esta tarea. La idea sería, realizar el mantenimiento del código fuente e ir realizando ampliaciones y mejoras para los usuarios, y que el sistema sea Plug&Play.

De cara a fidelizar a los usuarios y ofrecerles más soluciones a otras necesidades, se podrían crear otros sistemas de monitorización alternativos, por ejemplo para animales de compañía tales como peces, tortugas, pájaros, etc.

También sería posible crear paquetes con distintas configuraciones y precios, así como módulos que permitieran ampliar un sistema adquirido. Por ejemplo, si un usuario comprase un sistema **Cactus Pi** básico, y después quisiera controlar también el pH de la tierra, podría comprar el módulo e instalarlo él mismo.

Finalmente, la opción más obvia sería liberar el código en Internet con una licencia de código abierto, y presentar el proyecto para que posibles desarrolladores interesados lo continúen ampliando y mejorando. Podría mantenerse dentro de un sistema de control de versiones como `GitHub`, y moderar los cambios que proponga la comunidad.

No requeriría de ninguna modificación, pero sí de un trabajo de mejoras continuas para dinamizar el proyecto y hacer que otros se interesasen en el mismo.

Y basándonos en esta última alternativa, también se podría separar el código que se ha generado, para ofrecer el sistema de gestión de usuarios a otros desarrolladores que quieran utilizarlo en sus *bots*. Es útil para controlar quién tiene acceso al *bot*, y esto puede ser interesante para muchos desarrolladores, sin tener que ocuparse de desarrollarlo desde cero. Personalmente, es algo que utilizaré en los próximos *bots* que cree, sobre todo si se encuentran en fase de desarrollo o son para uso exclusivamente personal.

Bibliografía

[1] Proyecto Tomaatit. Regado automático usando Arduino.
https://blog.bricogeek.com/noticias/arduino/sistema-de-riego-automatico-casero-con-arduino/

[2] Sistema hidropónico AeroGarden.
https://www.aerogarden.com/

[3] 15 usos diferentes de Rasperry Pi.
https://computerhoy.com/noticias/hardware/15-usos-raspberry-pi-que-no-sabias-que-podias-darle-74905

[4] TeleMonBot: Bot de Telegram para monitorizar un servidor propio.
https://blog.desdelinux.net/monitorear-tu-servidor-telegram/

[5] DownTime Bot: Bot de Telegram para monitorizar una web propia.
https://wwwhatsnew.com/2018/01/01/downtime-bot-para-ser-alertados-via-telegram-cuando-una-web-propia-no-este-disponible/

[6] Kindle Robot, un bot de Telegram para enviar libros y documentos al Kindle.
https://wwwhatsnew.com/2018/08/20/kindle-robot-un-bot-de-telegram-para-enviar-libros-y-documentos-al-kindle/

[7] Bot de Telegram para consultar las condiciones climatológicas locales.
https://downtimebot.com/i-made-a-telegram-bot-that-tells-me-my-local-surf-conditions/

[8] Bot de Telegram para consultar las condiciones climatológicas locales.
https://domoticaintegrada.com/casa-domotica/

[9] Monitor de energía Smappee.
https://www.smappee.com/eu_es/home https://www.xataka.com/energia/un-mes-usando-smappee-el-gadget-para-conocer-al-instante-el-consumo-de-cada-electrodomestico

[10] Monitor de calidad del aire, temperatura, humedad, luminosidad y ruido.
https://domboo.es/producto/monitor-calidad-del-aire-temperatura-humedad-luminosidad-ruido-broadlink/

[11] Centro de control domótico.
https://www.zipato.com/product/zipatile-zbee

[12] Sensor de temperatura y humedad digital.
https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/#DHT11_un_unico_sensor_para_la_temperatura_y_humedad

[13] Instalación de librería AdafruitDHT para sensor DHT11 de temperatura y humedad.
http://www.internetdelascosas.cl/2017/05/19/raspberry-pi-conectando-un-sensor-de-temperatura-y-humedad-dht11/

[14] Módulo I2C LCD1602 sobre placa Sunfounder.
http://wiki.sunfounder.cc/index.php?title=I%C2%B2C_LCD1602

[15] Sistema operativo Raspbian para Raspberry Pi.
https://www.raspberrypi.org/downloads/raspbian/

[16] Lista actualizada de sistemas operativos para Raspberry Pi.
https://raspberryparatorpes.net/raspberry-pi-sistemas-operativos/

[17] Diferentes tutoriales sobre configuración de Raspberry Pi.
https://raspberryparatorpes.net

[18] Programa para copiar imágenes de sistemas operativos en tarjetas SD o memorias USB.
https://www.luisllamas.es/instalar-raspbian-en-raspberry-pi-con-etcher/

[19] Instalación de fswebcam.
http://ask.xmodulo.com/install-usb-webcam-raspberry-pi.html

[20] API del Bot para Telegram.
https://github.com/eternnoir/pyTelegramBotAPI.git

[21] Consejos sobre el cuidado de plantas de interior.
https://www.joseeljardinero.com/consejos-cuidar-plantas-de-interior/

[22] Listado de cámaras web compatibles con Raspberry Pi
https://elinux.org/RPi_USB_Webcams

APÉNDICE A
Código del bot

A.1 Fichero cactus_pi.py

```
import os
os.system('/usr/bin/python /home/pi/CactusPi/cactus_bot.py&')
os.system('/usr/bin/python /home/pi/CactusPi/cactus_sensor.py&')
```

A.2 Fichero cactus_bot.py

```
# -*- coding: utf-8 -*-
import telebot
import time
import os
import commands
import threading
import cactus_messages as cmsg
import cactus_functions as cfun
import cactus_menus as cmenu
import logging

logging.basicConfig(filename='cactus.log',level=logging.INFO)
# Chat_id of the bot admin
ADMIN = 11536851
# TOKEN provided by @BotFather
TOKEN = '408284487:AAG_RT1pLx-TC_vVu-T9KzYuHE1ONdIa0eo'
bot = telebot.TeleBot(TOKEN)
createAlertTemp = []
createAlertHum = []

def startBot():
        logging.info(time.strftime("%d/%m/%Y") + ' - bot started at: ' +
        ↪  time.strftime("%H:%M:%S"))
        print('\nBot running')
        msg = cmsg.msg
```

```
############################
#####     LISTENER    ######
############################

        def listener(messages):
                for m in messages:
                        chat_id = m.chat.id
                        #if m.content_type == 'text':
                                #print("[" + str(chat_id) + "]: " + m.text)
        bot.set_update_listener(listener)


############################
#####  USER HANDLERS  ######
############################

        @bot.message_handler(commands=['ping'])
        def command_ping(m):
                chat_id = m.chat.id
                bot.send_message(chat_id, 'pong')
                cmenu.main_menu(chat_id, bot)

        @bot.message_handler(commands=['start'])
        def command_start(m):
                chat_id = m.chat.id
                name = m.from_user.first_name
                userName = m.from_user.username

                if cfun.hasStarted(chat_id) == False:
                        cfun.register(chat_id, name, userName)
                        bot.send_message(ADMIN,
                         ↪  msg['user_msg']['new_user'])
                if cfun.isAllowed(chat_id) or chat_id == ADMIN:
                        bot.send_message(chat_id, msg['user_msg']['start']
                         ↪  + '\n\n' + msg['user_msg']['help'])
                else:
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['not_auth'])

        @bot.message_handler(commands=['stop'])
        def command_stop(m):
                chat_id = m.chat.id
                if cfun.hasStarted(chat_id):
                        cfun.unregister(chat_id)

        @bot.message_handler(commands=['main'])
        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
         ↪  'Main menu')
        def command_main(m):
                chat_id = m.chat.id
                cmenu.main_menu(chat_id, bot)
```

```
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Actions')
def command_actions(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                cmenu.user_menu_main(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Temperature')
def command_temp(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                cmenu.user_menu_temp(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Humidity')
def command_hum(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                cmenu.user_menu_hum(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Light')
def command_light(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                cmenu.user_menu_light(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Photo')
def command_photo(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                cmenu.user_menu_photo(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Current temperature')
```

```
def command_temp(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        hour, temp = cfun.getTemp()
                        bot.send_message(chat_id,
                        ↪ msg['user_msg']['temp'].format(str(temp),
                        ↪ hour))
                except:
                        bot.send_message(chat_id,
                        ↪ msg['user_msg']['temp_error'])
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Max temperature')
def command_maxTemp(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        today = time.strftime("%d/%m/%Y")
                        tempMax, hora, x, y =
                        ↪ cfun.getMaxMinTemp(today)
                        bot.send_message(chat_id,
                                                'Maximum
                                                ↪ temperature
                                                ↪ today:
                                                ↪ ' +
                                                ↪ str(tempMax)
                                                ↪ +
                                                ↪ 'ºC\nRegistered
                                                ↪ at: '
                                                ↪ +
                                                ↪ str(hora))
                except:
                        bot.send_message(chat_id,
                        ↪ msg['user_msg']['temp_error'])
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Min temperature')
def command_minTemp(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        today = time.strftime("%d/%m/%Y")
                        x, y, tempMin, hora =
                        ↪ cfun.getMaxMinTemp(today)
                        bot.send_message(chat_id,
```

```
                                                        'Minimum
                                                        ↪ temperature
                                                        ↪ today:
                                                        ↪ ' +
                                                        ↪ str(tempMin)
                                                        ↪ +
                                                        ↪ 'ºC\nRegistered
                                                        ↪ at: '
                                                        ↪ +
                                                        ↪ str(hora))
                        except:
                                bot.send_message(chat_id,
                                ↪ msg['user_msg']['temp_error'])
                else:
                        bot.send_message(chat_id,
                        ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Last week temperature')
def command_minTemp(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        lastWeek = cfun.getLastWeekTemp()
                        bot.send_message(chat_id,lastWeek)
                except:
                        bot.send_message(chat_id,
                        ↪ msg['user_msg']['temp_error'])
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Last week humidity')
def command_minTemp(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        lastWeek = cfun.getLastWeekHum()
                        bot.send_message(chat_id,lastWeek)
                except:
                        bot.send_message(chat_id,
                        ↪ msg['user_msg']['hum_error'])
        else:
                bot.send_message(chat_id,
                ↪ msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪ 'Current humidity')
def command_hum(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
```

```
                try:
                        hour, hum = cfun.getHum()
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['humidity'].format(str(hum),
                        ↪  hour))
                except:
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['hum_error'])
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Max humidity')
def command_maxHum(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        today = time.strftime("%d/%m/%Y")
                        humMax, hora, x, y =
                        ↪  cfun.getMaxMinHum(today)
                        bot.send_message(chat_id, 'Maximum humidity
                        ↪  today: ' + str(humMax) + '%\nRegistered
                        ↪  at: ' + str(hora))
                except:
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['hum_error'])
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Min humidity')
def command_minHum(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 0:
                try:
                        today = time.strftime("%d/%m/%Y")
                        x, y, humMin, hora =
                        ↪  cfun.getMaxMinHum(today)
                        bot.send_message(chat_id, 'Minimum humidity
                        ↪  today: ' + str(humMin) + '%\nRegistered
                        ↪  at: ' + str(hora))
                except:
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['hum_error'])
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])
```

```
# FOTO
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Take a photo')
def command_photo(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 1:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['wait_photo'])
                os.system('fswebcam -r 640x480 img/foto.jpg --title
                ↪  "Cactus Pi Bot"')
                photo = open('img/foto.jpg', 'rb')
                bot.send_photo(chat_id, photo)
                os.system('rm img/foto.jpg -f')
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(commands=['help'])
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Help')
def handle_message(m):
        chat_id = m.chat.id
        try:
                bot.send_message(chat_id, msg['user_msg']['user_' +
                ↪  str(cfun.getUserType(chat_id))])
        except:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['help_error'])

#Devuelve el estado general de la planta según la temperatura y
↪  humedad.
#Esta información la puede consultar cualquier tipo de usuario.
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Plant status')
def handle_message(m):
        res = msg['user_msg']['plant_status']
        chat_id = m.chat.id
        h, temp = cfun.getTemp()
        h, hum = cfun.getHum()
        status = 0
        if(temp < 10):
                status = 1
                res += msg['user_msg']['temp_cold']
                res += msg['user_msg']['should_cover']
        elif(temp > 10 and temp < 15):
                status = 2
                res += msg['user_msg']['temp_warm']
        else:
                status = 3
                res += msg['user_msg']['temp_hot']
        if(hum < 30):
```

```
                res += msg['user_msg']['hum_low']
                res += msg['user_msg']['should_water']
                if (status == 1):
                        res += msg['user_msg']['check_temphum']
                elif (status == 2):
                        res += msg['user_msg']['check_hum']
                else:
                        res += msg['user_msg']['check_temphum']

        elif(hum > 30 and hum < 70):
                res += msg['user_msg']['hum_normal']
                if (status == 1):
                        res += msg['user_msg']['check_temp']
                elif (status == 2):
                        res += msg['user_msg']['plant_ok']
                else:
                        res += msg['user_msg']['check_temp']
        else:
                res += msg['user_msg']['hum_high']
                if (status == 1):
                        res += msg['user_msg']['check_temphum']
                elif (status == 2):
                        res += msg['user_msg']['check_hum']
                else:
                        res += msg['user_msg']['check_temphum']
        bot.send_message(chat_id, res)

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Settings')
def handle_message(m):
        chat_id = m.chat.id
        bot.send_message(chat_id, msg['user_msg']['settings'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Options')
def handle_message(m):
        chat_id = m.chat.id
        cmenu.user_options(chat_id, bot)

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Privileges')
def handle_message(m):
        chat_id = m.chat.id
        cmenu.user_privileges(chat_id, bot)

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Ask for more privileges')
def handle_message(m):
        chat_id = m.chat.id
        today = time.strftime("%d/%m/%Y")
        userType = cfun.getUserType(chat_id)
        if userType == 3:
```

```
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['already_allowed'])
        elif userType == 2 or userType == 1:
                cfun.sendRequest(chat_id, today, 1)
                bot.send_message(ADMIN,
                 ↪  msg['admin_msg']['priv_sent'])
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['priv_sent'])
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['priv_a_sent'])

@bot.message_handler(commands=['request'])
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Send a request')
def handle_message(m):
        chat_id = m.chat.id
        today = time.strftime("%d/%m/%Y")
        if cfun.isAllowed(chat_id):
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['already_allowed'])
        elif(cfun.sendRequest(chat_id, today, 0)):
                bot.send_message(ADMIN,
                 ↪  msg['admin_msg']['req_sent'])
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['req_sent'])
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['req_a_sent'])



@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Turn on light')
def command_ligthOn(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 1:
                try:
                        if cfun.getLightStatus() == 'on':
                                bot.send_message(chat_id,
                                 ↪  msg['user_msg']['light_a_on'])
                        else:
                                cfun.setLightOn()
                                bot.send_message(chat_id,
                                 ↪  msg['user_msg']['light_on'])
                except:
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['light_error'])
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])
```

```
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Turn off light')
def command_ligthOff(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 1:
                try:
                        if cfun.getLightStatus() == 'off':
                                bot.send_message(chat_id,
                                ↪  msg['user_msg']['light_a_off'])
                        else:
                                cfun.setLightOff()
                                bot.send_message(chat_id,
                                ↪  msg['user_msg']['light_off'])
                except:
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['light_error'])
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Light status')
def command_getLigth(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 1:
                try:
                        ligth = cfun.getLightStatus()
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['light_is'] + ligth)
                except:
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['light_error'])
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Turn on fan')
def command_fanOn(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                try:
                        if cfun.getFanStatus() == 'on':
                                bot.send_message(chat_id, 'Fan is
                                ↪  already on')
                        else:
                                cfun.setFanOn()
                                bot.send_message(chat_id,
                                ↪  msg['user_msg']['fan_on'])
                except:
                        bot.send_message(chat_id,
                        ↪  msg['user_msg']['fan_error'])
```

```
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Turn off fan')
def command_fanOff(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                try:
                        if cfun.getFanStatus() == 'off':
                                bot.send_message(chat_id, 'Fan is
                                 ↪  already off')
                        else:
                                cfun.setFanOff()
                                bot.send_message(chat_id,
                                 ↪  msg['user_msg']['fan_off'])
                except:
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['fan_error'])
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Fan status')
def command_getFan(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                try:
                        fan = cfun.getFanStatus()
                        bot.send_message(chat_id, 'Fan is ' + fan)
                except:
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['fan_error'])
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Water the plant')
def command_water(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                cmenu.user_menu_water(chat_id, bot)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Water the plant now')
def command_water_now(m):
```

```
        chat_id = m.chat.id
        today = time.strftime("%d/%m/%Y")
        hour = time.strftime("%H:%M:%S")
        if cfun.getUserType(chat_id) > 1:
                if(cfun.waterPlant(chat_id,today,hour)):
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['plant_watered'])
                else:
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['water_locked'])
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'See last watering')
def command_water(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                lastWatering = cfun.lastWatering()
                bot.send_message(chat_id, lastWatering)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Lock watering')
def command_water(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 1:
                cfun.lockWatering()
                bot.send_message(chat_id, 'The watering was
                 ↪  locked')
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Unlock watering')
def command_water(m):
        chat_id = m.chat.id
        if cfun.getUserType(chat_id) > 1:
                cfun.unlockWatering()
                bot.send_message(chat_id, 'The watering was
                 ↪  unlocked')
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Air the plant')
def command_air(m):
```

```
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                cmenu.user_menu_air(chat_id, bot)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Alerts')
def command_alerts(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                cmenu.user_alerts(chat_id, bot)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'See my Alerts')
def command_showalerts(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                alerts = cfun.getAlerts(chat_id)
                if (alerts == ''):
                        bot.send_message(chat_id, 'No alerts to
                         ↪  show')
                else:
                        bot.send_message(chat_id, alerts)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'New Alert')
def command_newalert(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                cmenu.user_newalert(chat_id, bot)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
 ↪  'Temperature Alert')
def command_tempalert(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                createAlertTemp.append(chat_id)
                cmenu.user_temp_alert(chat_id, bot)
        else:
                bot.send_message(chat_id,
                 ↪  msg['user_msg']['not_auth'])
```

```
@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Humidity Alert')
def command_humalert(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                createAlertHum.append(chat_id)
                cmenu.user_hum_alert(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
↪  'Cancel')
def command_humalert(m):
        chat_id = m.chat.id
        if cfun.isAllowed(chat_id):
                if(chat_id in createAlertTemp):
                        createAlertTemp.remove(chat_id)
                if(chat_id in createAlertHum):
                        createAlertHum.remove(chat_id)
                cmenu.main_menu(chat_id, bot)
        else:
                bot.send_message(chat_id,
                ↪  msg['user_msg']['not_auth'])

@bot.message_handler(regexp="/delalert_")
def command_delalert(m):
        chat_id = m.chat.id
        text = str(m.text).split('_')
        cid = text[1]
        temp = hum = 0
        if(text[2] == 'temp'):
                temp = 1
        else:
                hum = 1
        value = text[3]
        if cfun.isAllowed(cid):
                cfun.deleteAlert(cid,temp,hum,value)
                bot.send_message(chat_id, 'Alert deleted')

@bot.message_handler(regexp="/mutealert_")
def command_mutealert(m):
        chat_id = m.chat.id
        text = str(m.text).split('_')
        cid = text[1]
        temp = hum = 0
        if(text[2] == 'temp'):
                temp = 1
        else:
                hum = 1
        value = text[3]
```

```
        if cfun.isAllowed(cid):
                cfun.muteAlert(cid,temp,hum,value)
                bot.send_message(chat_id, 'Alert muted')

@bot.message_handler(regexp="/unmutealert_")
def command_handler(m):
        chat_id = m.chat.id
        text = str(m.text).split('_')
        cid = text[1]
        temp = hum = 0
        if(text[2] == 'temp'):
                temp = 1
        else:
                hum = 1
        value = text[3]
        if cfun.isAllowed(cid):
                cfun.unmuteAlert(cid,temp,hum,value)
                bot.send_message(chat_id, 'Alert unmuted')

@bot.message_handler(regexp="/newalert_")
def command_newalert(m):
        chat_id = m.chat.id
        text = str(m.text).split('_')
        cid = text[1]
        temp = hum = 0
        if(text[2] == 'temp'):
                temp = 1
        else:
                hum = 1
        value = text[3]
        if cfun.isAllowed(cid):
                cfun.createAlert(cid,temp,hum,value)
                bot.send_message(chat_id, 'Alert created')

@bot.message_handler(regexp="^[0-9]{2}$")
def command_regexp(m):
        chat_id = m.chat.id
        value = str(m.text[0:2])
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                if(chat_id in createAlertTemp):
                        if(cfun.createAlert(chat_id,1,0,value)):
                                bot.send_message(ADMIN,
                                 ↪  'Temperature alert created')
                                createAlertTemp.remove(chat_id)
                                cmenu.user_alerts(chat_id, bot)
                elif(chat_id in createAlertHum):
                        if(cfun.createAlert(chat_id,0,1,value)):
                                bot.send_message(ADMIN, 'Humidity
                                 ↪  alert created')
                                createAlertHum.remove(chat_id)
                                cmenu.user_alerts(chat_id, bot)
```

```
############################
#####  ADMIN HANDLERS ######
############################

        @bot.message_handler(commands=['admin'])
        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
         ↪  'Admin menu')
        def command_admin(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        cmenu.admin_menu_main(chat_id, bot)

        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
         ↪  'Manage users')
        def command_admin(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        cmenu.admin_menu_users(chat_id, bot)

        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
         ↪  'Manage alerts')
        def command_admin(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        cmenu.admin_menu_messages(chat_id, bot)

        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
         ↪  'See all alerts')
        def command_showalerts(m):
                chat_id = m.chat.id
                if cfun.isAllowed(chat_id):
                        alerts = cfun.getAllAlerts()
                        if (alerts == ''):
                                bot.send_message(chat_id, 'No alerts to
                                 ↪  show')
                        else:
                                bot.send_message(chat_id, alerts)
                else:
                        bot.send_message(chat_id,
                         ↪  msg['user_msg']['not_auth'])

        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
         ↪  'Admin options')
        def handle_message(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        cmenu.admin_options(chat_id, bot)

        @bot.message_handler(regexp="/sendmsg ")
        def command_sendmsg(m):
                chat_id = m.chat.id
```

```
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                try:
                        message = str(m.text)[9:]
                        allowed = cfun.getIdAllowed()
                        for a in allowed:
                                if(a == ADMIN):
                                        bot.send_message(a,
                                         ↪  message)
                except:
                        bot.send_message(chat_id,
                         ↪  msg['admin_msg']['usr_problem'] + cid)

@bot.message_handler(regexp="/user_")
def command_user(m):
        chat_id = m.chat.id
        cid = str(m.text)[6:]
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                name = cfun.getUserName(cid)
                if cfun.getUserType(cid) == 0:
                        bot.send_message(chat_id, 'Name: ' + name +
                         ↪  '\nUser: ' + cid + '\n/allow_' + cid)
                elif cfun.getUserType(cid) == 1:
                        bot.send_message(chat_id, 'Name: ' + name +
                         ↪  '\nThe user is Supervisor\nPromote to
                         ↪  Gardener:\n/promote_' + cid + '\nRemove
                         ↪  permission:\n/deny_' + cid)
                elif cfun.getUserType(cid) == 2:
                        bot.send_message(chat_id, 'Name: ' + name +
                         ↪  '\nThe user is Gardener\nPromote to
                         ↪  Admin:\n/promote_' + cid + '\nDegrade
                         ↪  to Supervisor:\n/degrade_' + cid +
                         ↪  '\nRemove permission:\n/deny_' + cid)
                else:
                        if(cid != str(ADMIN)):
                                bot.send_message(chat_id, 'Name: '
                                 ↪  + name + '\nThe user is
                                 ↪  Admin\nDegrade to
                                 ↪  Gardener:\n/degrade_' + cid +
                                 ↪  '\nRemove permission:\n/deny_'
                                 ↪  + cid)
                        else:
                                bot.send_message(chat_id,
                                 ↪  'Superadmin can not be
                                 ↪  modified')

@bot.message_handler(regexp="/promote_")
def command_promote(m):
        chat_id = m.chat.id
        cid = str(m.text)[9:]
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                if(cid == str(ADMIN)):
```

```
                        bot.send_message(chat_id, 'Superadmin can
                        ↪ not be modified')
                else:
                        cfun.promote(cid)
                        bot.send_message(chat_id, 'The user was
                        ↪ promoted')

@bot.message_handler(regexp="/degrade_")
def command_degrade(m):
        chat_id = m.chat.id
        cid = str(m.text)[9:]
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                if(cid == str(ADMIN)):
                        bot.send_message(chat_id, 'Superadmin can
                        ↪ not be modified')
                else:
                        cfun.degrade(cid)
                        bot.send_message(chat_id, 'The user was
                        ↪ degraded')

@bot.message_handler(regexp="/allow_")
def command_allow(m):
        chat_id = m.chat.id
        cid = str(m.text)[7:]
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                try:
                        cfun.allow(cid)
                        bot.send_message(chat_id,
                        ↪ msg['admin_msg']['access_allowed'] +
                        ↪ cid)
                except:
                        bot.send_message(chat_id,
                        ↪ msg['admin_msg']['usr_problem'])

@bot.message_handler(regexp="/deny_")
def command_deny(m):
        chat_id = m.chat.id
        cid = str(m.text)[6:]
        if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                if cid == str(ADMIN):
                        bot.send_message(chat_id, 'Superadmin can
                        ↪ not be modified')
                elif cfun.isAllowed(cid):
                        cfun.deny(cid)
                        bot.send_message(chat_id,
                        ↪ msg['admin_msg']['access_denied'] +
                        ↪ cid)
                else:
                        bot.send_message(chat_id, ('The user {} was
                        ↪ not allowed'.format(cid)))
```

```
        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
        ↪  'Allowed users')
        def command_showallowed(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        allowed = cfun.getAllowed()
                        bot.send_message(chat_id, allowed)

        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
        ↪  'Not allowed users')
        def command_shownotallowed(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        notAllowed = cfun.notAllowed()
                        bot.send_message(chat_id, notAllowed)

        @bot.message_handler(func=lambda msg: msg.text.encode("utf-8") ==
        ↪  'See requests')
        def command_showreq(m):
                chat_id = m.chat.id
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        requests = cfun.showRequests()
                        bot.send_message(chat_id, requests)

        @bot.message_handler(regexp="/delreq_")
        def command_degrade(m):
                chat_id = m.chat.id
                cid = str(m.text)[8:]
                if chat_id == ADMIN or cfun.getUserType(chat_id) == 3:
                        cfun.deleteRequest(cid)
                        bot.send_message(chat_id, 'The request was
                        ↪  deleted')

        bot.polling(none_stop=True)

def checkAlerts():
        while True:
                #Se comprueba la base de datos
                alerts = cfun.getAllAlerts2()
                today = time.strftime("%d/%m/%Y")
                tmax,a,tmin,b = cfun.getMaxMinTemp(today)
                hmax,a,hmin,b = cfun.getMaxMinHum(today)
                resalerts = ''
                for a in alerts:
                        user  = a[0]
                        temp  = a[1]
                        hum   = a[2]
                        value = a[3]
                        mute  = a[4]
                        if(temp == 1 and mute == 0):
                                if(value <= tmax and value >= tmin):
```

```
                                        resalerts += 'Alerta de
                                         ↪  temperatura: ' + str(value) +
                                         ↪  'ºC\nMute Alert: /mutealert_' +
                                         ↪  str(user) + '_temp_' +
                                         ↪  str(int(value)) + '\n'
                                elif(hum == 1 and mute == 0):
                                        if(value <= hmax and value >= hmin):
                                                resalerts += 'Alerta de humedad: '
                                                 ↪  + str(value) + '%\nMute Alert:
                                                 ↪  /mutealert_' + str(user) +
                                                 ↪  '_hum_' + str(int(value)) +
                                                 ↪  '\n'
                        # Si hay alguna alerta activa, avisa al usuario
                        if(resalerts != ''):
                                bot.send_message(user, resalerts)
                        # Comprueba cada 10 minutos
                        time.sleep(600)

if __name__ == "__main__":
        try:
                thread1 = threading.Thread(target=startBot)
                thread2 = threading.Thread(target=checkAlerts)
                thread1.start()
                thread2.start()

        except Exception as err:
                logging.info(time.strftime("%d/%m/%Y") + ' - bot stoped at:
                 ↪  ' + time.strftime("%H:%M:%S"))
                logging.error(err)
                time.sleep(30)
                startBot()
```

A.3 Fichero cactus_functions.py

```
# -*- coding: utf-8 -*-
import time
import sqlite3
from gpiozero import LED
import cactus_messages as cmsg

led = LED(17)
ledStatus = 'off'
fan = LED(22)
fanStatus = 'off'
water = LED(25)

db = "cactuspi.sqlite3"
msg = cmsg.msg
```

```
############################
#####    SENSORES     ######
############################

#Devuelve la hora y la temperatura
def getTemp():
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select time,temp from current_temp_hum")
        res = cur.fetchone()
        cur.close()
        conn.commit()
        return res[0],res[1]

#Devuelve la humedad
def getHum():
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select time,hum from current_temp_hum")
        res = cur.fetchone()
        cur.close()
        conn.commit()
        return res[0],res[1]

#Devuelve la temperatura y la humedad
def getTempHum():
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select temp,hum from current_temp_hum")
        res = cur.fetchone()
        cur.close()
        conn.commit()
        return res[0],res[1]

#Devuelve las temperaturas máximas y mínimas
def getMaxMinTemp(day):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select temp_max,time_temp_max,temp_min,time_temp_min
        ↪  from hist_temp_hum where day = ?", [day])
        temp = cur.fetchone()
        if temp == None: temp=(None,None,None,None)
        cur.close()
        conn.commit()
        return temp[0],temp[1],temp[2],temp[3]

#Devuelve las humedades máximas y mínimas
def getMaxMinHum(day):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select hum_max,time_hum_max,hum_min,time_hum_min from
        ↪  hist_temp_hum where day = ?", [day])
```

```
        hum = cur.fetchone()
        if hum == None: hum=(None,None,None,None)
        cur.close()
        conn.commit()
        return hum[0],hum[1],hum[2],hum[3]

#Actualiza la temperatura máxima
def setMaxTemp(day,tempMax,hour):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update hist_temp_hum set temp_max=?,time_temp_max=?
        ↪  where day=?", [tempMax,hour,day])
        temp = cur.fetchone()
        cur.close()
        conn.commit()

#Actualiza la temperatura mínima
def setMinTemp(day,tempMin,hour):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update hist_temp_hum set temp_min=?,time_temp_min=?
        ↪  where day=?", [tempMin,hour,day])
        temp = cur.fetchone()
        cur.close()
        conn.commit()

#Actualiza la humedad máxima
def setMaxHum(day,humMax,hour):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update hist_temp_hum set hum_max=?,time_hum_max=?
        ↪  where day=?", [humMax,hour,day])
        temp = cur.fetchone()
        cur.close()
        conn.commit()

#Actualiza la humedad mínima
def setMinHum(day,humMin,hour):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update hist_temp_hum set hum_min=?,time_hum_min=?
        ↪  where day=?", [humMin,hour,day])
        temp = cur.fetchone()
        cur.close()
        conn.commit()

#Crea un nuevo histórico de temperatura y humedad
def setNewHistTempHum(today,temp,hum,hour):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
```

```
        cur.execute("insert into hist_temp_hum
        ↪  (day,temp_min,time_temp_min,temp_max,time_temp_max,hum_min,time_hum_min,hum_max
        ↪  values (?,?,?,?,?,?,?,?,?)",
        ↪  [today,temp,hour,temp,hour,hum,hour,hum,hour])
        cur.close()
        conn.commit()

#Devuelve las temperaturas de la última semana
def getLastWeekTemp():
        msg = ''
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select day,temp_min,temp_max from hist_temp_hum order
        ↪  by id desc limit 7")
        for i in range(7):
                day,t1,t2 = cur.fetchone()
                msg += str(day) + '\nTmin: ' + str(t1) + 'º' + ' Tmax: ' +
                ↪  str(t2) + 'º\n'
        cur.close()
        conn.commit()
        return msg

#Devuelve las humedades de la última semana
def getLastWeekHum():
        msg = ''
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select day,hum_min,hum_max from hist_temp_hum order by
        ↪  id desc limit 7")
        for i in range(7):
                day,t1,t2 = cur.fetchone()
                msg += str(day) + '\nHmin: ' + str(t1) + '%' + ' Hmax: ' +
                ↪  str(t2) + '%\n'
        cur.close()
        conn.commit()
        return msg

############################
#####   ACTUADORES    ######
############################

#Enceder luz
def setLightOn():
        global led, ledStatus
        led.on()
        ledStatus = 'on'

#Apagar luz
def setLightOff():
        global led, ledStatus
        led.off()
        ledStatus = 'off'
```

```
#Ver estado de la luz
def getLightStatus():
        return ledStatus

#Enceder ventilación
def setFanOn():
        global fan, fanStatus
        fan.on()
        fanStatus = 'on'

#Apagar ventilación
def setFanOff():
        global fan, fanStatus
        fan.off()
        fanStatus = 'off'

#Ver estado del ventilador
def getFanStatus():
        return fanStatus

#Regar
def waterPlant(cid,today,hour):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from water")
        wat = cur.fetchone()
        if wat == None:
                water.on()
                time.sleep(1)
                water.off()
                cur.execute("insert into water values
                 ↪  (?,?,?,?)",[cid,today,hour,0])
        else:
                if(wat[3] == 0):
                        water.on()
                        time.sleep(1)
                        water.off()
                        cur.execute("delete from water")
                        cur.execute("insert into water values
                         ↪  (?,?,?,?)",[cid,today,hour,0])
                else:
                        cur.close()
                        conn.commit()
                        return False
        cur.close()
        conn.commit()
        return True

#Devuelve la última vez que se regó
def lastWatering():
        conn = sqlite3.connect(db)
```

```
        cur=conn.cursor()
        cur.execute("select * from water")
        water = cur.fetchone()
        user = getUserName(water[0])
        day = water[1]
        hour = water[2]
        if water == None:
                res = 'There are not registers of watering'
        else:
                res = 'Last watering: \nUser: ' + user + '\nDay:' + day +
                 ↪  '\nHour: ' + hour
        cur.close()
        conn.commit()
        return res

def lockWatering():
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from water")
        water = cur.fetchone()
        if water == None:
                cur.execute("insert into water values
                 ↪  (?,?,?,?)",[0,'0','0',1])
        else:
                cur.execute("update water set locked = ?",[1])
        cur.close()
        conn.commit()

def unlockWatering():
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from water")
        water = cur.fetchone()
        if water == None:
                cur.execute("insert into water values
                 ↪  (?,?,?,?)",[0,'0','0',0])
        else:
                cur.execute("update water set locked = ?",[0])
        cur.close()
        conn.commit()

############################
#####    USUARIOS     ######
############################

def sendRequest(cid, today, priv):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from requests where cid = ?",[str(cid)])
        aut = cur.fetchone()
        if aut == None:
```

```
                cur.execute("insert into requests values
                 ↪  (?,?,?)",[cid,today,priv])
                cur.close()
                conn.commit()
                return True
        else:
                return False

def showRequests():
        try:
                conn = sqlite3.connect(db)
                cur=conn.cursor()
                cur.execute("select * from requests")
                req = ''
                requests = cur.fetchall()
                for r in requests:
                        priv = r[2]
                        if(priv == 1):
                                req += 'Privileges request:\nDay: ' +
                                 ↪  str(r[1]) + ' \nUser: /promote_'+
                                 ↪  str(r[0]) + '\nIgnore request:
                                 ↪  /delreq_' + str(r[0])
                        else:
                                req += 'Request of use:\nDay: ' + str(r[1])
                                 ↪  + ' \nUser: /allow_'+ str(r[0]) +
                                 ↪  '\nIgnore request: /delreq_' +
                                 ↪  str(r[0])
                cur.close()
                conn.commit()
                if(req == ''):
                        req = msg['admin_msg']['no_req']
                return req
        except:
                return msg['admin_msg']['no_req']

############################
#####      ADMIN      ######
############################

#Comprobar si un chat_id está autorizado
def isAllowed(cid):
        conn = sqlite3.connect(db)
        #Creamos el cursor
        cur=conn.cursor()
        cur.execute("select user_type from users where id=?",[str(cid)])
        aut = cur.fetchone()
        if(aut == None):
                return False
        elif (aut[0] >= 1):
                cur.close()
                return True
        else:
```

```
                cur.close()
                return False

# Comprobar si un chat_id ya había iniciado el bot
def hasStarted(cid):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from users where id = ?",[str(cid)])
        aut = cur.fetchone()
        if aut == None:
                cur.close()
                conn.commit()
                return False
        else:
                cur.close()
                conn.commit()
                return True

#Registrar un usuario en la base de datos
def register(cid, name, userName):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("insert into users values (?,?,?,?)",[cid, name,
        ↪  userName, 0])
        cur.close()
        conn.commit()

#Borrar un usuario de la base de datos
def unregister(cid):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("delete from users where id = ?", [cid])
        cur.close()
        conn.commit()

#Da permisos de supervisor a un usuario
def allow(cid):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update users set user_type=? where id=?",[1,cid])
        #Se comprueba si había un request y se elimina
        cur.execute("select cid from requests where cid = ?",[str(cid)])
        res = cur.fetchone()
        if(res != None and str(res[0]) == cid):
                cur.execute("delete from requests where cid =
                ↪  ?",[str(cid)])
        cur.close()
        conn.commit()

#Quita permisos a un usuario
def deny(cid):
        conn = sqlite3.connect(db)
```

```
        cur=conn.cursor()
        cur.execute("update users set user_type=? where id=?",[0,cid])
        cur.close()
        conn.commit()

#Promociona a un usuario a un nivel superior, si no es ya administrador.
def promote(cid):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select user_type from users where id=?",[cid])
        userType = cur.fetchone()[0]
        if (userType < 3):
                userType += 1
                cur.execute("update users set user_type=? where
                 ↪  id=?",[userType,cid])
        cur.close()
        conn.commit()

#Degrada a un usuario a un nivel inferior
def degrade(cid):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select user_type from users where id=?",[cid])
        userType = cur.fetchone()[0]
        if (userType > 0):
                userType -= 1
                cur.execute("update users set user_type=? where
                 ↪  id=?",[userType,cid])
        cur.close()
        conn.commit()

#Devuelve los usuarios con permisos
def getAllowed():
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select id,first_name,username,user_type from users
         ↪  where user_type > 0")
        allowed = ''
        for uid,nombre,username,usertype in cur.fetchall():
                allowed += 'Id: ' + str(uid) + '\nName: ' + str(nombre) +
                 ↪  '\n@' + str(username) + '\nType: ' +
                 ↪  msg['admin_msg']['user_'+str(usertype)+'_name'] +
                 ↪  '\n/user_'+str(uid) + '\n\n'
        cur.close()
        conn.commit()
        if(allowed == ''):
                allowed = 'There are not allowed users to show'
        return allowed

#Devuelve las ids de los usuarios con permisos
def getIdAllowed():
        conn = sqlite3.connect(db)
```

```
        cur=conn.cursor()
        cur.execute("select id from users where user_type > 0")
        allowed = []
        for cid in cur.fetchall():
                allowed.append(cid[0])
        cur.close()
        conn.commit()
        return allowed

#Devuelve los usuarios que han iniciado el bot, pero no tienen permisos
def notAllowed():
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select id,first_name,username from users where
         ↪  user_type = 0")
        notAllowed = ''
        for uid,name,username in cur.fetchall():
                notAllowed += 'Id: ' + str(uid) + '\nNombre: ' + str(name)
                 ↪  + '\n@' + str(username) + '\nEditar: /user_' + str(uid)
                 ↪  + '\n\n'
        cur.close()
        conn.commit()
        if(notAllowed == ''):
                notAllowed = 'There are not not-allowed-users to show'
        return notAllowed

#Devuelve el tipo de usuario, dado un chat_id
def getUserType(cid):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select user_type from users where id = ?" , [cid])
        userType = cur.fetchone()
        cur.close()
        conn.commit()
        return userType[0]

#Devuelve el nombre de un usuario
def getUserName(cid):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select first_name from users where id = ?" , [cid])
        name = cur.fetchone()
        cur.close()
        conn.commit()
        return name[0]

#Crea una alerta para el usuario cid
def createAlert(cid,temp,hum,value):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("insert into alerts values (?,?,?,?,?)",[cid, temp,
         ↪  hum, value,0])
```

```
        cur.close()
        conn.commit()
        return True

#Borra una alerta para el usuario
def deleteAlert(cid,temp,hum,value):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("delete from alerts where cid = ? and temp = ? and hum
         ↪  = ? and value = ?",[cid, temp, hum, value])
        cur.close()
        conn.commit()

#Devuelve las alertas creadas por el usuario cid
def getAlerts(cid):
        res = ''
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from alerts where cid = ?" , [cid])
        alerts = cur.fetchall()
        cont = 1
        for a in alerts:
                if(a[1] == 1):
                        res += str(cont) + '. Temperature alert: ' +
                         ↪  str(a[3]) + ' ºC' + '\n' + '/delalert_' +
                         ↪  str(cid) + '_temp_' +  str(a[3]) + '\n'
                        if(a[4] == 1):
                                res+= ' ALERT MUTED /unmutealert_' +
                                 ↪  str(cid) + '_temp_' +  str(a[3]) + '\n'
                        else:
                                res+= ' ALERT ACTIVE /mutealert_' +
                                 ↪  str(cid) + '_temp_' +  str(a[3]) + '\n'
                        cont += 1
                if(a[2] == 1):
                        res += str(cont) + '. Humidity alert: ' + str(a[3])
                         ↪  + ' %' + '\n' + '/delalert_' +   str(cid) +
                         ↪  '_hum_' +  str(a[3]) + '\n'
                        if(a[4] == 1):
                                res+= ' ALERT MUTED /unmutealert_' +
                                 ↪  str(cid) + '_hum_' +  str(a[3]) + '\n'
                        else:
                                res+= ' ALERT ACTIVE /mutealert_' +
                                 ↪  str(cid) + '_hum_' +  str(a[3]) + '\n'
                        cont += 1
        cur.close()
        conn.commit()
        return res

#Devuelve las alertas creadas por el usuario cid
def getAllAlerts():
        res = ''
        conn=sqlite3.connect(db)
```

```
        cur=conn.cursor()
        cur.execute("select * from alerts")
        alerts = cur.fetchall()
        cont = 1
        for a in alerts:
                user = getUserName(a[0])
                if(a[1] == 1):
                        res += str(cont) + '. ' + user + ' Temperature
                        ↪ alert: ' + str(a[3]) + ' C' + '\n' +
                        ↪ '/delalert_' +  str(a[0]) + '_temp_' +
                        ↪ str(a[3]) + '\n'
                        if(a[4] == 1):
                                res+= ' ALERT MUTED /unmutealert_' +
                                ↪ str(a[0]) + '_temp_' +  str(a[3]) +
                                ↪ '\n'
                        else:
                                res+= ' ALERT ACTIVE /mutealert_' +
                                ↪ str(a[0]) + '_temp_' +  str(a[3]) +
                                ↪ '\n'
                        cont += 1
                if(a[2] == 1):
                        res += str(cont) + '. ' + user + ' Humidity alert:
                        ↪ ' + str(a[3]) + ' %' + '\n' + '/delalert_' +
                        ↪ str(a[0]) + '_hum_' +  str(a[3]) + '\n'
                        if(a[4] == 1):
                                res+= ' ALERT MUTED /unmutealert_' +
                                ↪ str(a[0]) + '_hum_' +  str(a[3]) + '\n'
                        else:
                                res+= ' ALERT ACTIVE /mutealert_' +
                                ↪ str(a[0]) + '_hum_' +  str(a[3]) + '\n'
                        cont += 1
        cur.close()
        conn.commit()
        return res

def getAllAlerts2():
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("select * from alerts")
        alerts = cur.fetchall()
        cur.close()
        conn.commit()
        return alerts

#Silencia una alerta
def muteAlert(cid,temp,hum,value):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update alerts set mute = 1 where cid = ? and temp = ?
        ↪ and hum = ? and value = ?",[cid, temp, hum, value])
        cur.close()
        conn.commit()
```

```
#Vuelve a activar una alerta
def unmuteAlert(cid,temp,hum,value):
        conn=sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("update alerts set mute = 0 where cid = ? and temp = ?
        ↪  and hum = ? and value = ?",[cid, temp, hum, value])
        cur.close()
        conn.commit()

#Borra una solicitud de acceso al bot
def deleteRequest(cid):
        conn = sqlite3.connect(db)
        cur=conn.cursor()
        cur.execute("delete from requests where cid = ?",[str(cid)])
        cur.close()
        conn.commit()
```

A.4 Fichero cactus_messages.py

```
# -*- coding: utf-8 -*-

msg = {
        'user_msg' : {
                'start': 'Welcome to @CactusPi_bot \n\nThis bot helps you
                ↪  to take care of your favourite plant by using sensors
                ↪  and triggers.',
                'new_user': 'A new user has started the bot.',
                'not_auth':        'You are not allowed to use this bot',
                'not_admin': 'This action is not available to you',
                'already_allowed': 'You are already allowed',
                'req_sent': 'A request was sent to the ADMIN',
                'req_a_sent': 'You sent a request before',
                'priv_sent': 'A request for privileges was sent to the
                ↪  ADMIN',
                'priv_a_sent': 'You asked for privileges before',
                'temp':        'The temperature is:  {}ºC\n(Updated at
                ↪  {})',
                'temp_error': 'The temperature could not be retrieved',
                'humidity':        'The % of humidity is: {}%\n(Updated at
                ↪  {})',
                'humidity_error': 'The humidity could not be retrieved',
                'wait_photo': 'Wait while taking a photo',
                'fan_on': 'Fan on!',
                'fan_off': 'Fan off!',
                'fan_error': 'Fan status could not be retrieved',
                'light_error': 'The status of the light could not be
                ↪  retrieved',
                'light_on': 'The light has turned on',
                'light_off': 'The light has turned off',
                'light_a_on': 'The light is already on',
```

```
'light_a_off': 'The light is already off',
'light_is': 'The light is ',
'plant_status': 'Plant
↪  status:\n=========================\n',
'temp_cold': 'The temperature is cold.\n',
'temp_warm': 'The temperature is warm.\n',
'temp_hot': 'The temperature is hot.\n',
'should_water': 'You should water the plant.\n',
'should_air': 'You should air the greenhouse.\n',
'should_cover': 'You should cover the plant, or get it in a
↪  warmer place.\n',
'hum_low': 'The humidity is very low.\n',
'hum_normal': 'The humidity is normal.\n',
'hum_high': 'The humidity is very high.\n',
'plant_ok': 'Everything seems right!\n',
'check_temp': 'Check the temperature regularly or go create
↪  a new temperature alert.\n',
'check_hum': 'Check the humidity regularly or go create a
↪  new humidity alert.\n',
'check_temphum': 'Check the temperature and humidity
↪  regularly or go create both alerts.\n',
'plant_watered': 'The plant has been watered',
'water_problem': 'There was a problem watering the plant',
'water_locked': 'The watering is locked',
'msg_sent': 'The message has been sent to the admin',
'user_0': '@CactusPi_bot help:\n====================\nYou
↪  do not have privileges to use the bot.\nYou can ask for
↪  privileges by sending a /request',
'user_1': '@CactusPi_bot help:\n====================\nYou
↪  are a supervisor:\nYou can check the temperature and
↪  humidity.\nPress "Status" to see the status of the
↪  plant.\nPress "Actions" to see all you can do with the
↪  bot.\nPress "Options" to manage alerts or ask for more
↪  privileges.',
'user_2': '@CactusPi_bot help:\n====================\You
↪  are a gardener:\nYou have all the privileges with the
↪  plant.\nPress "Status" to see the status of the
↪  plant.\nPress "Actions" to see all you can do with the
↪  bot.\nPress "Options" to manage alerts or ask for more
↪  privileges.',
'user_3': '@CactusPi_bot help:\n====================\nYou
↪  are Admin.\nYou have all the privileges.\nAdmin menu:
↪  /admin\nPress "Status" to see the status of the
↪  plant.\nPress "Actions" to see all you can do with the
↪  bot.\nPress "Options" to manage alerts.',
'user_promote': 'The user was promoted',
'user_degrade': 'The user was degraded',
'settings': 'No settings to configure',
'write_msg': 'Write a message to send:',
'msg_error': 'There was a problem with the message. Try not
↪  sending emojis.',
```

```
                'help':         'Help of the
                ↪  bot:\n\n=====================\nYour user type is: ',
                'help_error': 'The help could not be retrieved'
        },

        'admin_msg' : {
                'admin_commands': 'The admin commands
                ↪  are:=========================\n',
                'req_sent': 'A user sent a request',
                'priv_sent': 'A user sent a request for privileges',
                'msg_sent':         'The message was sent to the user: ',
                'msg_received':         'A message has arrived',
                'msg_erased': 'The messages were erased',
                'no_msg': 'There are no messages to show',
                'no_req': 'There are no requests to show',
                'access_allowed': 'Access has been allowed to the user ',
                'access_allowed_received': 'You have been allowed to use
                ↪  the bot',
                'access_denied': 'Access has been denied to the user ',
                'access_denied_received': 'You have been denied to use the
                ↪  bot',
                'usr_problem': 'There was a problem with the user',
                'user_0_name': 'Not allowed',
                'user_1_name': 'Supervisor',
                'user_2_name': 'Gardener',
                'user_3_name': 'Administrator'
        }
}
```

A.5 Fichero cactus_menus.py

```
# -*- coding: utf-8 -*-
import telebot
from telebot import types

def main_menu(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Plant status')
        itembtn2 = types.KeyboardButton('Actions')
        itembtn3 = types.KeyboardButton('Options')
        itembtn4 = types.KeyboardButton('Help')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_options(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Privileges')
        itembtn2 = types.KeyboardButton('Alerts')
        itembtn3 = types.KeyboardButton('Main menu')
        markup.row(itembtn1, itembtn2)
```

```
        markup.row(itembtn3)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_privileges(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Send a request')
        itembtn2 = types.KeyboardButton('Ask for more privileges')
        itembtn3 = types.KeyboardButton('Options')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_alerts(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('See my Alerts')
        itembtn2 = types.KeyboardButton('New Alert')
        itembtn3 = types.KeyboardButton('Main menu')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_newalert(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Temperature Alert')
        itembtn2 = types.KeyboardButton('Humidity Alert')
        itembtn3 = types.KeyboardButton('Main menu')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_temp_alert(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Cancel')
        markup.row(itembtn1)
        bot.send_message(cid, "Write a temperature (integer) or Cancel:",
        ↪  reply_markup=markup)

def user_hum_alert(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Cancel')
        markup.row(itembtn1)
        bot.send_message(cid, "Write a humidity (integer) or Cancel:",
        ↪  reply_markup=markup)

def user_menu_main(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Temperature')
        itembtn2 = types.KeyboardButton('Humidity')
        itembtn3 = types.KeyboardButton('Water the plant')
        itembtn4 = types.KeyboardButton('Air the plant')
        itembtn5 = types.KeyboardButton('Light')
        itembtn6 = types.KeyboardButton('Photo')
```

```
        itembtn7 = types.KeyboardButton('Main menu')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        markup.row(itembtn5, itembtn6)
        markup.row(itembtn7)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_menu_temp(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Current temperature')
        itembtn2 = types.KeyboardButton('Last week temperature')
        itembtn3 = types.KeyboardButton('Max temperature')
        itembtn4 = types.KeyboardButton('Min temperature')
        itembtn5 = types.KeyboardButton('Actions')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        markup.row(itembtn5)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_menu_hum(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Current humidity')
        itembtn2 = types.KeyboardButton('Last week humidity')
        itembtn3 = types.KeyboardButton('Max humidity')
        itembtn4 = types.KeyboardButton('Min humidity')
        itembtn5 = types.KeyboardButton('Actions')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        markup.row(itembtn5)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_menu_light(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Turn on light')
        itembtn2 = types.KeyboardButton('Turn off light')
        itembtn3 = types.KeyboardButton('Light status')
        itembtn4 = types.KeyboardButton('Actions')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_menu_water(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Water the plant now')
        itembtn2 = types.KeyboardButton('See last watering')
        itembtn3 = types.KeyboardButton('Lock watering')
        itembtn4 = types.KeyboardButton('Unlock watering')
        itembtn5 = types.KeyboardButton('Actions')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        markup.row(itembtn5)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)
```

```
def user_menu_photo(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Take a photo')
        itembtn2 = types.KeyboardButton('Light status')
        itembtn3 = types.KeyboardButton('Turn on light')
        itembtn4 = types.KeyboardButton('Turn off light')
        itembtn5 = types.KeyboardButton('Actions')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        markup.row(itembtn5)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

def user_menu_air(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Turn on fan')
        itembtn2 = types.KeyboardButton('Turn off fan')
        itembtn3 = types.KeyboardButton('Fan status')
        itembtn4 = types.KeyboardButton('Actions')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3, itembtn4)
        bot.send_message(cid, "Choose one option:", reply_markup=markup)

############################
#####   ADMIN MENUS   ######
############################

def admin_menu_main(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Manage users')
        itembtn2 = types.KeyboardButton('See requests')
        itembtn3 = types.KeyboardButton('Admin options')
        itembtn4 = types.KeyboardButton('Main menu')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3,itembtn4)
        bot.send_message(cid, "Admin menu:", reply_markup=markup)

def admin_menu_users(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('Allowed users')
        itembtn2 = types.KeyboardButton('Not allowed users')
        itembtn3 = types.KeyboardButton('Admin menu')
        markup.row(itembtn1, itembtn2)
        markup.row(itembtn3)
        bot.send_message(cid, "Admin menu (Users):", reply_markup=markup)

def admin_options(cid, bot):
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        itembtn1 = types.KeyboardButton('See all alerts')
        itembtn2 = types.KeyboardButton('Admin menu')
        markup.row(itembtn1)
```

```
        markup.row(itembtn2)
        bot.send_message(cid, "Admin menu (Users):", reply_markup=markup)
```

A.6 Fichero cactus_sensor.py

```
import sys
import time
import sqlite3
import cactus_functions as cfun
import Adafruit_DHT

def measureTempHum():
        print('\nSensor running')
        # Configuracion del tipo de sensor DHT
        sensor = Adafruit_DHT.DHT11
        # Configuracion del puerto GPIO al cual esta conectado  (GPIO 23)
        pin = 23

        # Intenta ejecutar las siguientes instrucciones, si falla va a la
        ↪  instruccion except
        try:
                # Ciclo principal infinito
                while True:
                        # Obtiene la humedad y la temperatura desde el
                        ↪  sensor
                        humidity, temperature =
                        ↪  Adafruit_DHT.read_retry(sensor, pin)
                        hour = time.strftime("%H:%M:%S")
                        # Y se almacena en la base de datos
                        conn = sqlite3.connect("cactuspi.sqlite3")
                        cur=conn.cursor()
                        cur.execute("DROP TABLE IF EXISTS
                        ↪  Current_Temp_Hum")
                        cur.execute("CREATE TABLE Current_Temp_Hum (time
                        ↪  varchar(25), temp REAL, hum REAL)")
                        cur.execute("insert into Current_Temp_Hum values
                        ↪  (?,?,?)",[hour,temperature,humidity])
                        #Se recuperan los maximos y minimos del dia actual
                        today=time.strftime("%d/%m/%Y")
                        tempMax,x,tempMin,y=cfun.getMaxMinTemp(today)
                        humMax,x,humMin,y=cfun.getMaxMinHum(today)
                        cur.close()
                        conn.commit()
                        #Si no habia registros, se crea uno nuevo
                        if(tempMax == None):
                        ↪  cfun.setNewHistTempHum(today,temperature,humidity,hour)
                        #Y se actualizan si han variado
                        if(temperature > tempMax):
                        ↪  cfun.setMaxTemp(today,temperature,hour)
                        if(temperature < tempMin):
                        ↪  cfun.setMinTemp(today,temperature,hour)
```

```
                        if(humidity > humMax):
                         ↪  cfun.setMaxHum(today,humidity,hour)
                        if(humidity < humMin):
                         ↪  cfun.setMinHum(today,humidity,hour)

                        # Y espera 5 minutos
                        time.sleep(300)

        # Se ejecuta en caso de que falle alguna instruccion dentro del try
        except Exception,e:
                # Imprime en pantalla el error e
                print('There was a problem retrieving the temperature and
                 ↪  humidity')
                print str(e)

if __name__ == "__main__":
  measureTempHum()
```

Glosario

Android Sistema operativo, basado en Linux, desarrollado para dispositivos móviles. 14

API Por sus siglas en inglés, Application Programming Interfaces (Interfaces de programación de aplicaciones). Una API es una especificación formal sobre cómo un módulo de un software se comunica o interactúa con otro. 9, 12, 17, 19, 31–34, 38

bot Aféresis de robot. Entendido como un programa informático que realiza tareas para los seres humanos u otros programas informáticos. 2, 3, 5, 9–12, 15, 17, 19–23, 29, 31–34, 37–43, 49–59, 62, 66–69, 71, 73, 76–78

CPU Unidad Central de Procesamiento. Hardware en un ordenador que interpreta las instrucciones de un programa. 10

dirección IP Número que identifica a un dispositivo dentro de una red informática. 38, 40

GPIO GPIO (General Purpose Input/Output, Entrada/Salida de Propósito General) es un pin genérico en un chip, cuyo comportamiento se puede programar por el usuario. 2, 19, 20, 24, 30, 31, 37, 39

iOS Sistema operativo desarrollado por la empresa Apple, originalmente para el iPhone y actualmente en uso para un gran número de dispositivos móviles, tales como el iPad. 14

IoT Por sus siglas en inglés, Internet of Things (Internet de las cosas). Se refiere a la interconexión digital de cualquier dispositivo cotidiano con internet. 1, 15

JSON JSON, acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. 19, 34

LCD Sigla del inglés Liquid Cristal Display, "representación visual por cristal líquido", sistema que utilizan determinadas pantallas electrónicas para mostrar información visual. 16, 29, 30

RAM Memoria de acceso aleatorio. Es la memoria de trabajo principal de un ordenador. 10

script Programa de ordenador, generalmente simple, que se encarga de realizar una tarea como interactuar con el sistema operativo, con el usuario o combinar distintos componentes informáticos. 10, 47, 57

URL Localizador Uniforme de Recursos. Dirección que cumple un estándar y que designa recursos dentro de una red, como puede ser Internet. 11

USB El Bus Universal en Serie (BUS) (en inglés: Universal Serial Bus (USB)), es un bus de comunicaciones que sigue un estándar que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos. 8, 20, 24, 29, 39, 41, 47