UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Sistema Domótico Inteligente
Documentación Técnica

Presentado por David Colmenero Guerra
en Universidad de Burgos — 13 de enero de 2021
Tutor: Álvar Arnaiz-González
Tutor: Alejandro Merino Gómez

Índice general

Índice de figuras

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En cualquier proyecto de este tipo es relevante incluir cierta información sobre la planificación, viabilidad y análisis de costes en el desarrollo del mismo con la finalidad de conseguir una visión de cómo ha ido evolucionando el proyecto y los ciclos de trabajo que se han llevado a cabo.

He de señalar que en la última tercera parte de los sprints, no siempre he contado con una línea de datos con la que poder replicar datos en GitHub, por lo que se han generado menos commits de los que se hubiera hecho si hubiera contado con medios a mi alcance. En cualquier caso, se ha intentado maximizar el número de puntos de salvado para que quedase constancia de la evolución del proyecto. Además, al redactar el proyecto en LaTeX desde la plataforma de Overleaf no refleja el tiempo real invertido puesto que no se realizan cambios en el repositorio de forma automática, hasta que compilo, descargo y actualizo el repositorio manualmente.

En el primer apartado trataremos la **planificación temporal**, donde podremos ver como han evolucionado los tiempos de trabajo durante las semanas en las que se ha trabajado en el proyecto. En el segundo apartado se reflejará un **estudio de viabilidad** sobre el proyecto, que a su vez incluye dos partes, la parte económica y la parte legal.

A.2. Planificación temporal

Al comenzar el proyecto se determinó que se utilizaría una metodología ágil para hacer el desarrollo del proyecto pero esta decisión ha terminado

siendo un hándicap producido por la falta de documentación y la dudosa credibilidad de muchas páginas web y documentación que he encontrado, y se ha traducido en una gran inversión en tiempo.

Ante todo, se ha intentado seguir un mínimo de pautas **Scrum** [17] pese a no existir un grupo de trabajo real con unas tareas diarias y roles definidos:

- Las tareas fueron siempre semanales en forma de «sprints».
- Al finalizar el sprint se hace la entrega del trabajo elaborado en la semana y se determinan las próximas tareas.
- Tras determinar las tareas se definen los «milestones» y los «issues».
- Para hacer el seguimiento de las tareas se ha utilizado en tablero **Kanban** de **ZenHub**.
- Tras finalizar el sprint se puede comprobar el trabajo mediante los *gráficos burndown*.

Las reuniones de trabajo han sido consensuadas y planificadas para realizarlas los jueves de cada semana, con el siguiente resultado:

Sprint 00 - 01/10/2020 - 08/10/2020

En la primera reunión nos reunimos Álvar Arnaiz González y yo. En ella hice la propuesta de temática del TFG y, además expuse diferentes ideas y enfoques sobre el proyecto en la que se determinó a grandes rasgos la posible viabilidad del proyecto. Los objetivos de este «sprint» introductorio fueron la búsqueda de repositorios y otros proyectos para tomar ideas o si era posible hacer algún fork. También, el estudio la estructura del proyecto y herramientas software e interfaces a utilizar.

Sprint 01 - 09/10/2020 - 15/10/2020

Tras determinar el enfoque final de proyecto, se consensuó que los objetivos de este sprint fueron la búsqueda de información relevante, otros proyectos y recursos que puedan servir de utilidad como pueden ser APIS. Se buscaron algunas páginas web para realizar pruebas de extracción de datos con «web scraping» [1] y se descompusieron las tareas en unidades pequeñas de trabajo para poder afrontarlas durante sprints.

[1] Se trata brevemente en el punto 4 de la memoria

En este sprint también generé el repositorio en GitHub para poder trabajar contra él y podemos ver las líneas de código utilizadas en el primer *issue*.

Sprint 02 - 16/10/2020 - 22/10/2020

Los objetivos de este sprint fueron el buscar repositorios, APIS, servicios y tecnologías con las que darle forma al proyecto. Se decidió generar algún tipo de aplicación desde la que poder controlar la instalación haciéndola más amigable y con mejor capacidad de interacción. Se investigan opciones.

En el issue 2 se hizo una búsqueda por la web para buscar repositorios y proyectos sobre domótica.

En el issue 3 buscamos algunas APIS para orientar el proyecto hacia el uso de APIS destinadas a ofrecer información que pueda servirnos y se realizaron algunas pruebas.

En el issue 4 se realizó un pequeño estudio de las tecnologías que podríamos utilizar, desde el procesador de textos, que finalmente se utilizó LaTeX, como la revisión de las normativas electrotécnicas que debía seguir en el proyecto y también se determinó la utilización de Telegram para interactuar con el Sistema Domótico.

Sprint 03 - 23/10/2020 - 29/10/2020

Los objetivos de este sprint fueron determinar los componentes hardware necesarios para completar el proyecto y se realiza la compra de material. En todos los «issues» del «milestone 3» podemos ver múltiples enlaces, comparativas, justificaciones e imágenes en las que se ha basado toda la instalación física posterior.

Sprint 04 - 30/10/2020 - 05/10/2020

Se incorpora D.Alejandro Merino Gómez al que se le presenta el proyecto y se le concede acceso a los repositorios para que pueda co-tutorizar el proyecto.

Los objetivos de este «milestone» fueron todos de la parte física de la instalación: Podemos ver en el issue 15 que finalicé la tirada de cable junto a un diagrama explicativo del funcionamiento de un pulsador de 3 posiciones (Posición 1, Reposo y Posición 2. Teniendo en cuenta que las dos posiciones que dan continuidad al circuito tienen invertida la polaridad).

También vemos en el issue 16 la presentación de la Raspberry Pi que utilizaremos, los conectores del tipo JST-XH que se han utilizado en el crimpado de los cables electrónicos, los cables finalizados con sus conectores y la disposición final de la Raspberry Pi en su ubicación final de la instalación.

Sprint 05 - 06/11/2020 - 12/11/2020

Se repasan los cambios propuestos y se incluyen otros nuevos sobre la parte física de la instalación. Se hace el seguimiento de la instalación y se comentan las fotos y el estado de la instalación. Los objetivos de este sprint son todos aquellos que sean necesarios para la adecuación del software básico para comenzar el proyecto como la actualización del sistema operativo o instalación de software adicional. Además se deben valorar opciones para controlar los GPIO.

Vemos en los issues 21 y 19 las configuraciones básicas realizadas a nuestro Raspbian. Para explicar este proceso grabé dos vídeos, titulados Primeros Pasos Raspberry Pi y Actualización de Raspberry Pi y configuración básica, respectivamente.

Sprint 06 - 13/11/2020 - 19/11/2020

Se revisan los puntos anteriores y se compone un tablero de pruebas de forma que se puede controlar el encendido de una bombilla desde nuestra Raspberry Pi. También se graba un vídeo explicativo y se diseñan unos diagramas explicativos.

Podemos ver en el issue 22 que se presenta un diagrama con la lista de los componentes que vamos a utilizar en el tablero de pruebas y un diagrama de como se compondrá el tablero. En este issue también grabé un vídeo para explicar el tablero de pruebas y decidí utilizar un efecto más conservador de cara al final del proyecto.

En el issue 23 he explicado como se controlan los GPIO de nuestra Raspberry Pi desde su distribución Linux [2] desde Bash y desde Python.

Sprint 07 - 20/11/2020 - 26/11/2020

Se realiza una investigación sobre como se puede implantar el bot en nuestro proyecto, se realiza el primer código de pruebas y se presentan las primeras pruebas con el bot. Podemos ver en el issue 13 un resumen de las

[2] Raspbery pi (Raspbian OS).

pruebas que estuve realizando con los diferentes teclados de que dispone Telegram. También, se proponen correcciones en la redacción.

Sprint 08 - 27/11/2020 - 03/12/2020

En el milestone 8, podemos ver que se incluyen varios issues:

De esta manera, se genera la primera automatización completa de la parte automática del proyecto utilizando CRON y lo pasamos del entorno de pruebas a un entorno de producción en fase experimental. En este momento únicamente tenemos control sobre la máquina de forma física o por VNC. Por último, se presentan las primeras funcionalidades del bot, se proponen cambios y mejoras.

Sprint 09 - 04/12/2020 - 10/12/2020

En el milestone 9 se proponen revisan las funcionalidades del Bot [12] de Telegram [13] que utilizaremos, se proponen cambios y mejoras en el código preexistente en fase de pruebas.

Finalmente se integra un bot de Telegram completamente funcional desde el que podremos lanzar órdenes a nuestro sistema domótico con los scripts existentes.

Sprint 10 - 11/12/2020 - 17/12/2020

En el milestone 10 se ha aprovechado para optimizar las rutas de los archivos a los que se invoca desde el código en ejecución y se han modificado algunas salidas para que tengan un aspecto visual más atractivo, incluyendo tablas, emoticonos desde Unicode [18].

Se comprueban las funcionalidades proponiendo cambios funcionales y de estilos de forma que se pueda interactuar contra el bot y se muestren los mensajes en un formato más atractivo. También se propone unificar las rutas de trabajo de los diferentes scripts.

Sprint 11 - 18/12/2020 - 24/12/2020

Se propone dividir el código por funcionalidades (obtención de información de la web, y por otro lado, modificación de archivos del sistema y se proponen cambios en la redacción a implementar en el sprint 12.

Finalmente, se ha conseguido generar la división del código en 3 archivos principales:

De esta manera conseguimos minimizar las comunicaciones con el exterior y poder relanzar el código separado por funcionalidad lógica de forma que podamos relanzarlo de forma aislada en caso de necesitarlo. Un ejemplo puede ser cuando se modifica la hora de subida de las persianas, el módulo de generación de CRON leería el archivo de parámetros personalizados y generaría la configuración a partir de éstos.

Sprint 12 - 25/12/2020 - 31/01/2021

Se finaliza la redacción de la memoria, antes de la revisión final del texto, y se comienza con los anexos. Además se produce depurado del código y se utiliza SonarCloud [4].

Sprint 13 - 01/12/2020 - 07/01/2021

Redacción de los anexos, corrección de la memoria y limpieza de código. Se ha proporcionado la primera Release y asignado la licencia del proyecto.

Sprint 14 - 08/01/2020 - 14/01/2020

Corrección de los anexos y grabado de vídeos.

Sprint 15 - 15/01/2020 - 20/01/2020

Edición de vídeos y entrega del proyecto.

A.3. Estudio de viabilidad

El estudio de viabilidad es inherente a cualquier proyecto y determina si éste puede ser aplicable o no, y también sirve para aclarar algunos puntos importantes para ayudar a la toma de decisiones ejecutivas:

- Señalar las características únicas propias del proyecto en cuestión que permitan poder realizar un estudio de riesgos.
- Enmarcar legalmente el proyecto a realizar y determinar el espacio de mercado encaja.
- Poder ajustar el presupuesto del proyecto a las necesidades existentes.

Viabilidad económica

El primer punto que se tratarán los detalles económicos del proyecto, que son una parte fundamental para saber si el proyecto se puede asumir en un proyecto con carácter comercial desde el ámbito empresarial con carácter profesional.

En esta sección se explica cada uno de los costes asociados al proyecto teniendo en cuenta que se ha intentado minimizar los costes en todos y cada uno de los puntos de la instalación y material.

Coste de personal

Aún habiendo sido desarrollado por una única persona, el apartado de costes de personal es el que contiene un gasto mayor con respecto al resto de apartados económicos.

Se ha estimado un sueldo de 22.000€ brutos anuales en 12 pagas para realizar el cálculo en el intervalo entre el 01 de octubre de 2020 y el 20 de enero de 2021. Si hacemos el cálculo de estos 111 días que ha durado el proyecto, vemos que se corresponde a 3,7 meses de trabajo por lo que el gasto total en personal durante el proyecto asciende a 8.818,33€. El desglose mensual podemos verlo en la tabla A.1 y el coste total en la tabla A.2.

Concepto	**Porcentaje**	**Coste**
Sueldo Bruto Mensual		1833,33 €
Contingencias Comunes	23,60 %	432,67 €
Tipo general	5,50 %	100,83 €
Fogasa	0,20 %	3,67 €
FP	0,70 %	12,83 €
Total Gasto Mensual (Suma)		**2.383,33 €**

Tabla A.1: Coste de personal mensual

Concepto	**Coste**
Número de meses	3.7 meses
Gasto mensual	2.383,33 €
Total Gasto Proyecto	**8.818,33 €**

Tabla A.2: Coste total en personal durante el proyecto

Para realizar el cálculo me he apoyado en la documentación oficial que tiene a disposición del ciudadano la Seguridad Social en su página web [11]. En ella podemos ver que hay que aportar un 23,60 % en concepto de Contingencias comunes, un 5,50 % en concepto de desempleo de Tipo general, un 0,20 % a FOGASA y un 0,70 % para FP.

Coste hardware

Aunque se ha procurado ahorrar en costes de hardware, y sabiendo que este proyecto se puede realizar íntegramente desde la misma máquina Raspberry Pi que se ha utilizado para desarrollar y que puede se trabajar conectado a la máquina incluiré un equipo portátil de gama corporativa desde el que trabajar cómodamente.

Tras valorar las características de las diferentes Raspberry Pi, y sabiendo la potencia que necesita el proyecto, adquiriría la versión 3B+ por disponer de mayor potencia y mejor conectividad por una inversión razonable aunque para el proyecto 'reciclaré' una RbP 2B que tengo en desuso. Por ello, en el apartado económico figurará la versión 3B a un precio de 34,30€. El estudio preliminar se realizó en el milestone 3 [3] y puede observarse online mientras que el coste final se representa en la tabla A.3

Para calcular la amortización del material en el tiempo transcurrido he determinado un periodo de amortización de 5 años, de los cuales han transcurrido 3,7 meses.

Concepto	Coste	Coste amortizado
Portátil Lenovo Thinkpad T50	1.427,86 €	88,05 €
Raspberry Pi 2B	34,30€	2,12€
Tarjeta SD	14,08 €	0,87 €
Cable UTP5e	14,41 €	0,89 €
Caja Raspberry Pi	15,28 €	0,94 €
Placa Board	3,50 €	0,22 €
Dongle WiFi	6,95 €	0,43 €
Relés	38,35 €	2,36 €
Cables electrónica	6,59 €	0,41 €
Conectores de cables electrónicos	10,70 €	0,66 €
Crimpadora JST	12,00 €	0,74 €

Guía pasacables	2,00 €	0,12 €
Total	**1.586,02 €**	**97,80 €**

Tabla A.3: Coste total en hardware

Coste software

Como el único programa que no es online lo podemos correr sobre Linux, utilizaremos Linux para ahorrar en costes. El único gasto es la licencia del programa Fritzing, que tiene un costo total de 8€ [6].

Costes varios

Únicamente incluiré una tarifa de datos y una de móvil por un precio de 50€/mes, al ser casi 4 meses el precio asciende a 200€. El motivo es que se necesita un número de teléfono móvil para poder hacer pruebas con Telegram aunque lo ideal es utilizar dos números de teléfono para hacer algunas pruebas.

Costes Total

Si sumamos los costes anteriormente descritos, vemos que la suma asciende a **10.612,35€** que han quedado descritos en la tabla A.4.

Concepto	**Coste**
Personal	8.818,33€
Hardware	1.586,02€
Software	8€
Caja Raspberry Pi	200€
Total	**10.612,35 €**

Tabla A.4: Tabla de costes totales

Beneficios

En un principio el proyecto está pensado para ser utilizado de forma libre pero, en el caso de cambiar de idea se puede monetizar de varias formas.

Quizás, la más sencilla sea un pago por licencia con carácter anual pudiendo poner una suscripción de 1€ o 2€ por cada mes de uso en cada instalación de forma que se podría rentabilizar rápidamente como se ha plasmado en la tabla A.5.

Tiempo	1 Instalación	2 Instalaciones	3 Instalaciones
1 año	12€	24€	36€
2 años	22€	45€	50€

Tabla A.5: Cuotas

Viabilidad legal

Tanto en el proyecto como en los anexos se ha hecho alusión a algunas licencias como «Creative Commons [2]» y «GNU [7]». En este apartado se detalla el marco legal del proyecto, pero para poder entenderlo debemos entender que es una licencia:

> *Una licencia de software es un contrato entre el licenciante (autor/titular de los derechos de explotación/distribución) y el licenciatario (usuario consumidor, profesional o empresa) del programa informático, para utilizarlo cumpliendo una serie de términos y condiciones establecidas dentro de sus cláusulas, es decir, es un conjunto de permisos que un desarrollador le puede otorgar a un usuario en los que tiene la posibilidad de distribuir, usar o modificar el producto bajo una licencia determinada.* [16]

Para poder determinar la licencia más adecuada para nuestro proyecto debemos hacer un estudio de las licencias que se utilizan en todo el proceso ya que hemos utilizado licencias a tener en cuenta.

Frontend

En el proyecto se ha utilizado un bot de Telegram como FrontEnd del sistema domótico. Podemos ver las licencias del sistema que utilizamos en la tabla A.6.

Nombre	Licencia	Descripción
Telegram Code	MIT	Código interno de Telegram
Telegram App [13]	GPLv2 y post	Aplicaciones móviles
Telegram Api [12]	GPLv3 excepto OpenSSL	API pública de Telegram

Tabla A.6: Dependencias en el FrontEnd.

Backend

En la parte de BackEnd nos hemos servido de otros códigos o «dependencias» que también cuentan con sus propias licencias, como podemos ver en la tabla A.7. Al realizar un pequeño estudio podemos determinar compatibilidad de las licencias de las dependencias utilizadas en nuestro proyecto y que podemos ver en la imagen A.1.

Nombre	Versión	Licencia	Descripción
Raspbian	5.4	GPL	Sistema Operativo de Raspberry Pi
Python [10]	3.7	PSF	Lenguaje de programación interpretado
urllib3	1.24.1	MIT	Cliente HTTP
datetime	-	ZPL	Módulo para trabajar con fechas y horas
pandas	1.1.4	BSD	Herramienta de manipulación de datos de alto nivel
matplotlib	3.3.3	PSF based	Biblioteca para la generación de gráficos
telebot	0.0.4	MIT	Biblioteca que encapsula todas las llamadas a la API
time	-	MIT	Biblioteca que ofrece funciones relacionadas con el tiempo
simplejson	3.16.0	GPL	Paquete para trabajar con objetos json [5]
requests	2.21.0	Apache 2.0	Biblioteca HTTP
pycurl	7.43.0.6	LGPL/MIT	Cliente HTTP
telegram	0.0.1	GPLv2 to LGPLv3	Cliente para Telegram
sys	-	MIT	Para acceder a funciones del sistema operativo
math	-	MIT	Para ealizar operaciones matemáticas
os	-	MIT	Para acceder a funciones del sistema operativo
pip	20.2.4	MIT	Administrador de paquetes de Python

Tabla A.7: Dependecncias en el BackEnd.

Tras ordenar las compatibilidades de las licencias de las dependencias de nuestro proyecto vemos que la licencia más compatible es GPL3 [8], y para evitar interferir de algún modo con las licencias de alguna de ellas, nuestro software cuenta con dicha licencia GPL3.

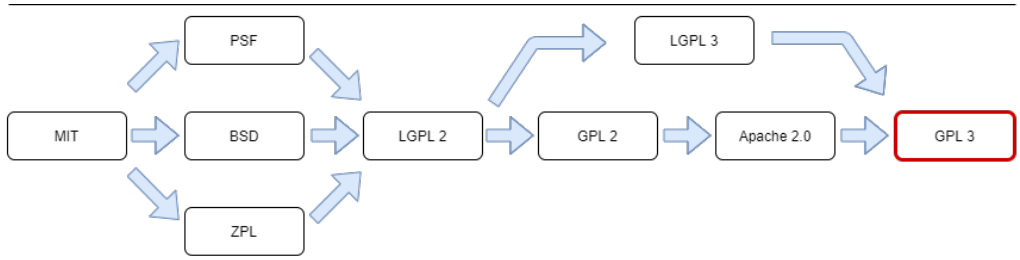


Figura A.1: Compatibilidad entre licencias.

El liberar el código bajo GPL3 [8] resulta beneficioso ya que al ser de libre disposición cualquiera puede beneficiarse de éste, del mismo modo que puede mejorarlo y licenciarlo con una licencia libre compatible y poder aprovechar también dichas mejoras en futuros desarrollos.

Por ello, para entender que permite la licencia GPL3 [8] he confeccionado un breve resumen. Aunque la tabla A.8 no representa la casuística completa de la licencia si nos permite tener una idea global del funcionamiento de ésta.

Permitido	Limitaciones	Obligaciones
Uso Comercial y privado.	Se prohíbe la concesión de sublicencias.	Se debe incluir el software original.
El código se puede modificar.	Responsabilidad limitada.	Debe incluir los cambios en el software.
El código se puede distribuir.	No dispone de garantía.	El código debe divulgarse bajo una licencia compatible con GPL 3.0.
Se puede otorgar garantía al código con licencia.		Debe incluir la licencia.

Tabla A.8: GPL3

Podemos ampliar la información en la página oficial de GNU [8].

Documentación

Para generar la documentación se han utilizado dos licencias, la referente a LaTeX y a la aplicación Fritzing. Los diagramas electrónicos del proyecto han sigo generados con esta licencia Creative Commons by-sa [1]. La documentación del proyecto tiene la misma cobertura CC by-sa, que permite el uso comercial y distribución tanto de esta obra como derivadas de ésta siempre que se cuente con la misma licencia que regula la obra original. Podemos ver los derechos adquiridos de la obra así como las obligaciones en la página oficial [1].

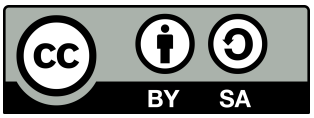
Nombre	Licencia	Descripción
LaTeX [15]	LPPL	Procesador de textos
Fritzing	CC by-sa	Diagramas electrónicos

Resumen del licenciamiento del presente proyecto

Para facilitar la búsqueda y comprensión de las licencias con que cuenta el proyecto he generado la tabla resúmen A.10 y también pueden ver las licencias en los enlaces correspondientes de GPL3 [8] y CC-BY-SA-3.0 [1].

Recurso	Licencia
Código Fuente	GPL3
Documentación	CC-BY-SA-3.0
Imágenes	CC-BY-SA-3.0
Vídeos	CC-BY-SA-3.0

Tabla A.10: Licencias que protegen el proyecto.



Apéndice B

Especificación de Requisitos

B.1. Introducción

En este punto se detallan los requisitos con que debe cumplir el proyecto y que determinarán restricciones, funcionalidades y comportamiento de éste.

B.2. Objetivos generales

Los objetivos generales enmarcan la funcionalidad del proyecto, que se detallan a continuación:

- Crear una plataforma que consiga aumentar confort, comodidad y seguridad dentro del domicilio.
- Generar un Sistema Domótico que nos permita automatizar parámetros de la vivienda así como poder ordenar acciones en tiempo real.
- Crear un Sistema Simulador de Presencia Domiciliaria de funcionamiento autónomo.
- Debe contar con una interfaz de comunicación multiplataforma para interactuar con el software.

B.3. Catálogo de requisitos

En este punto se desglosan los requisitos con que debe cumplir el proyecto, derivando de los anteriormente expuestos.

Requisitos funcionales

- **RF-1 Obtención de datos:** El Sistema debe ser capaz de obtener datos necesarios para su correcto funcionamiento.
  - **RF-1.1 Obtención de ubicación:** Debe ser capaz de determinar su ubicación geográfica.
  - **RF-1.2 Obtención de posición solar:** Debe poder obtener información sobre la posición solar con respecto del planeta según su ubicación geográfica.
  - **RF-1.3 Obtención de temperaturas:** Debe poder obtener información meteorológica de las próximas horas en la ubicación en la que se encuentra el Sistema.
  - **RF-1.4 Datos bajo demanda:** El usuario debe poder solicitar la obtención de los datos bajo demanda.
- **RF-2 Control de periféricos:** El Sistema debe poder controlar los periféricos que tiene conectados.
  - **RF-2.1 Control automático:** El sistema debe poder controlar de forma autónoma los dispositivos conectados.
  - **RF-2.2 Control instantáneo:** El usuario debe poder controlar los diferentes periféricos a placer.
  - **RF-2.3 Control planificado:** El sistema debe poder planificar el control de los periféricos conectados.
  - **RF-2.4 Parametrización:** El usuario debe poder parametrizar la automatización de los periféricos.
- **RF-3 Información:** La información de que dispone el Sistema siempre estará a disposición del usuario.
  - **RF-3.1 Información de sistema:** El usuario podrá obtener información básica sobre el sistema.
  - **RF-3.2. Información de planificación:** El usuario debe poder obtener información sobre la planificación de la automatización de los periféricos.
  - **RF-3.3. Información parametrización:** El usuario debe poder obtener información sobre la parametrización de la automatización de los periféricos.

  - **RF-3.4 Diagramas:** El usuario debe poder obtener un diagrama de temperaturas del día u otros días anteriores que estén archivados en la máquina a modo de registro.

- **RF-4 Control máquina:** Se deben poder ejecutar algunas tareas básicas sobre la máquina que soporta el Sistema.
  - **RF-4.1 Operaciones bajo demanda:** El usuario podrá apagar y reiniciar la máquina siempre que lo desee.
  - **RF-4.2 Operaciones automáticas:** El sistema se reiniciará, al menos, una vez al día.
- **RF-5 Gestión de la máquina:** La máquina que soporta el sistema debe poder ser accesible por parte del usuario.
  - **RF-5.1 Gestión local:** El usuario podrá gestionar el Sistema Operativo de la máquina que soporta el Sistema de forma local.
  - **RF-5.2 Gestión remota:** El usuario podrá gestionar el Sistema Operativo de la máquina que soporta el Sistema de forma remota vía VNC.
- **RF-6 Comunicación entre BackEnd y FrontEnd:** Debe existir un filtro en las comunicaciones.
  - **RF-6.1 Selección de usuarios:** El propio usuario debe poder gestionar los usuarios que pueden interactuar con el Sistema.
  - **RF-6.2 Rechazo de peticiones:** Los usuarios que no estén admitidos en la lista no podrán acceder a la comunicación con el Sistema.
  - **RF-6.3 Interacción multiplataforma:** Debe poder interactuar mediante las aplicaciones móviles existentes de Telegram y desde cualquier navegador popularmente extendido como pueden ser Chrome, Edge o Firefox entre otros.

Requisitos no funcionales

- **RNF-1 Escalabilidad:** Debe poder ampliarse fácilmente.
- **RNF-2 Eficiencia:** Debe minimizar la carga computacional para minimizar también el consumo energético del Sistema, además de permitir que la temperatura exterior incida en menos medida en las condiciones deseadas optimizando el consumo de recursos.

- **RNF-3 Rendimiento:** El sistema debe ser fluido y evitar cargas innecesarias.
- **RNF-4 Usabilidad:** El FrontEnd debe ser lo más usable posible, es decir, fácil de utilizar y aprender e intuitivo, y adaptado a las necesidades que pretende cubrir.
- **RNF-5 Disponibilidad:** El Sistema debe estar siempre en correcto funcionamiento.
- **RNF-6 Durabilidad:** El software debe poder funcionar correctamente durante un tiempo relativamente largo.
- **RNF-7 Capacidad:** Debe poder obtener la información necesaria y actuar conforme a lo que se espera de él.
- **RNF-8 Documentación:** Debe existir la suficiente documentación para poder implementar e interactuar con el Sistema.
- **RNF-9 Operabilidad:** Debe permitir manejar y controlar el Sistema según los requisitos funcionales.
- **RNF-10 Mantenibilidad:** Debe desarrollarse de tal manera que el mantenimiento sea lo más fácil y rápido posible.
- **RNF-11 Seguridad:** Todas las operaciones desde el FrontEnd deben ser seguras y estar cifradas.
- **RNF-12 Legibilidad:** El software debe ser fácilmente legible.
- **RNF-13 Extensibilidad:** El código debe ser fácilmente adaptable y reutilizable.
- **RNF-14 Liberación de código:** Debe disponer de un Sistema Operativo GNU y el código debe tener algún tipo de licencia GNU.
- **RNF-15 Respaldo documental:** Toda la instalación debe realizarse conforme a los estándares legales vigentes.

B.4. Especificación de requisitos

La especificación de requisitos contempla los casos de uso contra nuestro código. Además, también podemos ver el diagrama de los casos de uso, aunque lo he dividido en los diagramas «1/2» B.1 y «2/2» B.2

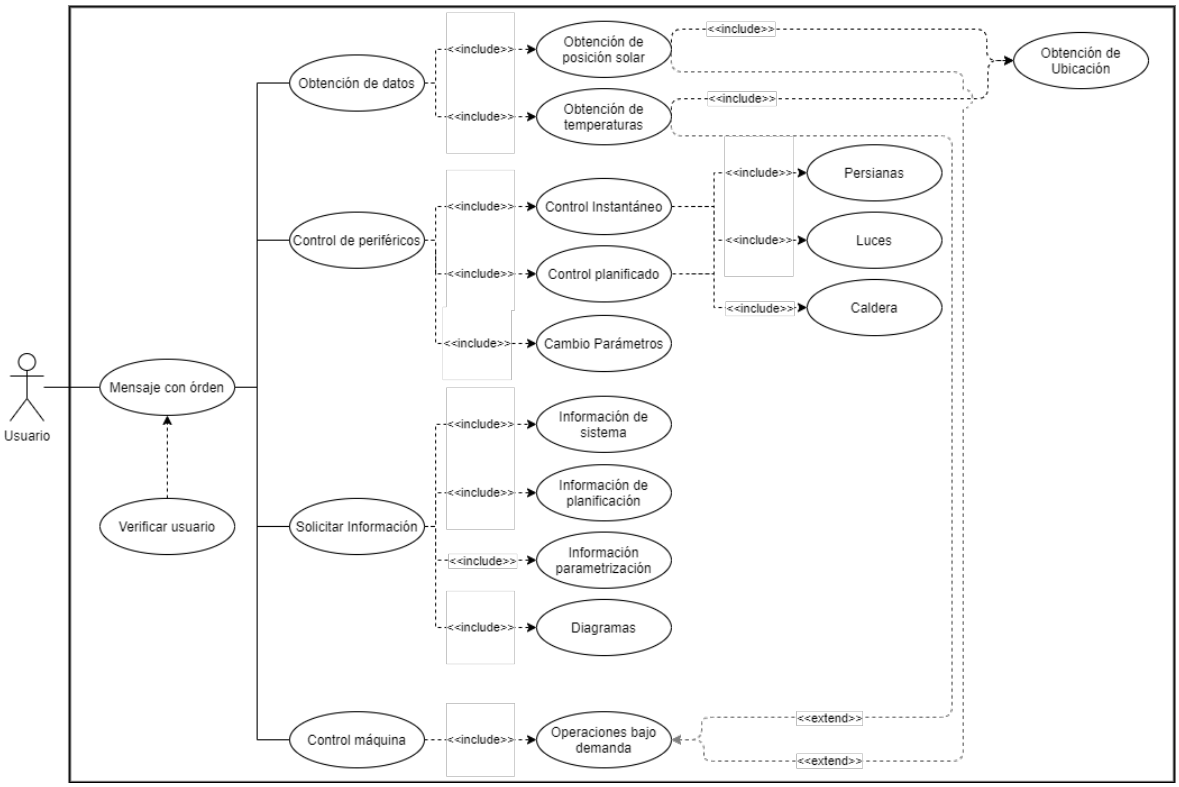


Figura B.1: Casos de uso 1/2.

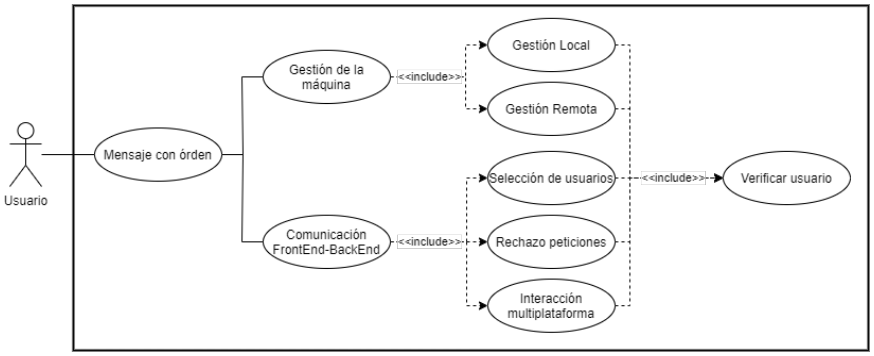


Figura B.2: Casos de uso 2/2.

Actores

Los actores serán cada uno de los usuarios del Sistema que controlará el sistema desde el FrontEnd.

CU-01	**Obtención de posición solar**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-1, RF-1.1, RF-1.2, RF-1.3, RF-1.4
Descripción	Permite al usuario obtener los datos necesarios para parametrizar el Sistema Domótico Inteligente.
Precondición	▪ Al ser un proceso principalmente automático, únicamente se requiere una línea de datos con acceso a Internet. ▪ En el caso de hacer la petición el usuario, también necesita ser uno de los usuarios autorizados.
Acciones	▪ El usuario solicita la obtención inmediata de los datos necesarios para realizar la próxima programación. ▪ El programa llama a las APIS de geolocalización, astrológicos y meteorológicos.
Postcondición	El bot lanza los scripts de recopilación de datos y almacena los datos.
Excepciones	Si no puede recopilar los datos envía mensaje al usuario.
Importancia	Alta

Tabla B.1: CU-01 - Obtención de datos

CU-02	**Control de periféricos**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-2, RF-2.1, RF-2.2, RF-2.3, RF-2.4
Descripción	Permite controlar los periféricos
Precondición	▪ Deben existir periféricos y estar configurados en el archivo al efecto. ▪ El usuario debe estar acreditado.
Acciones	▪ El usuario puede ordenar controlar un elemento.
Postcondición	El periférico cambia de estado.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Media

Tabla B.2: CU-02 - Control de periféricos.

CU-03	**Parametrización de periféricos**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-2, RF-2.4
Descripción	Permite parametrizar el control automático
Precondición	▪ Deben existir periféricos y estar configurados en el archivo al efecto. ▪ El usuario debe estar acreditado. ▪ Deben introducirse los parámetros correctos.
Acciones	El usuario parametriza la automatización de un periférico.
Postcondición	El periférico cambiará de estado.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Alta

Tabla B.3: CU-03 - Parametrización de periféricos

CU-04	**Solicitar Información**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-3, RF-3.1, RF-3.2, RF-3.3, RF-3.4
Descripción	Permite obtener información del sistema, de automatización, de la parametrización.
Precondición	▪ El usuario debe estar acreditado. ▪ Debe existir la información.
Acciones	Lectura de información.
Postcondición	Se envía información del sistema.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Baja

Tabla B.4: CU-04 Solicitar Información

CU-05	**Solicitar Información de sistema**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-3, RF-3.1
Descripción	Permite obtener información del sistema
Precondición	▪ El usuario debe estar acreditado. ▪ Debe existir el diagrama.
Acciones	Se envía información del sistema.
Postcondición	Se enviará la información solicitada.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Baja

Tabla B.5: CU-05 Solicitar Información de sistema

CU-06	**Solicitar Información de planificación**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-3, RF-3.1
Descripción	Permite obtener información sobre la planificación del funcionamiento de los periféricos.
Precondición	▪ El usuario debe estar acreditado. ▪ Debe existir el diagrama.
Acciones	Lectura de información.
Postcondición	Se envía información del sistema
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Baja

Tabla B.6: CU-06 Solicitar Información de planificación

CU-07	**Solicitar Información de parametrización**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-3, RF-3.1
Descripción	Permite obtener información sobre los parámetros personalizados introducidos por el usuario.
Precondición	▪ El usuario debe estar acreditado. ▪ Debe existir el diagrama.
Acciones	Lectura de información.
Postcondición	Se envia información de parametrización.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Baja

Tabla B.7: CU-07 Solicitar Información de parametrización

CU-08	**Solicitar Diagramas informativos**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-3, RF-3.1
Descripción	Permite obtener información sobre los parámetros personalizados introducidos por el usuario.
Precondición	▪ El usuario debe estar acreditado. ▪ Debe existir el diagrama.
Acciones	Busca el diagrama elegido.
Postcondición	Envía el diagrama.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Baja

Tabla B.8: CU-08 Solicitar Diagramas informativos

CU-09	**Operaciones bajo demanda**
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-4, RF-4.1
Descripción	Permite ejecutar apagado y reinicio de la máquina a placer.
Precondición	El usuario debe estar acreditado.
Acciones	Apaga o Reinicia el Sistema.
Postcondición	La máquina ejecuta la órden enviada.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Baja

Tabla B.9: CU-09 Operaciones bajo demanda

CU-10	Gestión local de la máquina
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-5, RF-5.1
Descripción	Permite administrar la máquina que soporta el sistema de forma local
Precondición	El usuario existir en el sistema.
Acciones	Administrar el sistema.
Postcondición	Si conoce las credenciales podrá administrar el sistema.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Alta

Tabla B.10: CU-10 Gestión local de la máquina

CU-11	Gestión remota de la máquina
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-5, RF-5.2
Descripción	Permite administrar la máquina que soporta el sistema de forma remota
Precondición	El usuario debe existir en el sistema, VNC debe estar funcionando y conocer las credenciales.
Acciones	Administrar el sistema.
Postcondición	Si conoce las credenciales podrá administrar el sistema.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Alta

Tabla B.11: CU-11 Gestión remota de la máquina

CU-12	Selección de usuarios
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-6, RF-6.1, RF-6.2
Descripción	Se deben poder escoger los usuarios que podrán interactuar con el bot, el resto quedan descartados.
Precondición	Los usuarios seleccionados deben existir en el archivo creado para ello.
Acciones	Permite interactuar con el bot o rechaza las peticiones.
Postcondición	Si está acreditado podrá interactuar con el sistema.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Alta

Tabla B.12: CU-12 Selección de usuarios

CU-13	Interacción Multiplaforma
Versión	1.0
Actor	Usuario
Requisitos asociados	RF-6, RF-6.1, RF-6.2, RF-6.3
Descripción	Debe poder interactuar desde un usuario existente y desde cualquier plataforma.
Precondición	El usuario debe existir y estar acreditado.
Acciones	Permite interactuar con el bot o rechaza las peticiones.
Postcondición	Podrá interactuar con el sistema.
Excepciones	Si no puede realizar la acción envía mensaje al usuario.
Importancia	Alta

Tabla B.13: CU-13 Interacción Multiplataforma

Apéndice C

Especificación de diseño

C.1. Introducción

En este punto se detalla cómo se han resuelto las especificaciones expresadas anteriormente.

C.2. Diseño de datos

Cabe recordar que este proyecto tiene la particularidad de que los elementos domóticos y lumínicos vuelven al estado de reposo al terminar el día y la programación diaria se realiza desde la posición de reposo.

El proyecto se divide en dos grupos funcionales: El primer grupo funcional se encarga de la obtención de datos del exterior y tratar estas cadenas de texto como json, y el otro grupo se basa en el tratamiento de cadenas de texto enviadas por el usuario mediante el bot de Telegram que hemos creado para la ocasión.

Ambas partes tienen un punto en común; éste es el archivo de configuración que contiene las credenciales y es dónde se plasma la arquitectura física entre los diferentes periféricos. El archivo se encuentra un nivel por encima del repositorio compartido en la carpeta «credentials» con el nombre «config2.bot».

La estructura del archivo será la expuesta en el Listing C.1:

Listing C.1: Estructura del archivo de configuración general de la arquitectura.

```
# Introduce tu Token aqui:
AQUI: TOKENBOT

# Introduce los usuarios autorizados separados por comas:
AQUI: USUARIO1, USUARIO2, ETC

# Key Climacell
AQUI: CLIMACELLKEY

# Key KeyWeatherapi
AQUI: WEATHERAPIKEY

#-------------------------------------------------------
# Numero de persianas [Situacion, PinSubir, PinBajar]
AQUI: EL NUMERO DE PERSIANAS A CONTROLAR
AQUI: SITUACION PINSUBIR PINBAJAR

#-------------------------------------------------------
# Numero de luces  [Situacion, Pin]
AQUI: EL NUMERO DE LUCES A CONTROLAR
AQUI: NOMBRE PINCONTROL

#-------------------------------------------------------
# Pin caldera [Situacion, Pin]
1
AQUI: NOMBRE PINCONTROL
```

Por un lado, el grupo funcional automático, llama a las APIS externas y guarda los datos en dos archivos antes de ser procesados y volcados al Cron. Para obtener los tokens, se sirve de la función `obtencionDatos.py` que lee la información del archivo anterior y calcula las rutas necesarias para el correcto funcionamiento del Sistema.

En este punto, de forma general, se recaba la información de forma automática llamando a las APIS aunque también se puede recopilar la información bajo petición del usuario.

El primer archivo, se llama «InfoRecabada» y es el archivo donde se almacenan los datos de las consultas en formato json. Posteriormente, se genera el archivo «log.cron», que nos presenta los datos junto a otros para

que podamos obtenerlos cuando queramos desde el bot. Ambos archivos necesitan ser leídos y tratados como «string».

Por otro lado, tenemos la parte del Sistema que nos permite operar contra éste con órdenes de carácter inmediato desde el FrontEnd de Telegram, pasando por su API hasta el «listener»; que es la parte de nuestro bot (que está corriendo continuamente) que está diseñado para leer el texto que le enviamos y actuar según tengamos programado en el archivo de configuración del bot. Esta configuración podemos verla en la imagen C.1.

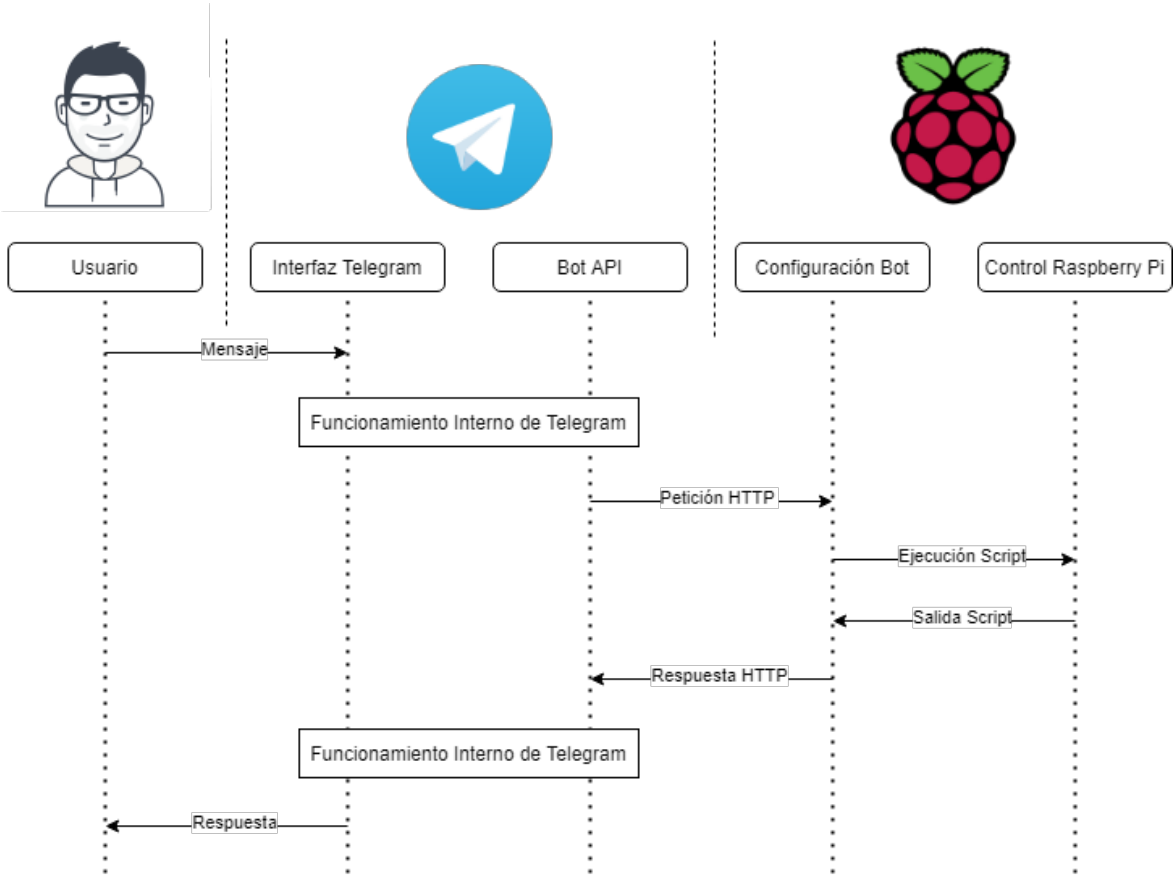


Figura C.1: Secuencia de acciones.

C.3. Diseño procedimental

En este punto se recogen los detalles más importantes en la ejecución las órdenes de nuestro Sistema, tanto la parte automática como la parte bajo demanda. Existen tres tipos de posibles llamadas de interacción desde

el usuario, éstas son: órdenes a la máquina, órdenes a los periféricos y órdenes para recopilar datos de las APIS. Estos tres tipos de llamadas se han ilustrado en la figura C.3. También podemos ver el flujo de comunicación entre las diferentes partes del software en la imagen C.2.

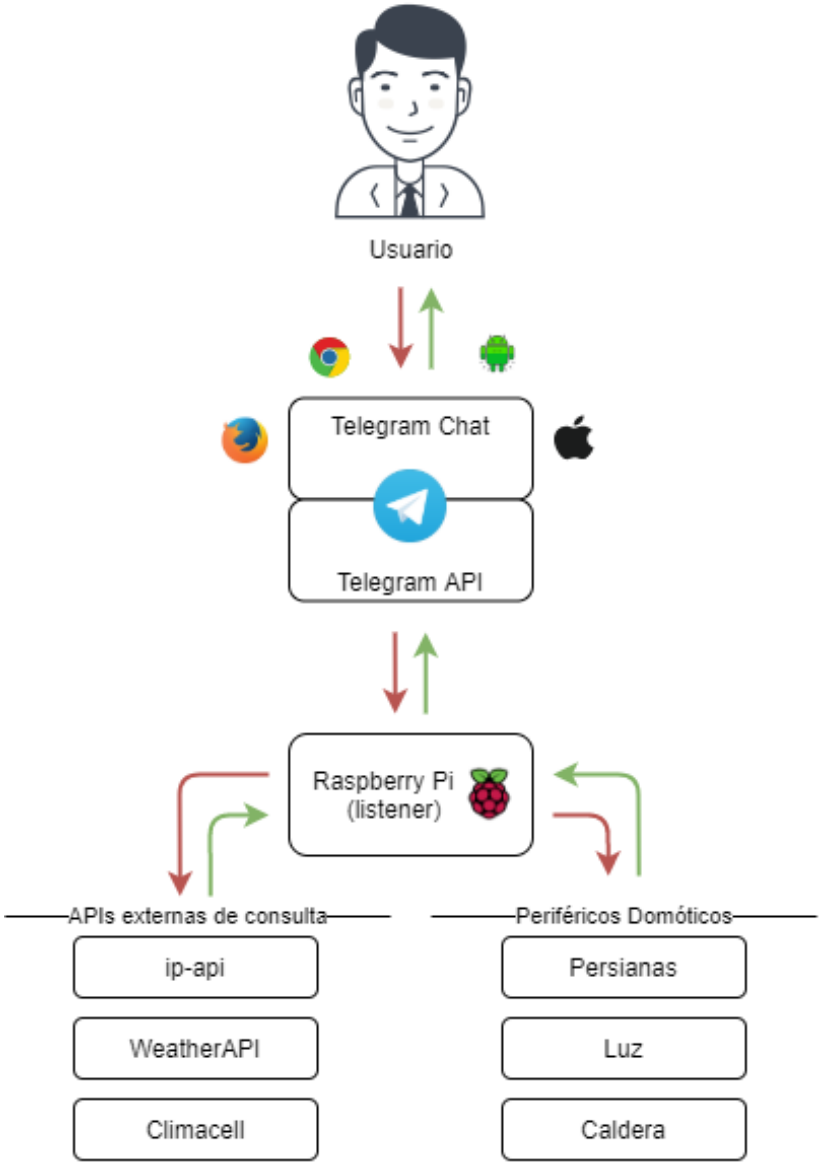


Figura C.2: Diagrama de comunicación.

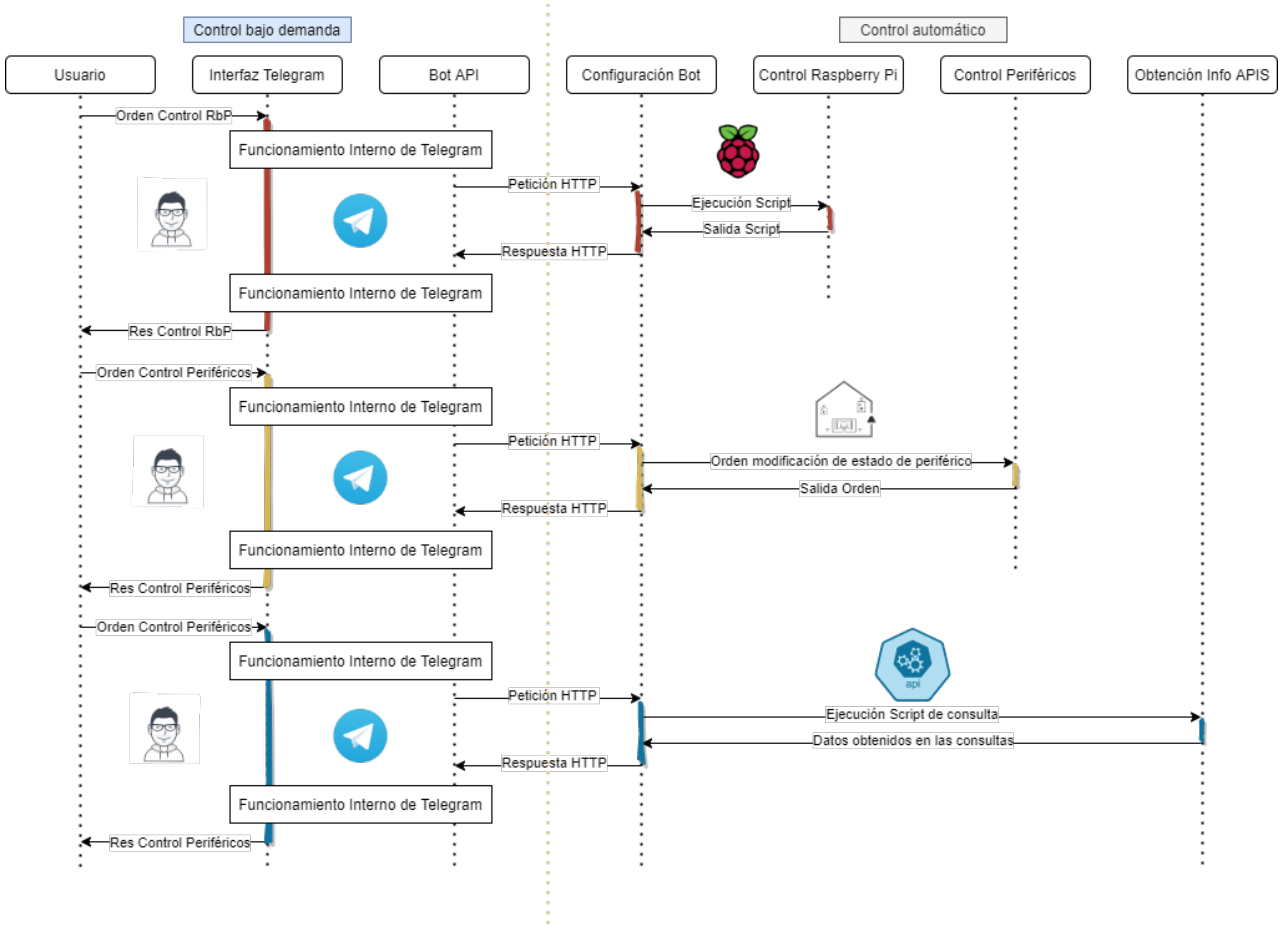


Figura C.3: Diagrama de secuencia de acciones del sistema.

C.4. Diseño arquitectónico

El diseño arquitectónico de nuestro sistema se basa en la intergación de diferentes microservicios independientes entre sí a nivel de desarrollo. Esto se refleja en que nos beneficiamos de unas APIS sin preocuparnos por la lógica del BackEnd permitiéndonos ampliar la carga de trabajo sobre estas APIs. Esto podemos verlo gráficamente en la imagen C.2.

Podemos ver la estructura de los diferentes módulos del software en la imagen C.4.

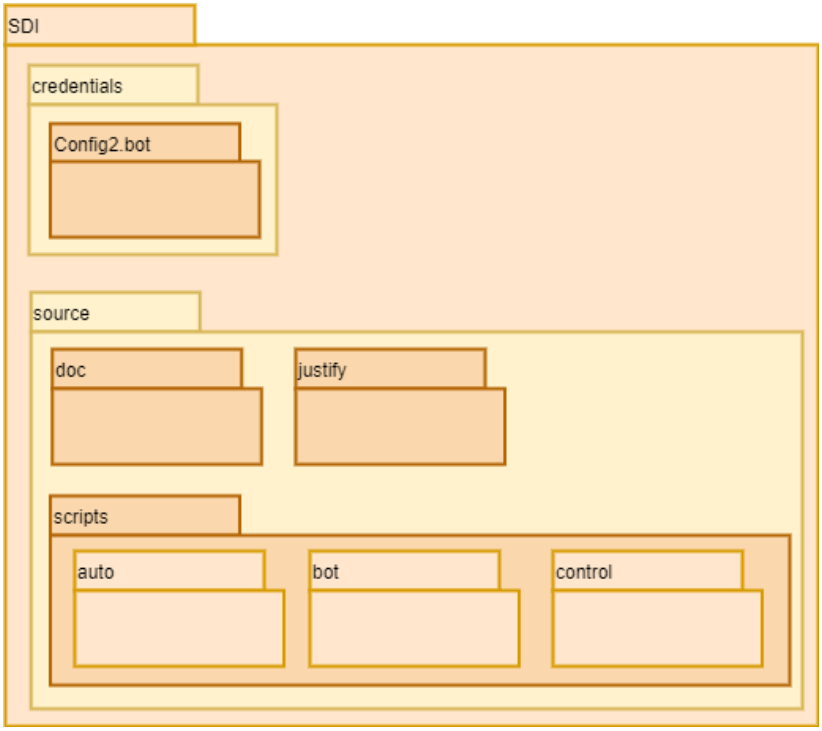


Figura C.4: Diagrama de paquetes.

Dentro de directorio scripts disponemos de los siguientes archivos de código:

- `obtencionDatos.py`: Lee el archivo de configuración y devuelve los valores de:
  - **tokenBot**: Token del bot de Telegram.
  - **users**: Usuarios acreditados.
  - **climacellKey**: Token de Climacell.

- **weatherApiKey**: Token de WeatherAPI.
- **persianas**: Información sobre las persianas.
- **luces**: Información sobre las luces.
- **calderas**: Información sobre la caldera.
- **rutaCred**: Ruta al directorio credentials.
- **rutaAuto**: Ruta al directorio auto.

En el directorio `auto` disponemos de los métodos y funciones propios de la parte automática:

- `1_recabaInfo.py`: Obtiene la información de las APIs externas y la graba en archivos de datos.
- `3_cocinado.py`: Procesa la información recogida y genera el próximo archivo de configuración de Cron.
- `4_reescribeCron.py`: Vuelca la nueva configuración y reinicia los servicios de la máquina.
- `LanzaTodoElProceso.sh`: Lanza los tres procesos anteriores.

En el directorio `control` disponemos de los métodos y funciones de control de los periféricos:

- `ApagarCaldera.sh`: Apaga la caldera.
- `EncenderCaldera`: Enciende la caldera.
- `Bajar`: Baja las persianas.
- `Subir`: Sube las persianas.
- `LucesOff`: Apaga las luces.
- `LucesOn`: Enciende las luces.

En el directorio `bot` disponemos de siguientes métodos y funciones que se encargan de realizar las acciones bajo demanda:

- `info(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, y nos envía por mensaje información sobre la máquina.

- `generador(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, lanza los scripts y nos devuelve la confirmación de la acción.
- `horaSubida(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, y nos informa de la hora a la que subirán las persianas y, si el mensaje incluye una hora la modificará. Finalmente nos envía la hora nueva de subida de las persianas.
- `diagrama(m, bot, pwdBot)`: Llama al método enviándole la orden por Telegram, el pwd y la información del bot, para enviar el último diagrama de temperaturas o el que el usuario solicite.
- `datos(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, y devuelve las principales horas de interacción del sistema automático.
- `controlPersianas(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, y nos permite interactuar con las persianas.
- `calefaccion(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, y nos informa de la temperatura a la que se encenderá la caldera, y si se incluye una temperatura se puede modificar. Finalmente envía la temperatura escogida tras grabarla.
- `temperaturas(m, bot)`: Llama al método enviándole la orden por Telegram y la información del bot, y devuelve por mensaje una tabla con las temperaturas por horas, además de la hora a la que suben las persianas y encienden las luces (y posteriormente bajan las persianas).
- `configBot()`: Obtiene los parámetros de configuración gracias a obtencionDatos(), obtiene la hora y la devuelve a todos los usuarios.
- `command_help(m)`:
  - Si `m` es **ayuda**, envía el menú de ayuda.
  - Si `m` es **reiniciar**, reinicia la máquina.
  - Si `m` es **apagar**, apaga la máquina.
- `compruebaUsuario(message)`: Comprueba si el remitente del mensaje es un usuario autorizado.
- `command_default(m)`: Devuelve el mensaje por defecto.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este apartado se incluye la información técnica relacionada con la instalación física y con el código, incluyendo la estructura de directorios y el proceso de la instalación física así como los pasos necesarios para la obtención de los diferentes tokens.

D.2. Estructura de directorios

Para comprender la estructura de directorios debemos conocer las diferentes partes del proyecto y tener en cuenta que son completamente diferentes aunque en este proyecto se han integrado para poder hacer uso del sistema domótico.

Tras accederal directorio raiz del proyecto vemos dos directorios, **credentials** y **source**. Dentro de **credentials** tenemos el archivo principal de configuración y dentro de **source** el resto de directorios del proyecto. Nada más acceder al directorio **source** podemos ver tres directorios principales:

1. El directorio **doc** contiene la redacción del proyecto.
2. El directorio **justify** contiene las normativas aplicables en la instalación física.
3. El directorio **scripts** es el que contiene todo el código del proyecto.

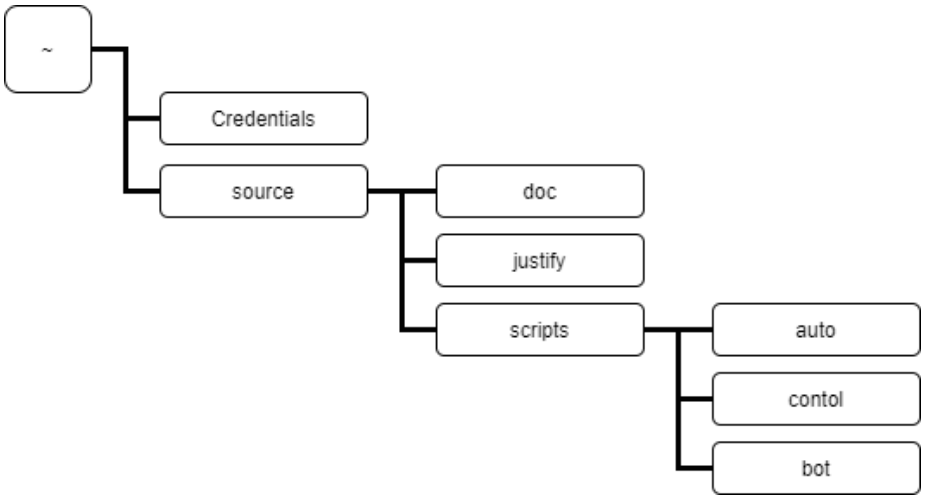


Figura D.1: Árbol de directorios.

Para entender la organización del directorio que almacena el código hay que saber que el código se divide, principalmente, en tres partes, que podemos verlas en la figura C.3 son las siguientes:

- La parte de toma de datos.
- La parte de mensajería.
- La parte física de nuestra instalación domótica, incluyendo la Raspberry Pi.

Por ello, dentro del directorio llamado «scripts» que es el que almacena nuestro código, éste se ha estructurado en tres partes:

- La carpeta **auto**, que permite obtener la información de las APIs seleccionadas de Internet.
- La carpeta **bot**, donde almacenamos la configuración de nuestro bot.
- La carpeta **control**, donde almacenamos los scripts de control de la instalación domótica.

En todas ellas, los scripts, tanto de Bash como de Python llevan la cabecera específica para el tipo de código contenido en el archivo:

```
0 #!/usr/bin/env python3
```

Ésta es la cabecera para los archivos Bash:

```
0 #!/bin/bash
```

He introducido estas cabeceras porque, sin ellas, he tenido problemas al lanzar las instrucciones desde otro script.

Carpeta auto

Al comienzo del proyecto esperé encontrar alguna web desde la que poder recoger la información haciendo webscraping pero tras valorarlo detenidamente en las tutorías me decanté por el uso de APIs. Uno de los motivos es que una web es más susceptible de sufrir cambios que una API que se ha predispuesto para una función específica. Finalmente, dispuse el proyecto para poder funcionar con dos APIs públicas y gratuitas para dotar a nuestro sistema domótico de autonomía.

Las siguientes fases de la parte de toma de datos, se albergan en la carpeta auto ya que es un proceso automatizado que permite al sistema obtener los datos de forma desatendida y continua. El proceso de automatización comprende varias fases:

1. La fase de **recopilación** de datos.
2. La fase de **lectura** local de parámetros.
3. El **cocinado** de los datos.
4. El **grabado** de datos en el sistema local.

La fase de **recopilación** de datos llama a la API de **geolocalización** y nos devuelve, entre otros, los parámetros de nuestra ubicación en formato de latitud y longitud. Posteriormente, estos datos de ubicación son necesarios para incorporarlos en la consulta de las APIs de **meteorología** y **astronomía** y obtener la información de la salida y puesta del sol así como la información de las temperaturas del día siguiente en dicha ubicación. Con esta recopilación de datos tenemos la base para poder determinar el comportamiento autónomo que necesitamos por lo que almacenaremos esta información en un archivo de datos.

El siguiente paso es leer las preferencias del usuario, que se almacenan en el archivo de condicionantes. Y, una vez tengamos estos datos, podemos pasar a generar el Cron para controlar de forma automática toda la instalación y

generar el diagrama de temperaturas para poder enviarlo cuando se solicite. Las imágenes se almacenan dentro del directorio ./diagramas, en formato PNG.

Estas fases se han confeccionado al final del proyecto puesto que al principio se desarrolló de forma unitaria pero surgió el problema de que, cada vez que queríamos hacer una modificación en el sistema domótico de cualquier índole, teníamos que hacer una petición nueva a las APIs y generar de nuevo un archivo Cron con los parámetros del usuario «al vuelo» para conseguir regenerar la automatización con los parámetros nuevos. Este sistema funcionaba correctamente pero era poco eficiente puesto que se multiplican las operaciones en función de las veces que se quiera modificar algún parámetro. Únicamente he permitido realizar dos salidas «al vuelo» tras obtener los datos, estos son: el grabado de los datos recién obtenidos y la gráfica de las temperaturas conforme a dichos datos.

He tenido algunos problemas al realizar varias peticiones a las APIS ya que, por falta de permisos, no podía sobreescribir los archivos ni las imágenes. Para subsanarlo, incluí una restricción para modificar los permisos de los archivos de forma que únicamente pueden acceder el propietario y el grupo.

Carpeta control

La carpeta control alberga aquellos scripts bash que controlan los periféricos. Este punto únicamente me dio problemas a la hora de implantarlo desde el bot, que se subsanó con la librería os.

Carpeta bot

La carpeta bot contiene varios archivos, desde el archivo de control del bot hasta otros que contienen la mayoría de funcionalidades de éste. En primera instancia, se dispuso todo el código dentro del archivo de control del bot, pero por legibilidad y facilidad de mantenimiento se extrajeron las funciones. En este punto, lo que más problemas me ha dado han sido los teclados. Como el comportamiento de los teclados es diferente entre ellos tendría que dividir las opciones del bot entre dos tipos de teclados diferentes. Finalmente, opté por no incluir estos teclados ya que no aporta funcionalidad alguna y disminuirían la legibilidad y facilidad de uso del bot. Además, para aumentar la usabilidad del bot, se ha dispuesto un menú que podemos ver al introducir el carácter «/» además de la posibilidad de utilizar la primera letra del comando siempre que no tengamos que acompañarlo de parámetros.

Regeneración del código al fraccionarlo por funcionalidades

Tuve un problema en la fase de regeneración del código al fracccionarlo por funcionalidades puesto que obtenía las rutas a los archivos que necesitaba llamar de forma unitaria desde el archivo en ejecución. Lo subsané con el siguiente código Python:

```
0 #!/usr/bin/env python3
1 #Calculamos ruta
2 ruta=os.getcwd().split('/')
```

Tuve que realizar un paso similar para los archivos Bash:

```
0 #!/bin/bash
1 path=$(pwd)
```

Aunque en la segunda versión de código se utiliza desde la propia variable de sistema:

```
0 python3 ${PWD%/*}/auto/1_recabaInfo.py
1 python3 ${PWD%/*}/auto/3_cocinado.py
2 sh ${PWD%/*}/auto/4_reescribeCron.sh
```

D.3. Manual del programador

Este punto tiene como objetivo explicar los medios necesarios para continuar con el desarrollo del código así como la explicación de las partes del proyecto. Como la parte de desarrollo se ha generado desde Raspbian OS utilizando la interfaz para desarrollo Python, «Thonny Python IDE», este punto será corto ya que el propio sistema operativo dispone del software necesario.

Requerimientos

Para poder hacer funcionar correctamente el código se debe clonar el repositorio e instalar los «requeriments» que están dentro de la ruta **/source/scripts/control/**:

```
0 pip install -r requirements
```

El software necesario es:

- pyTelegramBotAPI
- python-telegram-bot
- pycurl
- telegram
- urllib3
- simplejson
- datetime
- requests
- datetime
- pandas
- matplotlib
- bs4

También hay que generar el archivo de configuración al mismo nivel que nuestro directorio raíz, dentro de un directorio llamado «**credentials**», llamado `config2.bot`, de forma que quede como en la imagen D.1. La información que debe contener el archivo es la que figura en el código del listing C.1, incluyendo los tokens correspondientes y los elementos a controlar. En las «releases» ya se incorpora el directorio y una plantilla para rellenar el archivo `config2.bot` correctamente.

Archivos

Dentro del directorio `auto` encontramos:

- `/source/scripts/auto/1_recabaInfo.py`: Obtiene la información necesaria de las APIS.
- `/source/scripts/auto/2_condicionantes.py`: Contiene preferencias de usuario.

- `/source/scripts/auto/3_cocinado.py`: Genera el archivo preliminar del Cron para controlar las instalaciones.
- `/source/scripts/auto/4_reescribeCron.sh`: Reescribe el archivo de configuración del servicio Cron y lo reinicia.
- `/source/scripts/auto/cabecera.txt`: Contiene la cabecera que utilizamos para incluirla en Cron.
- `/source/scripts/auto/CronPruebas`: Contiene el texto que se volcará al archivo de configuración de Cron.
- `/source/scripts/auto/InfoRecabada`: Contiene la información que se ha obtenido de las APIs.
- `/source/scripts/auto/LanzaTodoElProceso.sh`: Lanza los 4 primeros scripts de forma desatendida.
- `/source/scripts/auto/log.cron`: Contiene información a consultar por el bot.
- `/source/scripts/auto/obtencionDatos.py`: Nos permite leer el archivo /credentials.config2.bot y obtiene las rutas que utilizaremos.

Dentro del directorio `bot` encontramos:

- `/source/scripts/bot/bot.py`: Contiene el archivo de configuración del bot.
- `/source/scripts/bot/datos.py`: Obtiene las horas de amanecer, anochecer, encendido de luces y subida de persianas por la mañana y los envía al usuario que lo solicita.
- `/source/scripts/bot/diagramaTemperaturas.py`: Envía la última imagen generada por el sistema u otra que especifique el usuario.
- `/source/scripts/bot/generador.py`: Lanza todo el proceso automático desde ./auto/LanzaTodoElProceso.sh.
- `/source/scripts/bot/horaSubida.py`: Envía la hora a la que se subirán las persianas por la mañana y permite cambiarla si el usuario especifica otra hora siempre que ésta sea después de la hora de amanecer.

- `/source/scripts/bot/info.py`: Nos muestra información de la máquina Raspberry Pi que soporta el Sistema.
- `/source/scripts/bot/obtencionDatos.py`: Nos permite leer el archivo /credentials.config2.bot y obtiene las rutas que utilizaremos.
- `/source/scripts/bot/subirPararBajar.py`: Nos permite controlar el movimiento de las persianas si existen en el archivo de configuración.
- `/source/scripts/bot/temperaturaCalefaccion.py`: Envía la temperatura a la que queremos que se encienda la caldera y nos permite modificarla.
- `/source/scripts/bot/temperaturasManana.py`: Envía la relación de temperaturas por horas en la ubicación de la máquina.

Dentro del directorio `control` encontramos:

- `/source/scripts/control/EncenderCaldera.sh`: Enciende la caldera.
- `/source/scripts/control/ApagarCaldera.sh`: Apaga la caldera.
- `/source/scripts/control/EstadoGPIO.sh`: Permite conocer el estado de los GPIO en ese instante de tiempo.
- `/source/scripts/control/Bajar.sh`: Baja todas las persianas.
- `/source/scripts/control/EstadoGPIO.sh`: Inhabilita los GPIO seleccionados.
- `/source/scripts/control/LucesOff.sh`: Apaga las luces.
- `/source/scripts/control/LucesOn.sh`: Enciende las luces.
- `/source/scripts/control/Prueba_General.sh`: Permite hacer una prueba de las persianas conectadas.
- `/source/scripts/control/Reinicia_Servicio_Cron.sh`: Reinicia el servicio Cron.
- `/source/scripts/control/subir.sh`: Sube todas las persianas.

Configuración del bot como servicio

El bot se puede lanzar como si fuera un script más pero en este caso es necesario incorporarlo como servicio. Esta decisión tiene beneficios, como la posibilidad de que se inicie con el sistema o que podamos lanzarlo o pararlo desde cualquier ubicación del SO sin tener que recordar la ubicación con la sentencia al efecto de «`sudo service bot start/stop`». Pero, una vez que el demonio está corriendo, tenemos que tener en cuenta de que ya está levantada la instancia del bot, por lo que tendremos que detenerla a la hora de hacer algún tipo de modificación así como recordar que debemos levantarlo una vez terminemos. Los pasos a seguir están en un archivo dentro de ~/**source/scripts/auto/**.

D.4. Compilación, instalación y ejecución del proyecto

Para realizar modificaciones en el código únicamente hay que seguir los siguientes pasos:

1. Descargar la última release que exista en el repositorio de GitHub en el formato que se prefiera: https://github.com/davidelinformatico/TFG/releases

2. Para extraer del archivo .tar utilizar:
   `tar -xvf SistemaDomoticoInteligente_v1.0.tar`.

3. Para extraer del archivo .tar.gz utilizar:
   `tar xzvf SistemaDomoticoInteligente_v1.0.tar.gz`.

En este punto ya tendremos todos los archivos extraídos y podremos editarlos. Recordar que para completar el software hay que instalar el software especificado en el archivo `requeriments` que se realiza de forma automática al correr el archivo setup contenido en ~/**source/scripts/auto/**.

Hay que tener en cuenta que en este punto no existen los archivos esperados de control de periféricos, en el directorio **control** porque se generan a medida del sistema que se configure el sistema tras completar el archivo `config2.bot` por lo que también se puede modificar la creación de estos achivos.

No hace falta un software específico para poder modificar el código ya que podemos utilizar los propios de la Raspberry Pi o algún editor sencillo como Nano según nuestras preferencias.

Si el objetivo es correr el software correctamente hay que recordar que es indispensable que el archivo config2.bot esté completo y que se lance el setup contenido en ~/**source/scripts/auto/**.

Finalmente, el proyecto puede lanzarse corriendo el archivo de configuración del bot: `bot.py` desde un terminal sh, aunque si se precisa también se pueden lanzar los scripts del directorio **auto** o **control** por separado ya que, aunque funciona todo como un conjunto, podemos probarlos de forma unitaria.

D.5. Pruebas del sistema

Como el proyecto no cuenta con testigos de cambios de estado de los periféricos es imposible realizar pruebas en este punto y lo mismo sucede con la parte de domótica automatizada, es imposible determinar si un código realiza correctamente una acción física ya que aunque el código funcione correctamente puede que no se esté llevando a cabo correctamente la acción.

Por otro lado, las pruebas sobre la parte del proyecto que se comunica con las APIs y procesa la información se ha realizado de forma manual ya que únicamente son dos archivos y son relativamente cortos.

Apéndice E

Documentación de usuario

E.1. Introducción

El usuario es aquella persona destinada a utilizar el Sistema generado.

E.2. Requisitos de la instalación física

Se incluye la instalación en este punto porque se entiende que cualquiera que disponga el código puede correrlo en su propia instalación realizando la instalación física.

Instalación física

Para poder realizar la instalación física de nuestra vivienda necesitamos el material detallado en la tabla A.3, ya que en este punto se cuenta con que la instalación física está completa para que el sistema pueda cumplir su cometido. Ahora utilizaremos los siguientes materiales:

- Bobina de Cable UTP5e.
- Placa Board.
- Relés con las características: [10A 250VAC 10A 125VAC] [10A 30VDC 10A 28VDC].
- Cables de electrónica.
- Conectores JST-XH.

Además requeriremos de crimpadora JST-XH y destornilladores de estrella y plano para trabajar con las clemas eléctricas (elementos de interconexión de conductores) del domicilio.

También necesitamos otros elementos de uso no específico para este proyecto pero que resultan necesarios para que el software funcione correctamente [1] como son el Dongle WiFi y una línea de datos con salida a Internet.

Tirada de cable

Para realizar la tirada de cable necesitaremos hacer algunos cálculos y tener claras algunas consideraciones:

Consideraciones

Debemos tener claras las características eléctricas que soporta cada uno de los pines GPIO de nuestra Raspberry Pi. Cada uno de los pines, soporta 3,3VDC [2] y 16 mA, excepto algunos pines de alimentación, que cuentan con hasta 5VDC. Hay que comprobar que cada uno de los hilos del cable UTP5e que vamos a instalar soporta la señal que vamos a transmitir por el cable aunque según la norma TIA/EIA568, en el punto A.3 especifica que deben pasar pruebas de hasta 500VDC y un mínimo entre dos cables de 100M. Además, también se especifica que debe contar con un AWG [14] de, entre 22 y 24, lo que se traduce en que soporta 0.577A-0.92A [9], respectivamente.

Cálculos necesarios

Debemos calcular si podemos introducir nuestros cables por los tubos con seguridad. Esto viene recogido en el REBT, más concretamente en apartado ITC_BT_21 de éste.

Sabiendo que nuestro cable UTP tiene unos 5mm de diámetro de sección, que la cara interna del tubo es de 17,8mm y que cada cable eléctrico tiene un área de 1,5mm^2 podemos calcular el área total ocupada y el tubo por el que debería hacerse la instalación con la seguridad requerida por norma:

$$\pi \cdot D^2/4 >= FactorCorrector \cdot (\sum (NumeroDeCabes \cdot \pi \cdot DiametroCable/4)) \quad \text{(E.1)}$$

[1] El software puede funcionar correctamente también vía Ethernet.
[2] VDC:Voltage Direct Current.

Si realizamos el cálculo, observamos que la suma del área que ocupan los tres cables de 1,5mm^2 con el área del UTP (19,63mm^2) hace una ocupación total de 24,13mm^2.

Por otro lado, tenemos que el tubo que alberga los cables es de 20mm de diámetro de sección por la parte exterior del tubo y 17,80mm por la parte interna. Si calculamos el área interna obtenemos 248,85mm^2. Según norma debemos ocupar la tercera parte del tubo. Para comprobar que el tubo puede contener los cables con seguridad, debemos calcular si la multiplicación del factor de corrección (en este caso de **3** por estar en una canalización empotrada de una vivienda), por la suma de las áreas que ocupan cada uno de los cables:

$$3 \cdot [(3 \cdot 1,5) + (1 \cdot 19,63)] = 72,40 \tag{E.2}$$

Ahora, debemos hacer el mismo cálculo del área de la cara interna del tubo:

$$\pi \cdot (17,8)/4 \tag{E.3}$$

Finalmente, vemos que E.2 es mayor que E.2 de forma que se cumple que el área ocupada (multiplicada por el factor de corrección) es inferior al área total, de modo que podemos albergar nuestro UTP en los tubos sin problema.

Por facilitar futuras instalaciones, figuro el cálculo para el mismo tubo también con cables de 2,5mm^2 de área en la imagen E.1:

	Tubos		Cables		
	Exterior	Interior	Cable UTP	Cable eléctrico	
Radio (mm)	10,00	8,90	2,50		
Diámetro (mm)	20,00	17,80	5,00		
Área (mm^2)	314,16	248,85	19,63	1,50	2,50

		Cálculo ocupación corregido	
Ocupación total de los cables		Cable 1,5	72,40
Factor corrector	3,00	Cable 2,5	81,40

Figura E.1: Cálculos para introducir cableado en tubo.

Tras finalizar los cálculos, se hacen las tiradas de cableado y se crimpan los extremos. Podemos ver en la imagen E.2 una caja de derivación donde ya tenemos albergada la placa de relés.

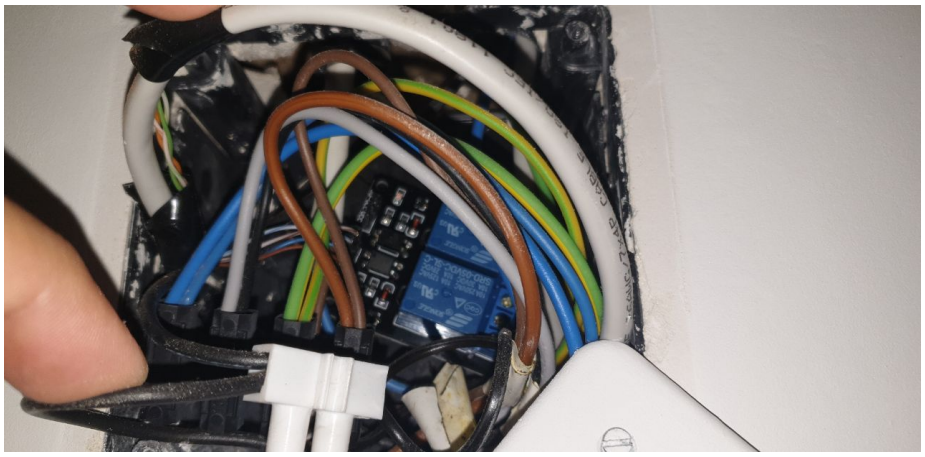


Figura E.2: Caja de derivación secundaria instalada.

En la imagen E.3 vemos la caja de derivación principal con la instalación terminada indicando algunos elementos.

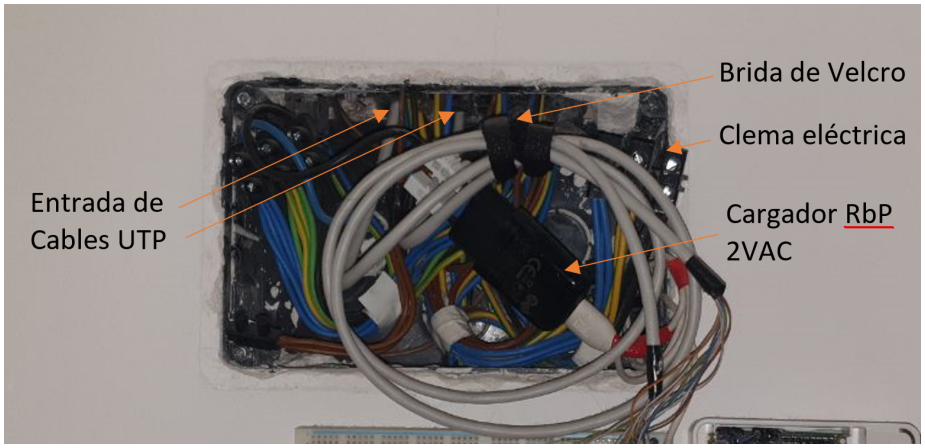


Figura E.3: Caja de derivación principal instalada.

El primer extremo del cable, podemos ver que termina en el relé dentro de la caja de derivación pero el extremo que conectará a la Raspberry Pi no se conectará directamente sino que terminará primero en una protoboard como podemos ver en la imagen E.4, donde podemos diferenciar que existen dos tipos de conectores. Los redondos ya vienen conectorizados de fábrica pero los cuadrados del tipo JST-XH los he fabricado según las necesidades.

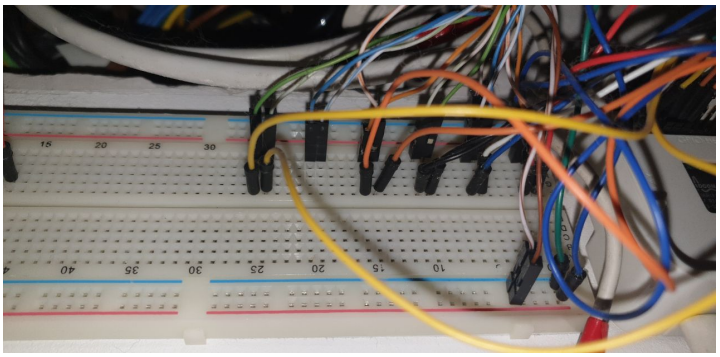


Figura E.4: Protoboard conectada.

En la imagen E.5 podemos ver la situación final de la tirada de cable y conectorizado.

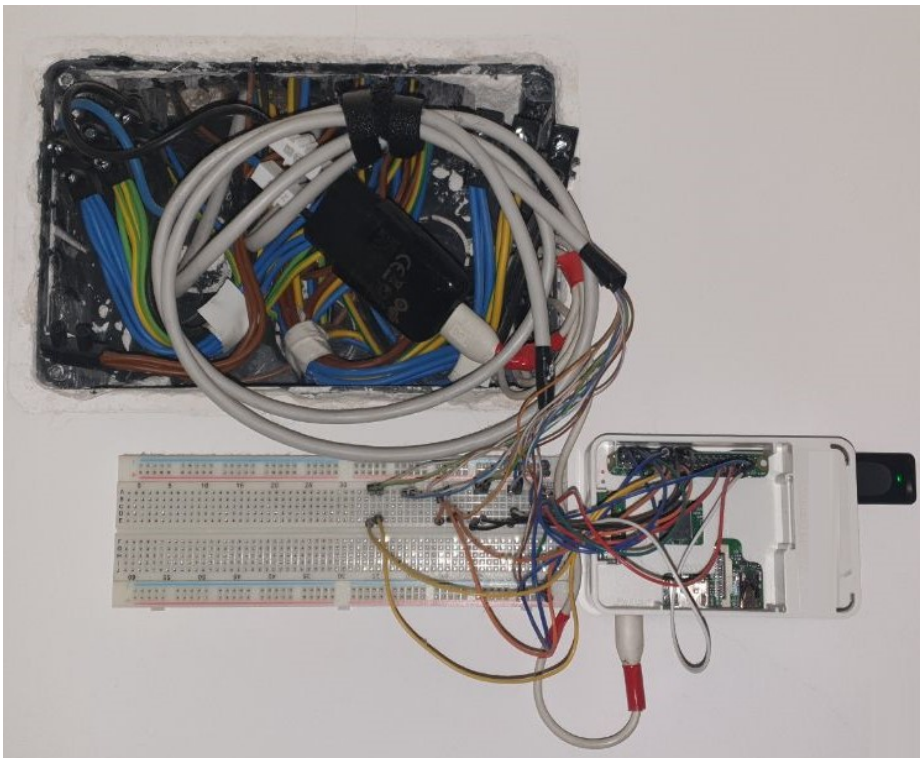


Figura E.5: Instalación domótica y caja de derivación principal.

El conectorizado final de las salidas de los GPIO es el que se puede ver en la imagen E.6

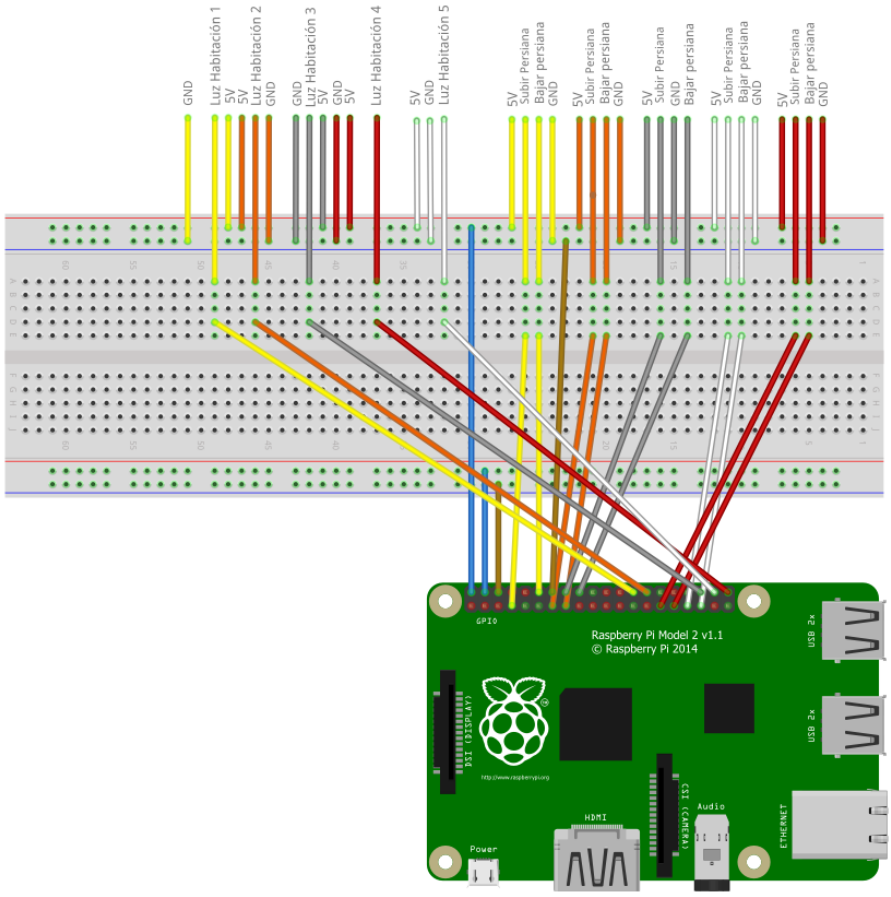


Figura E.6: Conectorizado final de las líneas GPIO.

Podemos interactuar con los puertos GPIO desde Bash y desde Python [10]. Se ha decidido utilizar lenguaje Bash para interactuar con los GPIO, dejando la potencia de Python para la obtención y procesado de datos. En ocasiones se producen errores al lanzar desde Cron scripts Python, por lo que se lanzará todo desde scripts bash.

Diagramas eléctricos

Para mayor claridad he generado unos planos eléctricos de cómo se ha realizado la instalación.

En la imagen E.7 podemos ver que tenemos conectada una placa con dos relés y un pulsador en paralelo, con el motor de la persiana. La protoboard realmente no existe en este punto pero la he incluido para clarificar el circuito.

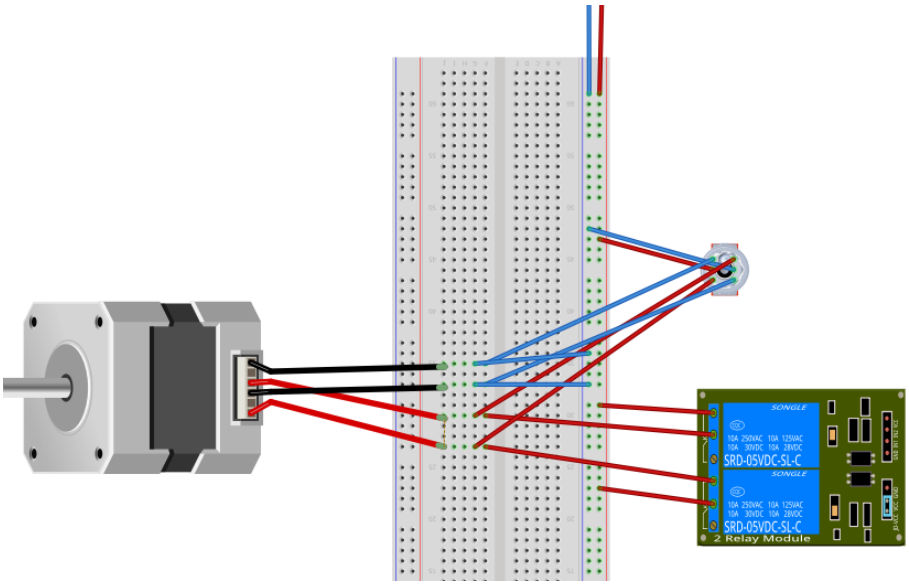


Figura E.7: Raspberry Pi, relé y pulsador.

En la imagen E.8 podemos ver como se conecta, desde la Raspberry, el relé que conecta con una persiana identificando cada uno de los pines.

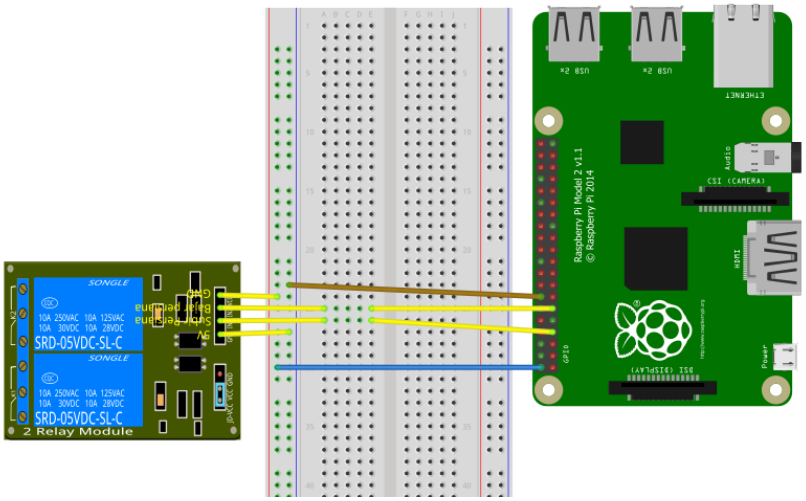


Figura E.8: Raspberry y relé conectados.

Para explicar el funcionamiento de un pulsador de 3 posiciones para persianas podemos ver la imagen E.9. En ella podemos ver que en la posición central no tiene ninguna salida eléctrica y que las salidas 1 y 2 tienen la polaridad invertida. La lógica del cambio de orden en la salida de cables es

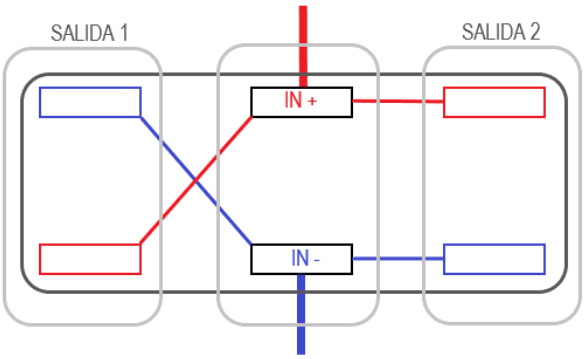


Figura E.9: Funcionamiento interno de un pulsador para persianas.

porque según el sentido de la polaridad el motor girará en un sentido o en otro. Esto se respalda mediante la segunda ley de Laplace para cada espira del bobinado del motor quedando de la forma $M = N * (I * B * l + sen\theta)$.

Siguiendo esta lógica, podemos ver que en la imagen E.7 tenemos dos relés de forma que cada uno de ellos deja pasar la corriente en un sentido. En la realidad, el motor no tiene por qué tener dos entradas positivas y dos negativas pero en el dibujo queda mejor ilustrado que son entradas diferentes.

Resumiendo, la instalación mecánica y nuestra instalación mediante relés se instalarán en paralelo para poder activar una opción o la otra según la ocasión.

E.3. Requisitos de usuarios

Para poder ejecutar el código, el usuario necesitará este material, que también podemos ver en la imagen E.10:

- Un terminal que soporte mensajería mediante Telegram. Cabe recordar que Telegram es multiplataforma, de modo que serviría un ordenador, un smartphone o un smartwatch entre otros.
- Una Raspberry Pi conectada a Internet.
- Una tarjeta uSD que haga de disco duro de nuestra Raspberry Pi.
- Alimentación de 2A para la Raspberry Pi.

- Otro equipo desde el que descargar y montar el Sistema Operativo de Raspberry Pi.
- Necesita generar cuentas en Climacell y WeatherAPI y un bot en BotFather.

Figura E.10: Material básico para ejecutar el código.

E.4. Obtención de tokens

En este punto se indica como obtener un token para poder utilizar las APIs.

Climacell

La primera API es Climacell. Para obtener el token debemos realizar los siguientes pasos:

- Accedemos a la web oficial: https://www.climacell.co/pricing/.
- Seleccionamos la opción gratuita.
- Rellenamos los datos que nos piden.
- Obtenemos el token de la pantalla que nos aparece.

Este token debemos guardarle muy bien puesto que es uno de los que utilizaremos para que el sistema funcione correctamente.

WeatherApi

La segunda API es WeatherApi. Para obtener el token debemos realizar los siguientes pasos:

- Accedemos a la web oficial: https://openweathermap.org/price.
- Seleccionamos la opción gratuita.
- Rellenamos los datos que nos piden.
- Accedemos a la pestaña de API Keys.
- Obtenemos el token de la pantalla que nos aparece.

Este token también debemos guardarlo muy bien puesto que es uno de los que utilizaremos para que el sistema funcione correctamente.

BotFather

Los pasos para crear nuestro bot y obtener su token, son:

- Desde una instancia logada de Telegram debemos buscar un chat llamado BotFather.
- Le enviamos el mensaje `/start`
- Le enviamos el mensaje `/newbot`
- Ahora, debemos enviarle el nombre que queremos ponerle a nuestro bot.
- La respuesta del BotFather es un mensaje donde nos indica el nombre de nuesto bot y el token que debemos guardar (en rojo).

Éste es el último token que debemos guardar para completar el archivo `config2.bot`.

E.5. Instalación

Sistema Operativo

El primer paso es descargar el Sistema Operativo de la web oficial y montarlo en la tarjeta SD con la ayuda de otro equipo informático y el adaptador USB para uSD que vemos en la imagen E.10.

1. Accedemos a: https://www.raspberrypi.org/software/
2. Descargamos el software para montar la imagen en nuestra uSD https://downloads.raspberrypi.org/imager/imager_1.5.exe
3. Introducimos el adaptador USB con la uSD en el equipo y corremos el software «imagen_1.5.exe» para instalarlo, en este caso.
4. En la ventana de instalación, seleccionamos «install» como figura en la imagen E.11 para instalar el programa «imager» de Raspberry Pi Foundation.

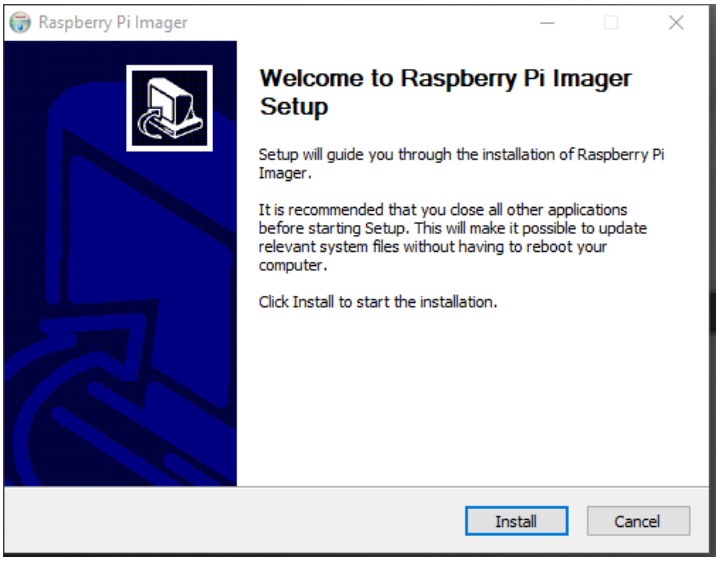


Figura E.11: Ventana instalación «imager», de Raspberry Foundation.

5. Seleccionamos las opciones de «Sistema Operativo» y «SD Card» como figura en la imagen E.12.

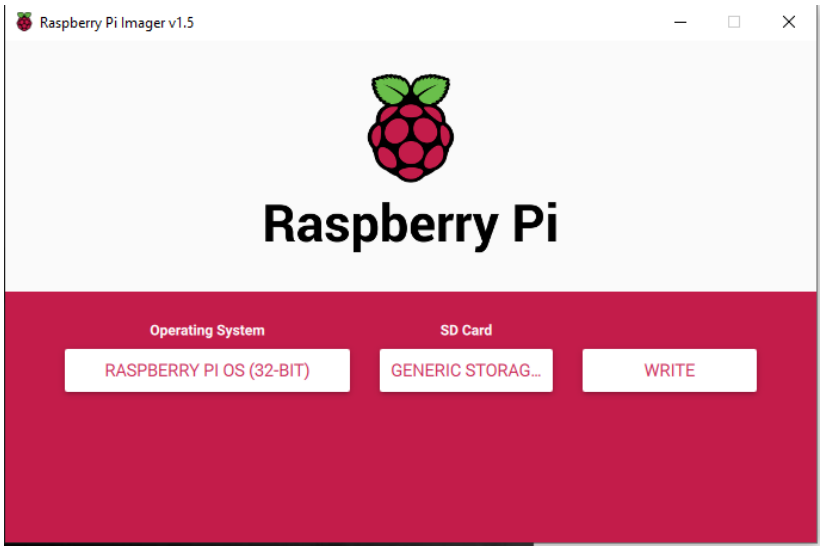


Figura E.12: Ventana selección imagen Raspbian.

6. Una vez termine la grabación de la tarjeta, procederemos a introducirla en la Raspberry Pi por la ranura destinada a ello en la parte inferior de la placa y podremos acceder a Sistema Operativo con la ayuda de una pantalla, un teclado y un ratón.

7. Debemos completar las siguientes ventanas de configuración:

Figura E.13: Ventana de configuración Raspbian 1.

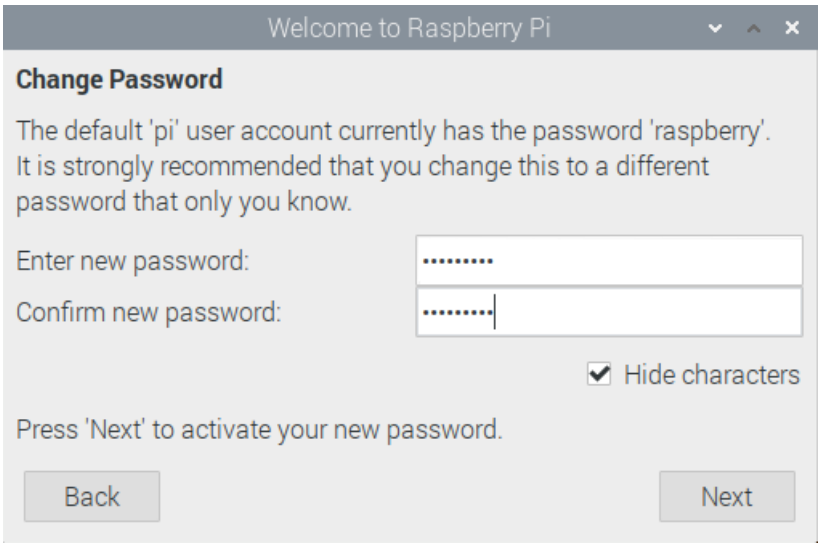


Figura E.14: Ventana de configuración Raspbian 2.

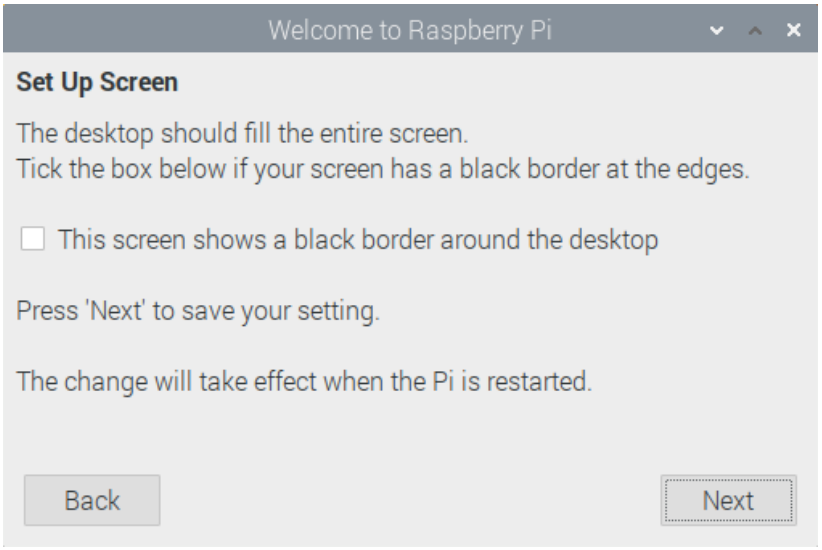


Figura E.15: Ventana de configuración Raspbian 3.

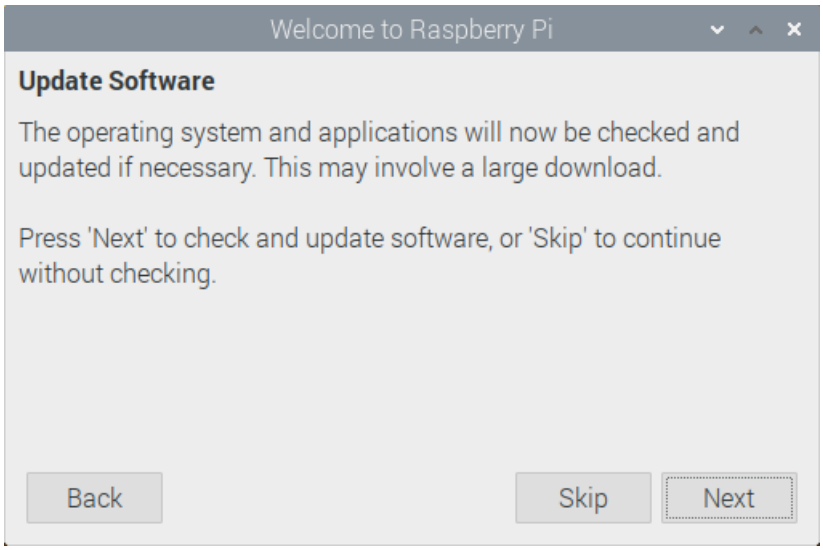


Figura E.16: Ventana de configuración Raspbian 4.

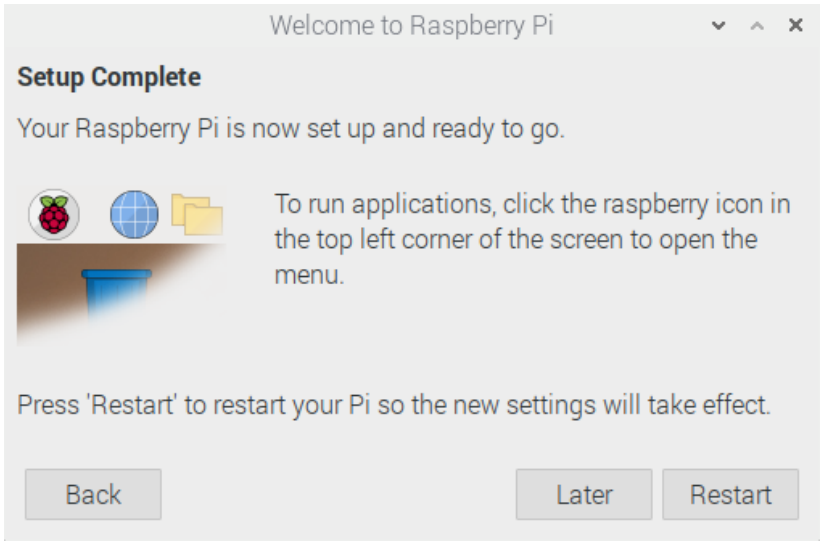


Figura E.17: Ventana de configuración Raspbian 5.

Una vez realizada la configuración básica vamos a habilitar VNC para poder conectarnos desde otros equipos de forma sencilla accediendo al menú de la imagen E.18. Este punto es opcional pero recomendable para evitar desplazarnos a la ubicación de la Raspberry Pi para hacer cualquier operación en local.

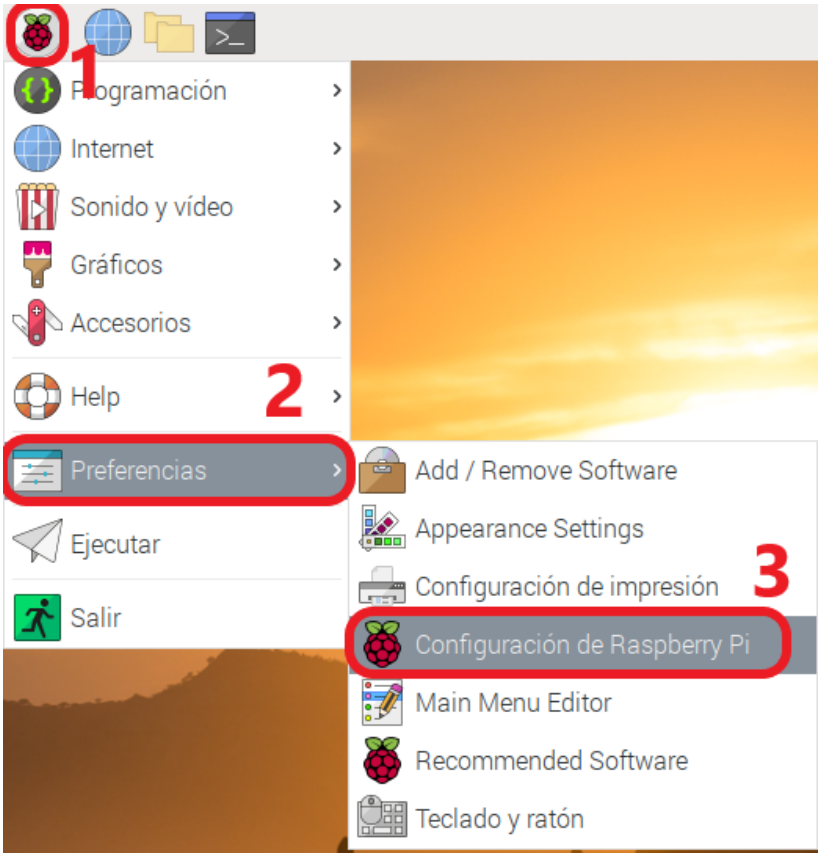


Figura E.18: Configura VNC 1.

Cuando abra la ventana, debemos acceder a la pestaña «interfaces» y activar VNC como podemos ver en la imagen E.19:

De esta forma podemos acceder desde VNC Viewer a nuestra máquina sabiendo la dirección IP, usuario y contraseña de ésta.

Descarga e instalación del software SDI

Para descargarnos e instalar el software SDI debemos seguir los siguientes pasos:

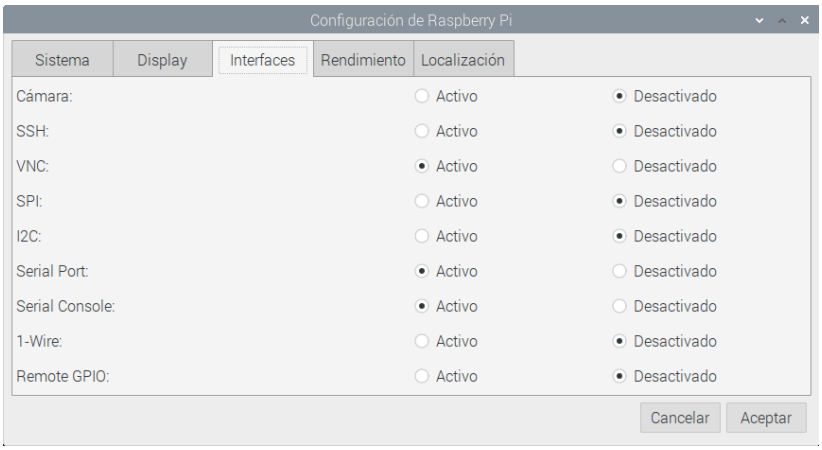


Figura E.19: Configura VNC 2.

1. Descargar la última release que exista en el repositorio de GitHub en el formato que se prefiera: https://github.com/davidelinformatico/TFG/releases
2. Para extraer del archivo .tar utilizar:
   `tar -xvf SistemaDomoticoInteligente_v1.0.tar`.
3. Para extraer del archivo .tar.gz utilizar:
   `tar xzvf SistemaDomoticoInteligente_v1.0.tar.gz`.

En este punto ya tendremos todos los archivos extraídos.

El siguiente paso es rellenar el archivo `config2.bot` con los tokens obtenidos anteriormente y la información de estancias donde se sitúan los periféricos y sus pines de control.

Ahora instalaremos el software. Para ello accederemos a **~/source/scripts/auto/** y corremos el archivo `setup.sh` con la siguiente sentencia:

```
0 sh setup.sh
```

En este momento se descargarán todos los elementos necesarios de internet y se instalarán.

Automatizado y puesta a producción

Uno de los puntos de la automatización es la generación de un demonio para nuestro bot de forma que se ejecutará con el inicio del sistema y podremos trabajar con el una sencilla sentencia:

```
sudo systemctl bot stop
```

```
sudo systemctl bot start
```

Para conseguirlo, he realizado los siguientes pasos:

Se debe generar el archivo de nuestro demonio en `lib/systemd/system/` con la extensión «.service»:

```
sudo nano /lib/systemd/system/bot.service
```

El contenido del archivo es:

Listing E.1: Modificaciones en el archivo /lib/systemd/system/bot.service.

```
[Unit]
Description=Lanza el bot de control domotico
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=pi
WorkingDirectory=/home/pi/source/TFG/scripts/bot/
ExecStart=/usr/bin/env python3 /home/pi/source/TFG/scripts/bot/bot.py

[Install]
WantedBy=multi-user.target
```

Después debemos actualizar los demonios con:

```
systemctl daemon-reload
```

Iniciar el demonio:

```
sudo systemctl start bot
```

Parar el demonio:

```
sudo systemctl stop bot
```

Estado del demonio, con el que podremos conocer el pid [3], que siempre es útil:

```
sudo systemctl status bot
```

[3] Identificador del proceso

Para incluirlo en el inicio de la máquina habría que moverlo a /etc/init.d y luego ejecutar:

```
0 sudo update-rc.d bot defaults
1 sudo systemctl daemon-reload
2 sudo systemctl enable bot
3 sudo systemctl start bot
```

Con esto tendríamos el software completamente instalado.

E.6. Inclusión de usuarios en el bot

El primer paso es buscar el bot por su nombre. En mi caso se llama **SDI_Domo_bot**. Una vez localizado, accedemos a él y le enviamos un mensaje. A lo que nuestro bot nos responderá con un mensaje con un número largo. Éste número largo es el que debemos incluir en el apartado de usuarios dentro del archivo `config2.bot`.

AAG: Reestructurar

Bibliografía

[1] Creative Commons. *Creative Commons by-sa 3.0*. [Página Oficial de la licencia].

[2] Creative Commons. Norma creative commons. [Página web Oficial, Noviembre de 2020].

[3] David. Presupuesto preliminar en milestone 3.

[4] David. Sonarcloud.

[5] ECMA. Javascript object notation. [Página web Oficial, Noviembre de 2020].

[6] Fritzing. Fritzing. [Página Oficial].

[7] GNU. *Free Software Foundation, Inc.* [Página web Oficial, Noviembre de 2020].

[8] GNU. *Licencia GPL3*. [Página Oficial de la licencia].

[9] Pepegreen. Tabla comparativa calibre awg.

[10] Python. Python software foundation. [Página web Oficial].

[11] SS. *Información General sobre cotización*, 2020. [Información General, Octubre de 2020].

[12] Telegram. Telegramapi. [Página Oficial].

[13] Telegram. Telegramapp. [Página Oficial].

[14] Wikipedia. Acalibre americano - awg.

[15] Wikipedia. Latex — wikipedia, la enciclopedia libre. [Página web, Noviembre de 2020].

[16] Wikipedia. Licencias software.

[17] Wikipedia. *Metodología Scrum*. [Página web, Noviembre de 2020].

[18] Wikipedia. Unicode-wikipedia.