Contenido

[app.py 3](#_Toc169202954)

[Alarm.py 8](#_Toc169202955)

[Clients.py 9](#_Toc169202956)

[ConvBelt.py 14](#_Toc169202957)

[Gates.py 16](#_Toc169202958)

[Lights.py 19](#_Toc169202959)

[Screen.py 22](#_Toc169202960)

[Supervisor.py 23](#_Toc169202961)

[Parte del front 31](#_Toc169202962)

[Views.py 31](#_Toc169202963)

[Urls.py 36](#_Toc169202964)

## app.py

**Ruta raíz (/)**

Devuelve un simple mensaje "Hello, World!" cuando se accede a la raíz del servidor.



**Control de Luces Individuales (/light)**

Texto

Descripción generada automáticamente

Recibe una solicitud POST con datos JSON para controlar el estado de una luz específica.

Llama a light\_action del butler con el número de luz y su estado (encendido/apagado).

**Control de Todas las Luces (/lights)**

Texto

Descripción generada automáticamente

Similar al anterior, pero controla todas las luces a la vez pasando -1 como número de luz.

**Activación de la Banda Transportadora (/transport\_band)**

Texto

Descripción generada automáticamente

Recibe una solicitud POST y activa la banda transportadora llamando a belt\_activate del butler.

**Estado de la Banda Transportadora (/transport\_band\_status)**

Texto

Descripción generada automáticamente

Devuelve el estado actual de la banda transportadora. Aquí está hardcodeado a 0.

**Control del Garaje (/Garage)**

Texto

Descripción generada automáticamente

Recibe una solicitud POST para controlar el estado del garaje llamando a gate\_action del butler.

**Estado del Garaje (/garage\_status)**

Texto

Descripción generada automáticamente

Devuelve el estado actual del garaje obteniéndolo del butler.

**Conteo Actual de Personas (/actual\_people)**

Texto

Descripción generada automáticamente

Devuelve el número actual de personas supervisadas por el butler.

**Conteo de Personas Total (/counter\_people)**

Texto

Descripción generada automáticamente

Devuelve el conteo total de personas (parece idéntico a get\_actual\_people en este ejemplo).

**Control de la Alarma (/alarm)**

Texto

Descripción generada automáticamente

Recibe una solicitud POST para configurar la alarma, pero no realiza ninguna acción específica en el butler.

**Estado de la Alarma (/alarm\_status)**

Texto

Descripción generada automáticamente

**Ejecución de la Aplicación**

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Ejecuta la aplicación Flask en modo de depuración cuando se ejecuta directamente el script.

## Alarm.py

Primero, importa la función sleep del módulo time y GPIO del modulo RPi.GPIO. Esta función se utiliza para hacer una pausa en la ejecución del programa durante un número especificado de segundos.

Luego, se configura el modo de los pines GPIO a GPIO.BOARD, lo que significa que los números de los pines se refieren a los números de los pines del conector físico de la Raspberry Pi.

Después, se desactivan las advertencias con GPIO.setwarnings(False). Esto es útil para evitar mensajes de advertencia si se ha configurado un pin de una manera particular en otro lugar de nuestro código.

Se define una variable pin con el valor 40, que representa el pin que se va a usar para controlar el buzzer.

La función GPIO.setup se utiliza para configurar el modo del pin como salida (GPIO.OUT). Además, se establece un estado inicial para el pin, que en este caso es GPIO.LOW (el buzzer está apagado).

Finalmente, se define una función turn\_on\_buzzer\_with\_timmer que toma un argumento time (el tiempo en segundos que el buzzer debe estar encendido). Dentro de esta función, se utiliza GPIO.output para cambiar el estado del pin a GPIO.HIGH (enciende el buzzer), se imprime el estado del pin, se hace una pausa por el tiempo especificado, luego se cambia el estado del pin a GPIO.LOW (apaga el buzzer) y se imprime el estado del pin.

Texto

Descripción generada automáticamente

## Clients.py

Primero, se establece el modo de numeración de los pines a GPIO.BOARD para usar la numeración física de la placa. Luego, se desactivan las advertencias con GPIO.setwarnings(False).

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Se define una lista pin\_arr que contiene los números de los pines que se van a utilizar. Para cada pin en pin\_arr, se configura como salida (GPIO.OUT) y se inicializa en estado bajo (GPIO.LOW).

Se define una lista binary\_nums que contiene representaciones binarias de los números del 0 al 9.

Texto

Descripción generada automáticamente

La función convert\_bin(num) toma un número entero como entrada y devuelve su representación binaria como una lista de 1s y 0s. Si el número es mayor o igual a 10, devuelve la representación binaria del número 9.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

La función convert\_num(binary) toma una lista de 1s y 0s como entrada y devuelve el número entero correspondiente. Si no se encuentra una coincidencia en binary\_nums, devuelve -1.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

La función show\_binary(num) toma un número entero como entrada, limpia el display con clean\_display(), convierte el número a binario con convert\_bin(num), y luego enciende los pines correspondientes en la placa Raspberry Pi.

Texto

Descripción generada automáticamente

Finalmente, la función enter\_exit\_client(entrada) implementa una lógica para determinar si un cliente entra o sale. Utiliza dos sensores, A y B. Si el sensor B se activa primero, se asume que un cliente ha entrado. Si el sensor A se activa primero, se asume que un cliente ha salido. En ambos casos, los sensores se resetean a False después de detectar una entrada o salida. Si no se detecta ninguna actividad, la función devuelve 0.

Texto

Descripción generada automáticamente

## ConvBelt.py

Primero, se configura el modo de numeración de los pines a GPIO.BOARD con GPIO.setmode(GPIO.BOARD). Esto significa que los números de los pines se corresponden con los números de los pines en el conector de la Raspberry Pi.

Luego, se desactivan las advertencias con GPIO.setwarnings(False). Esto es útil para evitar mensajes de advertencia si se ha configurado un pin en un script anterior.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Después, se definen tres pines: pinLV (luz verde), pinLR (luz roja) y pinM (motor), que corresponden a los pines 8, 10 y 12 respectivamente.

A continuación, se configuran estos pines como salidas con GPIO.setup(). El parámetro initial define el estado inicial del pin. En este caso, el pin de la luz verde y el motor se inicializan en estado bajo (GPIO.LOW), lo que significa que están apagados, y el pin de la luz roja se inicializa en estado alto (GPIO.HIGH), lo que significa que está encendido.

Texto

Descripción generada automáticamente

El código también define dos funciones: start\_motor() y stop\_motor(). Estas funciones intentan encender y apagar el motor respectivamente, cambiando el estado de los pines correspondientes. Si ocurre algún error al intentar cambiar el estado de los pines, se imprime un mensaje de error.

En start\_motor(), se intenta poner el pin del motor y la luz verde en estado alto, y la luz roja en estado bajo. Esto enciende el motor y la luz verde, y apaga la luz roja.

En stop\_motor(), se intenta poner el pin del motor y la luz verde en estado bajo, y la luz roja en estado alto. Esto apaga el motor y la luz verde, y enciende la luz roja.

Texto

Descripción generada automáticamente

## Gates.py

Primero, se importa la función sleep del módulo time. Esta función se utiliza para hacer una pausa en la ejecución del programa durante un número especificado de segundos.

Luego, se configura el modo de los pines GPIO a GPIO.BOARD para referirse a los pines por el número del pin del conector físico.

La línea GPIO.setwarnings(False) se utiliza para desactivar las advertencias de GPIO.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

La función put\_angle toma un ángulo como entrada y calcula un valor de ciclo de trabajo para un servo. Si el ángulo está fuera del rango de 0 a 180 grados, la función devuelve False. De lo contrario, calcula un valor de ciclo de trabajo proporcional al ángulo y lo devuelve.

Texto

Descripción generada automáticamente

Las funciones open\_gates y close\_gates se utilizan para abrir y cerrar las compuertas, respectivamente. Ambas funciones configuran el pin 32 como salida, inician un objeto PWM en el pin 32 con una frecuencia de 50Hz, y luego cambian el ciclo de trabajo del PWM.

En open\_gates, el ciclo de trabajo se establece en 3, lo que puede mover el servo a una posición que abre las compuertas. En close\_gates, el ciclo de trabajo se establece en 12, lo que puede mover el servo a una posición que cierra las compuertas.

Después de cambiar el ciclo de trabajo, el programa se pausa durante 0.25 segundos para dar tiempo al servo para moverse, y luego se detiene el PWM.

Texto, Chat o mensaje de texto

Descripción generada automáticamente

## Lights.py

Primero, se configura el modo de los pines GPIO a GPIO.BOARD, lo que significa que los números de los pines se refieren a los números de los pines físicos de la placa. Luego, se desactivan las advertencias para evitar que se muestren mensajes de advertencia innecesarios.

Se define una lista llamada lights que contiene los números de los pines que se utilizarán para las luces. Luego, en un bucle for, se configura cada pin de luz como salida (GPIO.OUT) y se inicializa en estado bajo (GPIO.LOW), lo que significa que las luces estarán apagadas al inicio.

Texto

Descripción generada automáticamente

La función turn\_light(number, on) se utiliza para encender o apagar una luz específica. El primer parámetro es el índice en la lista lights para la luz que se quiere controlar, y el segundo parámetro es un booleano que indica si la luz debe estar encendida (True) o apagada (False). Dentro de la función, se verifica si el número proporcionado está dentro del rango válido. Si es así, se cambia el estado de la luz correspondiente. Si el número es 10, se llama a la función turn\_all\_lights(on) para encender o apagar todas las luces. Si el número no es válido, se imprime un mensaje de error.

Texto

Descripción generada automáticamente

La función turn\_all\_lights(on) se utiliza para encender o apagar todas las luces. Recibe un parámetro booleano que indica si las luces deben estar encendidas (True) o apagadas (False). Dentro de la función, se recorre la lista lights y se cambia el estado de cada luz según el valor del parámetro on.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

## Screen.py

Primero, importa la clase LCD del módulo lib.LCD y la renombra como lcd para su uso en el código. Luego, configura el modo de los pines GPIO de la Raspberry Pi para que se refieran por su número de pin físico en la placa (esto es lo que hace GPIO.setmode(GPIO.BOARD)). Desactiva las advertencias de GPIO con GPIO.setwarnings(False).

Después, crea una instancia de la clase lcd con los argumentos 2 y 0x27. El 2 indica que se va a usar para una versión de Raspberry Pi 2 en adelante. El 0x27 es probablemente la dirección I2C del LCD.

Texto

Descripción generada automáticamente

La función show\_text(action) toma una cadena de texto como entrada, la divide en palabras y las muestra en la pantalla LCD. Primero, borra cualquier texto existente en la pantalla con lcd.clear(). Luego, divide la cadena de entrada en palabras individuales con action.split(" ").

El código luego itera sobre cada palabra en la lista de palabras. Si la longitud de la palabra y el texto actual en row1 es menor o igual a 15 y row1 aún no está llena (rowIsFull es False), añade la palabra a row1. Si row1 está llena o la adición de la palabra haría que row1 exceda 15 caracteres, cambia rowIsFull a True y comienza a añadir palabras a row2.

Finalmente, muestra row1 y row2 en la pantalla LCD en las líneas 1 y 2 respectivamente con lcd.text(row1, 1) y lcd.text(row2, 2).

Texto

Descripción generada automáticamente

## Supervisor.py

El código comienza configurando el modo de los pines GPIO y desactivando las advertencias. Luego, se configuran tres pines como entradas, que se utilizarán para leer los estados de los sensores.

Texto

Descripción generada automáticamente

La función safe\_exit se define para manejar las señales SIGTERM y SIGHUP, que son señales del sistema operativo que indican que el programa debe terminar. Cuando se recibe una de estas señales, el programa se cierra con un código de salida de 1.

Texto

Descripción generada automáticamente

La clase butler es el núcleo del sistema de supervisión. En su método \_\_init\_\_, inicializa un objeto LCD, y establece contadores para los clientes, el estado de las puertas y las luces.

Pantalla de computadora con números

Descripción generada automáticamente con confianza media

El método light\_action controla las luces. Si se pasa -1 como el número de luz, todas las luces se encienden o apagan. De lo contrario, se enciende o apaga la luz especificada.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

El método gate\_action controla las puertas. Si se pasa 1 como estado, las puertas se cierran. De lo contrario, las puertas se abren.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

El método belt\_activate inicia el motor de la cinta transportadora, espera 5 segundos y luego lo detiene.

Interfaz de usuario gráfica

Descripción generada automáticamente

El método messages muestra mensajes en el LCD. Los mensajes se actualizan cada 2 segundos.

Texto

Descripción generada automáticamente

El método clients\_counter cuenta el número de clientes. Utiliza dos sensores para determinar si un cliente está entrando o saliendo, y ajusta el contador de clientes en consecuencia.

Texto

Descripción generada automáticamente

El método perimeter\_alarm activa una alarma si se detecta una entrada en el perímetro.

Texto

Descripción generada automáticamente

Finalmente, el método start\_supervisor inicia tres hilos para ejecutar los métodos messages, clients\_counter y perimeter\_alarm en paralelo.

Texto

Descripción generada automáticamente

# Parte del front

Solo se van a explicar cosas sencillas, para esto se utilizo django y flask (es el archivo app.py que se explico al principio)

## Views.py

Las clases MainData y Light son estructuras de datos. MainData almacena información sobre el número total de clientes que entran y salen, el estado de la cinta transportadora, la puerta y la alarma. Light representa una luz individual con un identificador y un estado.



La función loadData se utiliza para cargar datos desde varias URL, cada una de las cuales devuelve un estado o un recuento de personas. Los datos se almacenan en la instancia global de MainData llamada data.

Texto

Descripción generada automáticamente

La función test es una vista que carga los datos utilizando loadData y luego renderiza una plantilla HTML llamada 'test.html', pasando los datos de las luces y la instancia de MainData.

Texto

Descripción generada automáticamente

Las funciones set\_all\_lights\_states, set\_lights\_states, set\_transport\_band y set\_garage son vistas que manejan las solicitudes POST para cambiar el estado de las luces, la cinta transportadora y la puerta, respectivamente. Cada una de estas funciones recibe una acción ('on' o 'off') y realiza una solicitud POST a una URL específica para cambiar el estado correspondiente. Si la solicitud POST es exitosa, se renderiza la plantilla 'test.html' con los datos actualizados. Si la solicitud POST falla, se devuelve una respuesta JSON con un mensaje de error.

Texto

Descripción generada automáticamente

Por último, se crea una lista de luces lights con 10 luces, todas inicialmente apagadas, y una instancia de MainData llamada data con todos los valores inicialmente en cero o False.

Texto

Descripción generada automáticamente

## Urls.py

En las primeras líneas, se importan varias vistas desde webApp.views. Las vistas son funciones que toman una solicitud web y devuelven una respuesta. En este caso, las vistas importadas son test, set\_all\_lights\_states, set\_lights\_states, set\_transport\_band y set\_garage.

Texto

Descripción generada automáticamente

A continuación, se define una lista llamada urlpatterns. Esta lista es una parte crucial de la configuración de URL de Django. Cada elemento de la lista es una llamada a la función path(), que crea una instancia de URLPattern.

La función path() toma tres argumentos: una cadena que define la ruta URL, una vista que se invocará cuando se solicite esa URL, y opcionalmente un nombre para la ruta que se puede usar para referirse a ella en otras partes del código.

Por ejemplo, path('set\_all\_lights\_states/', set\_all\_lights\_states, name='set\_all\_lights\_states') define una ruta URL que, cuando se solicita, invocará la vista set\_all\_lights\_states. También se le da un nombre, set\_all\_lights\_states, que se puede usar para referirse a esta ruta en otras partes del código, como en las plantillas de Django.

En resumen, este código define las rutas URL para una aplicación web Django, mapeando URL específicas a sus correspondientes vistas.

Texto

Descripción generada automáticamente