APbackend operates on top of ASGI (Asynchronous Server Gateway Interface). Daphne is a pure Python ASGI server. The WSGI version is Gunicorn. Django is a web framework that supports ASGI, which is also supported by web hosting cloud service, Heroku.

This project employs a website, a webhook connection, and a websocket connection.

The website is primarily used for user registration, users would sign in using their google email, which would then be stored in the server. The website then will allow the user to enter a UID of their product, which would be attached to their email in the sqlite3 database.
Besides email and UID, the database also stores internal communications channels and status information of each user. The internal information is used by the server to facilitate google home action commands and internal commands going to the hardware device.

Communications between google home action commands and this server is accomplished through webhook messages. After the user has logged their product UID into the website, the user can then use Home Control app on their phone to add the device to their home control panel. Communication between google home action and the server are based on various google defined json object, like SYNC and QUERY commands. Initially, Home Control requests a SYNC, where the server will respond with the device's profile and capabilities. Normal operations of turning on and off the device, along with changing speed of the device, are done through EXECUTE commands. QUERY commands are used to ping the server for device status.

The backend communication between the server and device goes through a websocket connection. The websocket connection is a TCP connection that is needed before data transmission can occur between server and device. It is designed to automatically connect to the internet with user defined information (WIFI network ID and password). The device will establish connection with the server's websocket service and will transmit the UID of the device to the server. If the server does not see the UID in its database (no user has placed the UID in the database) then the server will close the connection. The device will periodically 'ping' the server with the UID. When the server finds a UID match within the database, it will reply with the same UID as an initial handshake. At this point, the server will be able to request status or commands to the device, as needed from the webhook side of the server.

The webhook connection is short lived, and message based, while the websocket connection stays open for as long as there is an internet connection. Communication between webhook and websocket connection is facilitated using Django web framework's Channels project. Django Channels allows for asynchronous communication between webhook and websocket. This is required since the user will not always send commands to the server, but the device has to be ready for a command at any time. It is based on an event-driven architecture, which means, at the application layer, the application(webhook) does not wait for a command. A webhook message will trigger the application to process the command, a consumer be created to 'consume' the command by using a channel layer to talk to the correct websocket connection,

and terminate after returning the webhook request. Knowing Which ‘channel’ is the correct one is done through storing channel names in the sqlite3 database.

Django Channels changes Django so that it can use Websocket, and other async support to the server. Channels serves as a layer below Django that provides asynchronous handling of connections and sockets. Channel layers is the key to the operation of this server. ASGI applications are instantiated once per socket. One application instance can talk to another thorugh direct events. So a webhook application can talk to a websocket application even if they do not instantiate at the same time.

https://www.websequencediagrams.com/
title Login Sequence (website)

User->+Django(Views.py): http request
note right of Django(Views.py): Generate dynamic webpage
Django(Views.py)->-User: http response
User->+Django(Views.py): http request (login btn)
note right of Django(Views.py): forward to auth0\nlogin page
Django(Views.py)->-User: http response(login URL)
User->+auth0: User logs in to auth0 using\nGoogle authentication
note right of auth0: auth0 processes login information,\nextracts email from Google, returns to site
auth0->-User: forward callback URL that points to website
User->+Django(Views.py): Using callback URL + logged in details
note right of Django(Views.py): Get user DB entry based on email\nIf email does not exist, create DB entry
note right of Django(Views.py): Generate dynamic webpage,\nshows form to enter device UID
Django(Views.py)->-User: http response
User->+Django(Views.py): http response(POST) with UID
note right of Django(Views.py): Checks UID for validity,\nstores UID in user DB entry
Django(Views.py)->-User: http response

Title Device Login Sequence (websocket)
Device->+Django(websocket.py): Establish connection to websocket service
Note right of Device: Device will repeatedly try to connect to websocket service,\nand send UID upon successful connection
Note right of Django(websocket.py): Upon receiving UID, it will find the existing UID in DB\n(User should have registered it with their email),\nmatch UID with email account, store websocket redis\nchannel name in DB.
Django(websocket.py)->-Device: Reply with device UID to confirm account.
Note right of Device: websocket connection stays open for as long as device is powered.

Title Query Sequence (webhook + websocket + device)

Google Home->+Django(webhook.py): json object in request
Note right of Django(webhook.py): json object has authentication bearer
Opt if bearer is not found in DB
Note right of Django(webhook.py): send bearer to Auth0 service to retrieve email\nuse email to find existing user account\nupdate bearer in DB
End
Note right of Django(webhook.py): Get UID from DB entry
Opt if UID not found in entry
Note right of Django(webhook.py): error, user did not enter UID in website first,\nreturns error intent
End
Note right of Django(webhook.py): found UID, check json object's command\n(query in this instance)
Note right of Django(webhook.py): use websocket channel name in DB
Django(webhook.py)->+Django(websocket.py): use websocket channel name in DB\nto send query request to websocket\nusing channel layer
Note right of Django(websocket.py): use websocket connection to\nsend query request to device
Django(websocket.py)->+device: sends command over\nwebsocket connection
note right of device: polls I/O ports in device
device->-Django(websocket.py): returns device statuses\nto websocket service
Note right of Django(websocket.py): parses device message,\nsends message back\nto webhook service
Django(websocket.py)->-Django(webhook.py): callback function: places device message\nin intent fields, place intent into jason object
Django(webhook.py)->-Google Home: sends json object response

Github source
https://github.com/Hyperian428/Django-Google-Home-Actions

auth0.com recovery code G5S4252STF6JNMZSEPQNJP54