Universitat de Lleida

Treball de final de grau

My Meteorological Station

Gerard Farré i Gomez

supervisat per

Fernando Cores Prado

Andreu Ibàñez Perales

8 de setembre de 2017

Índex

1 Introducció **9**

1.1 Objectius del treball 12

1.2 Tasques a realitzar 14

1.2.1 Anàlisis 14

1.2.2 Disseny 14

1.2.3 Implementació 14

1.2.4 Validació 15

1.3 Memòria 15

1.4 Planificació temporal de les tasques a realitzar 15

2 Estat de l'art **17**

2.1 Raspberry Pi 17

2.1.1 Sistema operatiu 20

2.2 Beacons & Physical Web 20

2.3 Liquid Galaxy 22

2.3.1 Google Earth 23

2.3.2 KML (Keyhole Markup Language) 24

2.3.3 Comunicació amb el Liquid Galaxy 25

2.4 Google Assistant (Actions on Google, Api.AI) 25

2.5 Node.js 31

2.5.1 Mòduls i NPM 32

2.5.2 Package.json 33

2.6 Firebase 34

2.6.1 Authentication 34

2.6.2 Realtime database 36

2.7 Twitter 38

3 Anàlisis **40**

3.1 Anàlisis de requeriments 40
3.1.1 Requeriments funcionals 40
3.1.2 Requeriments no funcionals 40
3.1.3 Diagrama de casos d'ús 41
3.1.4 Especificació dels casos d'ús 42
3.1.5 Diagrames de seqüencia 46
3.2 Anàlisis de costs 49

4 Disseny **51**

4.1 Base de dades 51
4.2 Fonts de dades 53
4.2.1 Sensors 53
4.2.2 API (Weather Underground) 55
4.3 Enviament i recepció de dades 58
4.4 Visualització de les dades 58
4.4.1 Interfície web 58
4.4.2 Physical web 60
4.4.3 Liquid Galaxy 60
4.4.4 Google Assistant 62
4.4.5 Twitter 64

5 Implementació **65**

5.1 Estructura del projecte 65
5.2 Servidor 65
5.3 Firebase 65
5.3.1 Lectura de dades 67
5.3.2 Escriptura de dades 68
5.4 Interfície 69
5.4.1 Google Sign in 70

5.5 Estacions 70
5.5.1 Interfície d'usuari 70
5.5.2 Còpia de la clau Maps API 73
5.5.3 Configuració estació - servidor 73
5.6 Balises 76
5.6.1 Interfície d'usuari 76
5.6.2 Escurçador de URL 79
5.6.3 Redirecció de URL 79
5.7 Liquid Galaxy 80
5.7.1 Comunicació 80
5.7.2 Emmagatzemament de la configuració del Liquid Galaxy 81
5.7.3 Generació de KML 82
5.8 Google Assistant 84
5.9 Twitter 86

6 Conclusions i línies obertes **89**

6.1 Conclusions 89
6.2 Línies obertes 89

Índex de figures

1 Sensor de temperatura i humitat, model DHT22 9

2 Placa *Raspberry Pi* 3 B 9

3 Escriptori de la distribució *Raspbian* 10

4 Balisa *Physical Web* 10

5 Instal·lació *Liquid Galaxy* de cinc monitors 11

6 Dispositiu *Google Home* que conté el *software Google Assistant* 11

7 Diagrama d'interacció del cas d'ús de *Physical Web* 12

8 Diagrama d'interacció del cas d'ús de *Liquid Galaxy* 13

9 Diagrama d'interacció del cas d'ús de *Google Assistant* 13

10 Diagrama de *Gantt* de les tasques a realitzar 16

11 Placa Raspberry Pi 3B 17

12 Consola arcade realitzada amb la Raspberry Pi 18

13 Dron amb una Raspberry Pi de controladora 18

14 Numeració dels pins GPIO 19

15 Balisa Physical Web 21

16 Casos d'us de Physical Web 21

17 Web de configuració de la balisa 22

18 Exemple de notificació physical web en un smartphone Android 22

19 Instal·lació *Liquid Galaxy* real de set monitors 23

20 Pantalla principal del software Google Earth. 24

21 Dispositius que executen el software Google Assistant (Google Home i Google Pixel). . 25

22 Intent bàsic d'un agent de Api.ai. 26

23 Enviar l'agent com a aplicació test. 27

24 Prova de l'agent en un dispositiu real. 28

25 Secció per introduir la URL del webhook al projecte. 29

26 Intent de prova per testejar el webhook. 29

27 Prova de l'agent en un dispositiu real. 31
28 Pantalla de selecció de proveïdor de Firebase 35
29 Captura de la base de dades de Firebase 36
30 Pàgina de detalls d'una aplicació de Twitter 39
31 Diagrama de casos d'ús 42
32 Diagrama de seqüencia entre la comunicació Raspberry Pi - Servidor 46
33 Diagrama de seqüencia Physical web 47
34 Diagrama de seqüencia Liquid Galaxy 47
35 Diagrama de seqüencia Google Assistant 48
36 Estructura de la base de dades 51
37 Numeració dels pins GPIO de la Raspberry Pi 2/3 54
38 Sensor DHt22 54
39 Sensor BMP180 54
40 Esquema de la Raspberry Pi amb els senors connectats 55
41 Enviament, recepció i emmagatzemament de les dades 58
42 Pantalla principal de l'aplicació web 59
43 Pantalla principal de l'aplicació web en mode smartphone 59
44 Notificació Physical Web 60
45 Menú desplegable per al Liquid Galaxy 61
46 Globus d'una estació mostrada al Liquid Galaxy 61
47 Conversa amb Google Assistant preguntant per un valor concret d'una estació 63
48 Botó per rebre la configuració de firebase del client 66
49 Botó per rebre la configuració de Firebase en administrador 66
50 Apartat de creació d'una nova estació 71
51 Formulari de creació d'una nova estació 72
52 Apartat d'administració de les estacions 72
53 Diàleg de confirmació per eliminar una estació 73
54 Secció Physical Web 76

55 Formulari de creació d'una balisa 77
56 Pantalla que retorna la URL un cop creada la balisa 77
57 Utilitat per emparellar la balisa mitjançant bluetooth 78
58 URL introduïda i balisa actualitzada correctament 78
59 Secció d'administració de les balises 79
60 Secció de configuració dels valors del Liquid Galaxy 82
61 Entitat weather-values a l'API.AI 84
62 Pagina de detall de l'intent a l'API.AI 84
63 Pagina d'administració de claus de l'aplicació de Twitter 87

Listings

1 Exemple de recepció de dades del sensor DHT22 20

2 Exemple XML 24

3 Exemple de webhook 30

4 Exemple aplicació Node.js 31

5 Exemple d'encaminament bàsic en express utilitzant el mètode GET. 33

6 Exemple d'ús de Body-parser. 33

7 Exemple del contingut del fitxer package.json 34

8 Exemple creació d'usuari en Firebase 35

9 Exemple d'inici de sessió d'usuari en Firebase 35

10 Exemple d'observador del canvi d'estat de l'usuari en Firebase 35

11 Exemple de recepció de dades d'un usuari en Firebase 36

12 Exemple d'estructura de la realtime database de Firebase 36

13 Exemple de seguritat desactivada en la base de dades de Firebase 37

14 Exemple de restricció d'usuaris no autenticats en la base de dades de Firebase 37

15 Exemple de restricció d'usuaris no propietaris de les dades en la base de dades de Firebase 37

16 Exemple de inserció de dades en la base de dades de Firebase 37

17 Resultat de la inserció de dades de la realtime database de Firebase 38

18 Exemple de lectura de dades en la base de dades de Firebase 38

19 Regles de seguretat de la base de dades 52

20 Resposta de l'API Wunderground a una petició 56

21 Codi d'arrencada del servidor 65

22 Contingut del fitxer firebase-config.json 65

23 Carrega i inicialització de la configuració de Firebase 67

24 Carrega i inicialització de la configuració de Firebase en administrador 67

25 Funció per rebre les dades de totes les estacions 67

26 Crida a la funció de lectura de totes les estacions 67

27 Funció que retrona les dades d'una estació 68

28 Funció que retrona les estacions que pertanyen a un usuari 68
29 Funció que guarda els valors dels sensors a la base de dades 68
30 Renderització de la pagina inicial 69
31 Funció que escolta els canvis d'autenticació 70
32 Contingut del fitxer maps-key.json 73
33 Contingut de la plantilla del servei dels sensors 74
34 Mètodes POST del servidor per rebre les dades 75
35 Funcions per guardar els valors dels sensors i comprovar que la estació sigui vàlida . . 75
36 Funció per escurçar la URL 79
37 Mètode que gestiona la notificació de la balisa 79
38 Vista de la notificació Physical Web 80
39 Funció per desplaçar-te a una localització al Liquid Galaxy 80
40 Escriu la URL del fitxer KML a kmls.txt 81
41 Envia kmls.txt al Liquid Galaxy 81
42 Emmagatzemament de les dades usant node persist 82
43 Recuperació de dades emmagatzemades localment usant node persist 82
44 Plantilla que mostra un globus en un punt 83
45 Recull les dades de Firebase i crida la funció per generar el KML 83
46 Renderitza la plantilla KML amb les dades rebudes 83
47 Mètode POST que gestiona el webhook 85
48 Funció que retorna el valor d'una estació a Google Assistant 85
49 Funció per Google Assistant que mostra un globus al Liquid Galaxy 85
50 Contingut del fitxer twitter-config.json 86
51 Inicialització de l'aplicació de Twitter 87
52 Funció que realitza la publicació de les dades a Twitter 87
53 Executa l'actualització a Twitter cada 24 hores 88

1 Introducció

A causa de la gran quantitat de dispositius i recursos disponibles actualment, com per exemple diferents tipus de sensors i micro computadors a un preu assequible, és possible monitorar l'estat meteorològic d'un lloc concret realitzant la teva pròpia estació meteorològica, sense dependre d'una estació meteorològica pròxima, on la precisió dels resultats dependrà de la seva distancia.

Hi ha diferents tipus de sensors que es poden utilitzar per mesurar les condicions meteorològiques. Alguns d'aquests sensors són, tal com podem observar en la Figura 1, el termòmetre, per mesurar la temperatura, l'higròmetre, per mesurar la humitat, el baròmetre per mesurar la pressió atmosfèrica, l'anemòmetre, per mesurar la velocitat del vent, etcètera.

Figura 1: Sensor de temperatura i humitat, model DHT22

Per gestionar tots els sensors es pot utilitzar la placa *Raspberry Pi*[10], tal com es mostra a la Figura 2. La *Raspberry Pi* és un *Single-Board Computer*[1] molt econòmic i de baix consum, altament configurable, basat en l'arquitectura *ARM*[2] i que permet instal·lar múltiples sistemes operatius. En aquest projecte s'utilitzarà *Raspbian* (Figura 3), la distribució *GNU/Linux* suportada oficialment basada en *Debian*[3].

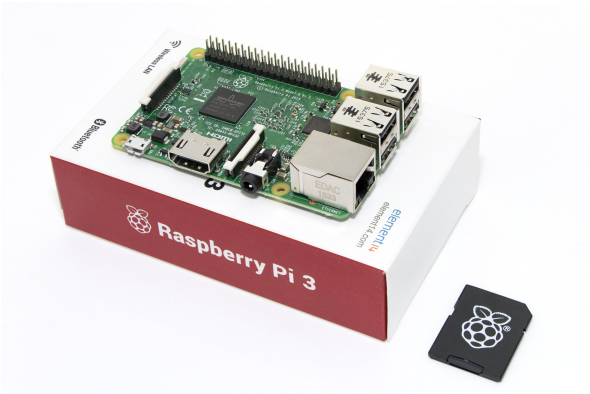

Figura 2: Placa *Raspberry Pi* 3 B

[1]És una placa de circuit amb tots els components integrats necessaris per a formar un computador

[2]És una família de microprocessadors *RISC* que requereixen una quantitat menor de transistors i per tant obté una reducció dels costos, calor i energia.

[3]És una distribució de *GNU/Linux* de programari lliure, no comercial.

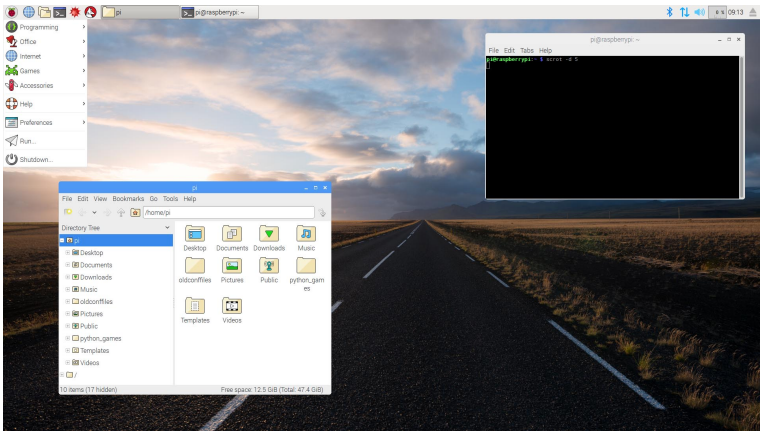

Figura 3: Escriptori de la distribució *Raspbian*

En els últims temps, s'han invertit molts recursos a desenvolupar noves formes d'interacció amb les dades i les aplicacions, donant pas a nous tipus de dispositius, com les *Google Glass*[4], la realitat virtual[5], entre d'altres. La part més important d'aquest projecte, doncs, és en la presentació de les dades a l'usuari final, que es realitzarà utilitzant tres tecnologies diferents.

La primera forma de presentar les dades serà mitjançant la tecnologia *Physical Web*[9].
Aquesta tecnologia consta d'unes balises *Bluetooth*[6] (també anomenades beacons), tal com s'observa en la Figura 4, que fent ús d'aquest protocol, emeten una *URL*[7]. Quan l'usuari s'apropa a la balisa i entra a la seva zona de transmissió, rep l'*URL* permetent consultar el seu contingut.

Figura 4: Balisa *Physical Web*

La segona manera de representar aquesta informació és d'una forma més visual i interactiva. Aquesta experiència ens l'ofereix el projecte *Liquid Galaxy* [6][7], que podem observar en la Figura 5.

El projecte *Liquid Galaxy* és un sistema *open-source* desenvolupat per enginyers de *Google* que ofereix una instal·lació de *Google Earth* en format multi pantalla, on les pantalles estan col·locades estratègicament en forma de semicercle creant una experiència d'usuari panoràmica, envoltant i immersiva.

[4]Són unes ulleres de realitat augmentada desenvolupades per *Google*.
[5]Simulació virtual que dóna la sensació de ser una representació real.
[6]És un protocol estàndard per intercanviar informació entre dispositius pròxims utilitzant tecnologia sense fils.
[7]És una adreça formada per caràcters alfanumèrics que permet identificar un recurs en xarxa.

D'aquesta manera, l'usuari podrà visualitzar les dades en el mapa, situades en la localització exacta de la nostra estació meteorològica.

Figura 5: Instal·lació *Liquid Galaxy* de cinc monitors

L'ultima forma de què l'usuari pugui rebre les dades serà mitjançant peticions de veu, utilitzant la tecnologia *Google Assistant*[3]. Aquesta tecnologia està incorporada al dispositiu *Google Home* (Figura 6) i en alguns *smartphones* amb el sistema operatiu *Android*[8].
L'usuari podrà mantenir una conversa natural amb l'agent[9], preguntant-li per qualsevol dada que pugui retornar l'estació meteorològica i essent respost en conseqüència.

Figura 6: Dispositiu *Google Home* que conté el *software Google Assistant*

El fet que no es puguin aconseguir tots els sensors meteorològics existents per la realització d'aquest projecte, implica que es rebran les dades restants d'altres serveis i *API*s[10][11] meteorològiques per acabar de completar totes les dades i així tenir una informació més completa.

En els següents capítols s'aprofundirà els detalls del sistema i de les seves tecnologies enllaçades.

[8]És un sistema operatiu per a mòbils desenvolupat per Google que utilitza com a base el *kernel* de *Linux*.

[9]És la forma d'anomenar una aplicació que rep i respon informació en *Google Assistant*.

[10] *Application Programming Interface*, és un una interfície que ofereix cert sistema i que defineix un conjunt de mètodes per ser utilitzat per un altre programa com una capa d'abstracció.

1.1 Objectius del treball

Tal com hem comentat en la secció d'introducció, l'objectiu del projecte consisteix a crear una estació meteorològica casolana, usant una *Raspberry Pi*, dotada d'interacció amb les tecnologies *Physical Web*, *Liquid Galaxy* i *Google Assistant*.

La *Raspberry Pi* serà l'encarregada de rebre la informació dels sensors meteorològics i de les *API*s, i farà la funció de servidor, proveint una *URL* que contindrà totes les dades actuals de la nostra estació.

En la part de Physical Web, es situaran diverses balises *Physical Web* repartides en en la zona de la localització de l'estació. Cada una d'aquestes balises realitzara la funció d'emetre la URL proporcionada per la *RPi*, dins del seu rang d'emissió, tal com mostra la Figura 7.
Quan l'usuari entri dins del rang d'emissió de la balisa, rebrà una notificació mostrant la *URL*, que permetrà accedir a les dades de la *RPi* i rebre tot el seu contingut.
La balisa emetrà una *URL* fixa que apuntarà a una altra, utilitzant la tècnica de redirecció d'*URL*, permetent modificar el contingut de la balisa remotament sense necessitat de modificar la seva configuració.

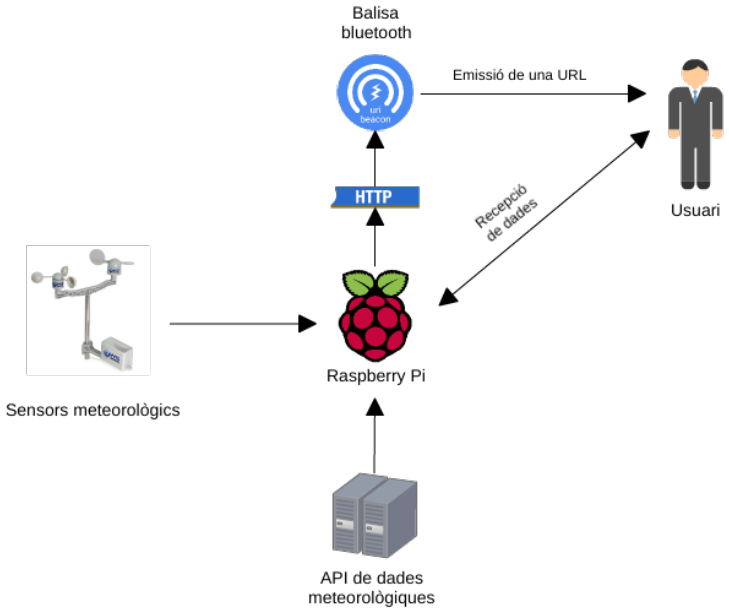

Figura 7: Diagrama d'interacció del cas d'ús de *Physical Web*

En el diagrama que es pot observar en la Figura 8, mostra la interacció del sistema usant la funcionalitat de visualitzar una capa de la informació meteorològica al *Liquid Galaxy*.
Aquesta funcionalitat requerirà un software de gestió, encarregat d'establir connexió amb el servidor de la *RPi* i rebre totes les dades de l'estació meteorològica, mostrant-les al *Liquid Galaxy* en una finestra al costat de la localització exacta de l'estació.
El software de gestió serà desenvolupat utilitzant *Node.js*[11] [8] i contindrà una interfície web que permetrà activar o desactivar la visualització de les dades al *Liquid Galaxy*. Aquesta interfície web es podrà controlar utilitzant els dispositius d'entrada més comuns (teclat i ratolí, panell tàctil, ...) o per veu.

[11] És un entorn en temps d'execució dissenyat per construir aplicacions de xarxa escalables.

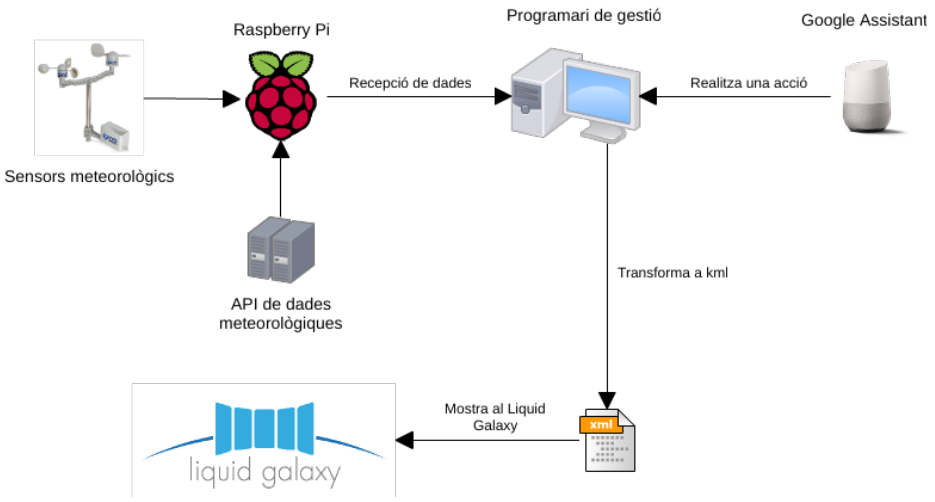

Figura 8: Diagrama d'interacció del cas d'ús de *Liquid Galaxy*

Per últim, la Figura 9 mostra la interacció de l'usuari amb el sistema utilitzant la veu, mitjançant un agent de *Google Assistant*, permetent mantenir una conversa en llenguatge natural.

L'agent s'implementarà utilitzant la plataforma *API.AI*[12][2], la qual mitjançant la seva interfície web, s'utilitzarà per crear tots els diàlegs. Un cop realitzats els diàlegs, l'agent necessitarà les dades que l'usuari li pregunta, per tant, per completar la resposta amb les dades que es necessiten, s'utilitzarà l'opció fulfitment de l'API.AI, que consta un *webhook*[13] que rebrà les peticions pertinents i les retornarà a l'agent, perquè acabi de construir la resposta amb les dades actuals i la retorni a l'usuari.

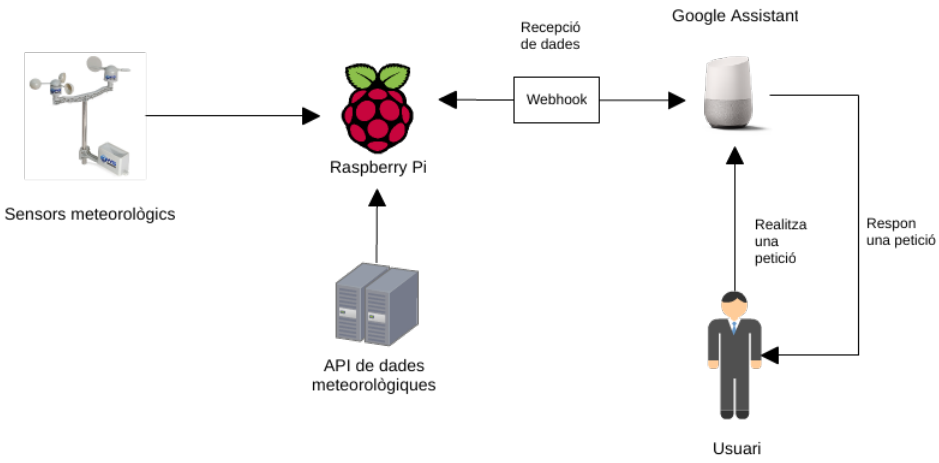

Figura 9: Diagrama d'interacció del cas d'ús de *Google Assistant*

El sistema, a més a més, actualitzarà l'estat de les estacions cada 24 hores en un compte de Twitter[14] creat especialment per aquesta finalitat.

[12]És una plataforma web que permet dissenyar interfícies d'usuari conversacionals en llenguatge natural. És la recomanada per *Google* per realitzar un agent de *Google Assistant*.

[13]És un mètode que defineix *callbacks* (retorns d'una crida) *HTTP* definides per l'usuari.

[14]Twitter és una xarxa social que permet als seus usuaris d'enviar i llegir missatges de text d'una llargada màxima de 140 caràcters anomenats tuits.

1.2 Tasques a realitzar

En aquesta secció s'introduiran les tasques a realitzar en aquest projecte i que són etapes habituals en el desenvolupament del *software*.

Abans de començar a realitzar les tasques, es requerirà un estudi previ que contemplarà tots els aspectes a tenir en compte, i recollirà informació sobre tots els condicionants previs. Un cop finalitzat aquest pas, es podrà procedir a iniciar les següents tasques.

1.2.1 Anàlisis

La primera tasca és analitzar i extreure tots els requisits necessaris del *software*, per conèixer quins són incomplets, ambigus o contradictoris i actuar en conseqüència escollint la millor opció i descriure detalladament i de forma rigorosa el comportament del *software* i la seva interacció entre els diferents sistemes i els usuaris.

Aquesta tasca inclou anàlisis de requeriments, anàlisis del domini i anàlisis de costs.

1.2.2 Disseny

Descriu i determina com el sistema satisfarà els requeriments i com funcionarà de forma general. Consisteix a dissenyar els components definits en les etapes anteriors basant-se en diagrames d'interacció. Es definiran les estructures de dades a utilitzar i la relació entre els components principals. També es realitzarà un prototip de la interfície web, es dissenyaran els diàlegs per *Google Assistant* i es definiran les crides al *webhook*.

1.2.3 Implementació

En aquesta etapa es codifica el sistema d'acord amb el disseny definit anteriorment, implementant totes les seves parts essencials.

Aquestes parts consisteixen en:

- Configuració i recepció de dades dels sensors a la *Raspberry Pi*.
- Recepció de dades de *API*s meteorològiques externes.
- Creació i posada en marxa del servidor.
- Configurar la balisa *bluetooth* aplicant-hi la redirecció de *URL*.
- Implementació de l'agent de *Google Assistant* utilitzant l'aplicació *API.AI*.
- Codificar el *Webhook* que interactuarà amb l'agent.
- Implementar l'aplicació en *Node.js*. Inclou la seva interfície web i la comunicació amb el *Liquid Galaxy*.

1.2.4 Validació

Un cop el sistema ha estat implementat, es realitzen un seguit de proves que validen la seva consistència respecte a l'anàlisi i el disseny i el seu correcte funcionament. En finalitzar aquesta etapa, es considera que el sistema ja està acabat i es pot implantar en producció.

1.3 Memòria

Al llarg de tot aquest projecte, i simultàniament amb els passos anteriors, es redactaran i documentaran tots els detalls del sistema.

Aquesta tasca es dividirà en els següents apartats:

- **Introducció:** Es contextualitzarà el projecte i es presentarà els seus objectius.
- **Estat de l'art:** S'aprofundirà en les totes tecnologies annexes necessàries per a realitzar amb èxit el projecte.
- **Tasques:** Es documentarà la realització de totes les tasques especificades al projecte.
- **Conclusions i línies obertes:** Es resumiran els punts principals als quals s'ha arribat un cop finalitzat el projecte i es comentaran els punts que es deixen oberts i que es poden continuar desenvolupant en un futur.

1.4 Planificació temporal de les tasques a realitzar

En aquesta secció es mostrarà la planificació temporal del projecte, mitjançant el diagrama de *Gantt* que s'observa en la Figura 10.
El projecte s'inicia el 3 d'abril de 2017 i finalitza el 15 de setembre de 2017 tenint una duració total de **5 mesos i 13 dies**.
Les tasques de major duració de la planificació són la implementació, que serà la part on es codificarà tot el sistema, i la memòria, que es realitzarà durant el llarg de tot el projecte paral·lelament amb les corresponents etapes.

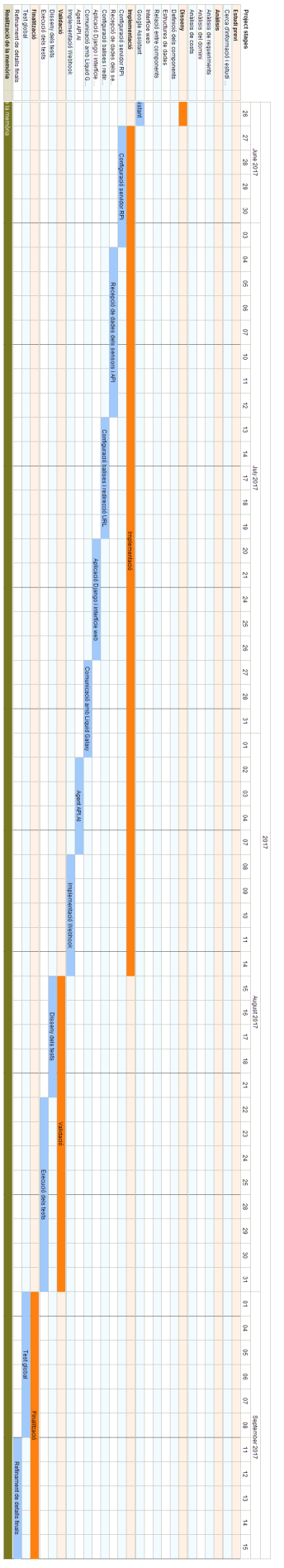

Figura 10: Diagrama de *Gantt* de les tasques a realitzar

2 Estat de l'art

En el transcurs d'aquest capítol estudiarem les tecnologies que es necessitaran per dur a terme aquest projecte amb èxit.

S'introduirà la Raspberry Pi, que realitzarà la tasca d'estació meteorològica gestionant les dades dels sensors i enviant-les al servidor.

S'estudiarà el framework Node.js, que s'utilitzarà per realitzar el servidor i l'aplicació web, i Firebase que contindrà la base de dades per realitzar la persistència.

Finalment s'aprofundirà en les tecnologies que presentaran alternativament les dades a l'usuari. La Physical Web per notificar a l'usuari que està a prop d'una estació, el Liquid Galaxy que mostrarà les dades en un globus terraqüi d'una forma interactiva i immersiva, i Google Assistant que ens permetrà rebre les dades mantenint una conversació mitjançant la veu.

2.1 Raspberry Pi

Raspberry Pi (fig. 11) és una sèrie de petites plaques *Single-Board Computer* desenvolupat per la *Raspberry Pi Fundation*. Té com a finalitat fomentar l'aprenentatge de la programació (especialment en Python) en escoles, però també s'utilitza en projectes de robòtica, IoT[15], etc.

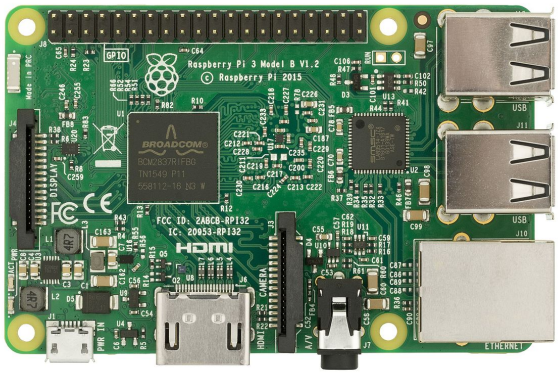

Figura 11: Placa Raspberry Pi 3B

La RPi permet múltiples casos d'ús, com per exemple:

- Funcions de servidor
- Dispositiu d'emmagatzemament en xarxa (NAS)
- Consola arcade (fig. 12)

[15]Internet of Things, és una xarxa d'objectes d'ús quotidià interconnectats mitjançant internet

Figura 12: Consola arcade realitzada amb la Raspberry Pi

- Projectes de robòtica, com realitzar un dron (fig. 13)

Figura 13: Dron amb una Raspberry Pi de controladora

- Projectes de domòtica, com regular la temperatura de l'habitatge
- I molt més...

En aquest projecte s'utilitzarà la tercera generació de RPi, model B, publicada el febrer de 2016, que conté les següents característiques de hardware:

- **CPU 1.2GHz 64-bit quad-core ARMv8:**

  La CPU utilitza l'arquitectura ARM, que és una família de microprocessadors RISC dissenyat per Acorn Computers. Els microprocessadors RISC requereixen menys transistors que els CISC que utilitzen l'arquitectura x86. La major fortalesa de ARM es el seu baix consum. La RPi conté l'última versió de l'arquitectura, ARMv8, que ofereix compatibilitat en 32 i 64 bits.

- **Wireless LAN 802.11n:**

  És necessari que la RPi estigui connectada a la xarxa per enviar les dades al servidor. Tot i que també es pot connectar per cable al port ethernet, utilitzar la xarxa sense fils ens donarà més portabilitat a l'estació.

- **1GB RAM:**

  Aquesta quantitat de memòria principal és suficient per executar el sistema operatiu i el script que envia i rep les dades.

- **40 pins GPIO (General Purpose Input/Output):**

  El port serie GPIO consta de 40 pins on la seva funció és realitzar una interfície física entre la RPi i l'exterior. La figura 14 mostra un esquema de la funció de cada pin. Per connectar els sensors utilitzarem els pins de 3.3v (taronja), GPIO (groc) i Ground (negre).

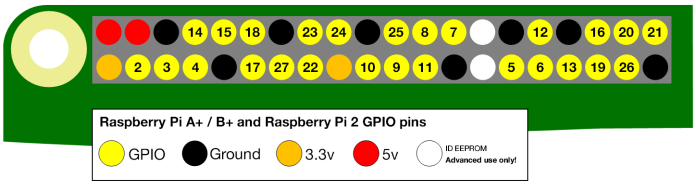

Figura 14: Numeració dels pins GPIO

- **Slot per a targeta Micro SD:**

  Fa la funció de memoria secundària, emmagatzemant el sistema operatiu i les dades.

- **Altres:**

  - 4 ports USB
  - 1 port HDMI
  - 1 port Ethernet
  - 1 port 3.5mm jack de audio

La RPi conté un sistema operatiu d'escriptori, basat en una distribució Linux, això permet tota la potència de programació com un altre PC qualsevol, usant el teu editor de text o IDE preferit, podent programar en múltiples llenguatges de programació:

- Python
- C
- Java
- Javascript
- Pascal
- Go
- ...

En la llista 1 es mostra un exemple bàsic de com es reben les dades del sensor DHT22 en llenguatge python fent ús de la llibreria AdafruitDHT especifica per aquest sensor.

```
import Adafruit_DHT

sensor = Adafruit_DHT.DHT22
pin = 2 # Nombre del port del sensor

humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
print('Temp={0:0.1f}*  Humidity={1:0.1f}%'.format(temperature, humidity))
```

Listing 1: Exemple de recepció de dades del sensor DHT22

2.1.1 Sistema operatiu

En la RPi s'hi pot instal·lar múltiples sistemes operatius:

- Raspbian
- Windows 10 IoT
- Android things
- Ubuntu (no oficial)
- Risc OS
- ...

Durant aquest projecte, com ja hem comentat en la introducció, utilitzarem Raspbian, ja que es la distribució suportada oficialment per la fundació Raspberry Pi.

Raspbian és un port armhf (arm hard float) de Debian Jessie amb l'escriptori PIXEL, que conté unes configuracions de compilació especifiques per produir codi "hard float"optimitzat. Això proporciona un major rendiment en aplicacions que fan ús intensiu de funcions aritmètiques de punt flotant.

2.2 Beacons & Physical Web

Una balisa o beacon Bluetooth Low Energy (Fig. 15), és un dispositiu de baixa potencia i bateria eficient que transmet una URL a través de Bluetooth.

La Physical web et permet veure una llista d'URL que són transmeses per objectes de l'entorn que t'envolta, sempre que estiguis dins del seu rang de transmissió. L'encarregat de transmetre l'URL és la balisa que s'ha anomenat anteriorment.

Figura 15: Balisa Physical Web

Aquesta senzilla idea pot donar pas a múltiples formes d'ús que poden millorar la nostra vida quotidiana, tal com mostra la figura 16. Podem interactuar amb un parquímetre, ajustant el temps i pagant, podem introduir les dades a la nostra mascota perquè sigui més senzill localitzar-la en cas de pèrdua, entre d'altres. En definitiva, és útil per a qualsevol objecte que necessiti transmetre informació quan hi estàs a prop, com la nostra estació meteorològica.

Figura 16: Casos d'us de Physical Web

En primer lloc es posarà la balisa en mode configuració i mitjançant Bluetooth i la seva web de configuració[16], que utilitza el protocol "web bluetooth API", s'introduirà la URL que es vol transmetre de la manera que es mostra en la figura 17.

[16]https://beaufortfrancois.github.io/sandbox/web-bluetooth/eddystone-url-config/index.html

Figura 17: Web de configuració de la balisa

Serveis com Nearby notifications o Google Chrome buscarà i mostrarà aquestes URL mitjançant una notificació tal com es pot veure a la figura 18. El servei el gestiona Google Play Services i és automàtic, sense necessitat de fer res per part del desenvolupador ni de l'usuari (excepte tenir el bluetooth encès). Quan se selecciona la notificació, s'obrirà el navegador i visitarà la URL que emet la balisa.

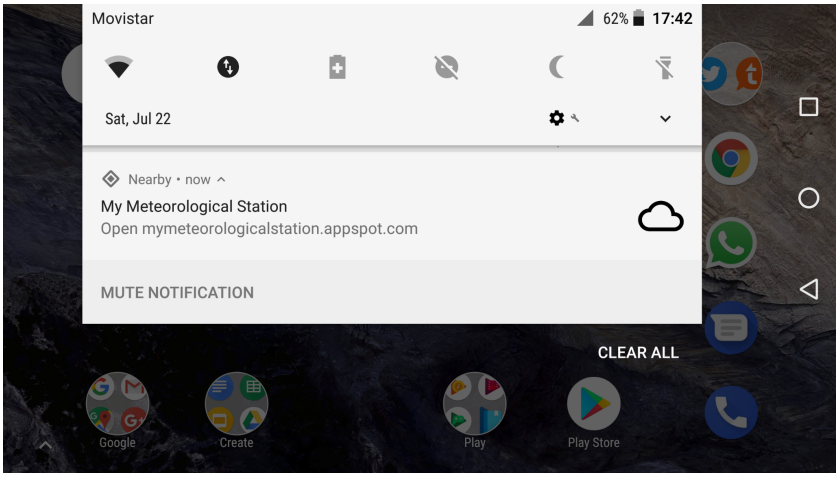

Figura 18: Exemple de notificació physical web en un smartphone Android

2.3 Liquid Galaxy

És un projecte de codi lliure desenvolupat per Google que permet visualitzar *Google Earth*[17] en mode multi pantalla creant una experiència panoràmica i envoltant.

Una instal·lació de *Liquid Galaxy* és un clúster de computadors on cada node gestiona una pantalla. Un dels nodes, generalment el que gestiona la pantalla central, és el *master* i s'encarrega de rebre tota la informació i replicar-la a la resta de nodes utilitzant *SSH*, permetent que es comportin de manera sincronitzada.

[17]És una aplicació que conté mapes geogràfics presentats en un globus terraqüi tridimensional, que conté tot tipus d'informació geogràfica, on l'usuari hi pot interactuar. S'hi pot representar dades utilitzant el llenguatge KML.

Figura 19: Instal·lació *Liquid Galaxy* real de set monitors

2.3.1 Google Earth

Tal com s'ha comentat anteriorment, el Liquid Galaxy executa el programari Google Earth (fig. 20). Aquest programa permet explorar qualsevol ubicació mitjanant un globus terraqüi tridimensional.

Damunt del globus pots dibuixar polígons, pop-ups amb text, o realitzar tours, que permet programar la visita automàtica de diferents ubicacions. La funcionalitat que s'implementarà en aquest projecte serà un pop-up amb les dades de l'estació damunt de la localització on es troba i un tour que permetrà recórrer totes les estacions que hi ha actives.

Totes aquestes opcions es permet realitzar-les important les dades des d'un fitxer en format KML, que s'introdueix en el següent apartat.

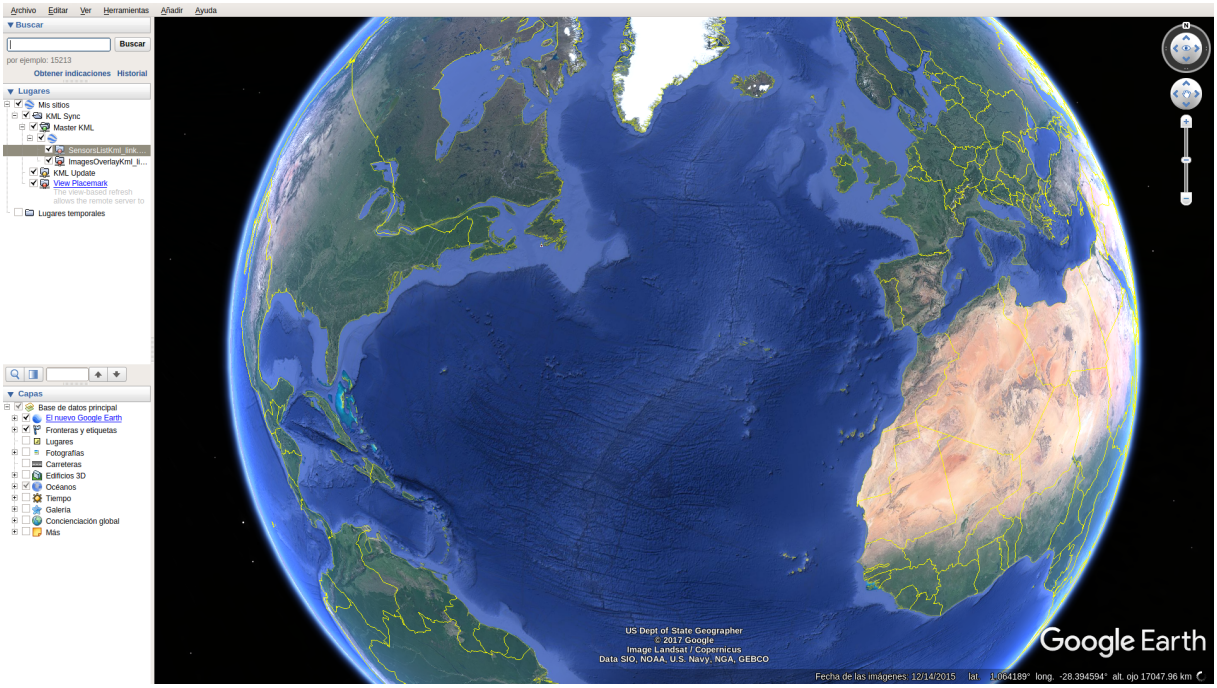

Figura 20: Pantalla principal del software Google Earth.

2.3.2 KML (Keyhole Markup Language)

Per tal d'enviar dades i representar-les al Liquid Galaxy en un format que Google Earth entengui utilitzarem el llenguatge KML.

El llenguatge *KML*[5] és un estàndard, basat en el llenguatge de marques basat en *XML*[18], que permet representar dades geogràfiques que especifiquen una característica, com per exemple una posició determinada en un lloc, o un polígon.

Un exemple del contingut d'un fitxer *KML*, especificant un títol, una descripció i unes coordenades podria ser:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <kml xmlns="http://www.opengis.net/kml/2.2">
3    <Placemark>
4     <name>Name</name>
5     <description>Description</description>
6     <Point>
7       <coordinates>0.843868,40.360758,0</coordinates>
8     </Point>
9    </Placemark>
10  </kml>
```

Listing 2: Exemple XML

Un fitxer *KMZ* és un fitxer comprimit que conté una sèrie de recursos a més a més del fitxer *KML*, com podria ser imatges, models 3D, etc.

[18] És un metallenguatge extensible, d'etiquetes, desenvolupat pel World Wide Web Consortium (W3C).

2.3.3 Comunicació amb el Liquid Galaxy

La comunicació amb el LG és d'un sol sentit, només rep dades de fora, però ell no retorna res a l'exterior.

El LG està permanentment escoltant el fitxer kmls.txt. En aquest fitxer s'hi pot introduir el contingut d'un fitxer kml o la URL que apunta a un fitxer KML i en el moment que detecta el canvi en aquest fitxer, el LG el descarrega i mostra el contingut a les pantalles. Es pot editar el fitxer kmls.txt des de l'exterior usant el servidor mitjançant les utilitats SSH[19] i SCP[20].

2.4 Google Assistant (Actions on Google, Api.AI)

És un assistent personal intel·ligent creat per Google. Va debutar el Maig del 2016, fruit de la dura competència amb Siri (Apple) i Cortana (Microsoft), ja que l'assistent anterior, el Google Now, no estava a l'altura d'aquests en el tema del llenguatge natural.

Actualment només està disponible en els idiomes Angles i Alemany. Pot ser executat en smartphones Android amb uns certs requisits o en el dispositiu Google Home, creat exclusivament per aquesta funció.

Google Assistant permet mantenir una conversa en llenguatge natural, podent demanar una mateixa informació de diferents maneres, en comptes d'una comanda determinada, i guardant informació d'una consulta a l'altra permetent a l'assistant conèixer en quin context es troba, per exemple, si preguntem pel temps de Lleida, després només dient "i demà?" sobreentendrà que li estàs preguntant pel temps i de Lleida sense haver de repetir-los.

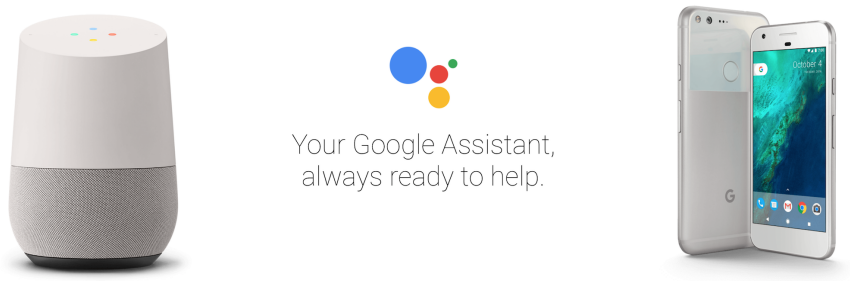

Figura 21: Dispositius que executen el software Google Assistant (Google Home i Google Pixel).

Per tal de desenvolupar l'agent per aquest sistema utilitzarem l'eina API.AI amb l'opció de fulfitment.

API.AI

És una plataforma web que permet crear aplicacions conversacionals de veu o text en llenguatge natural. Permet integració en diversos sistemes com Google Assistant, Slack, Facebook, Skype, Twitter,

[19]SSH (Secure SHell) és un protocol que serveix per accedir a màquines remotes a través de la xarxa.

[20]SCP (Secure Copy Protocol o Simple Communication Protocol) és un mitjà de transferència segura d'arxius informàtics entre un host local i un altre remot o entre dos hosts remots, usant el protocol Secure Shell (SSH).

etc.

Quan un usuari realitza una pregunta, es llença un intent, que definirà tant la pregunta com la resposta. En la figura 22 es defineix un intent d'exemple que quan l'usuari digui "Hello" l'agent contestarà "Hi! How are you?".

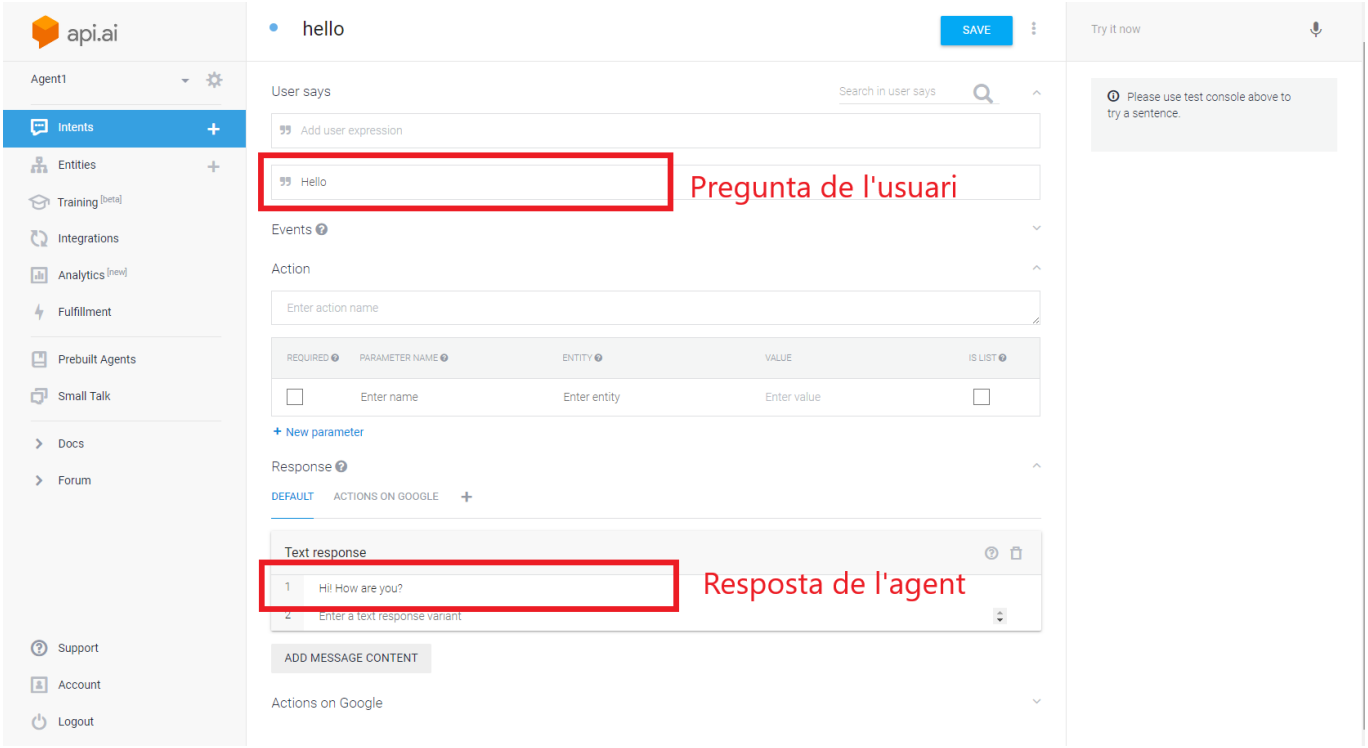

Figura 22: Intent bàsic d'un agent de Api.ai.

La plataforma et permet definir diferents preguntes per llençar el mateix intent i diferents respostes que contestarà l'agent aleatòriament.

En aquest moment podem posar l'agent en mode test per veure com respon. Navegant fins a la secció integrations i dins de Actions on Google clicar el botó Test tal com mostra la figura 23.

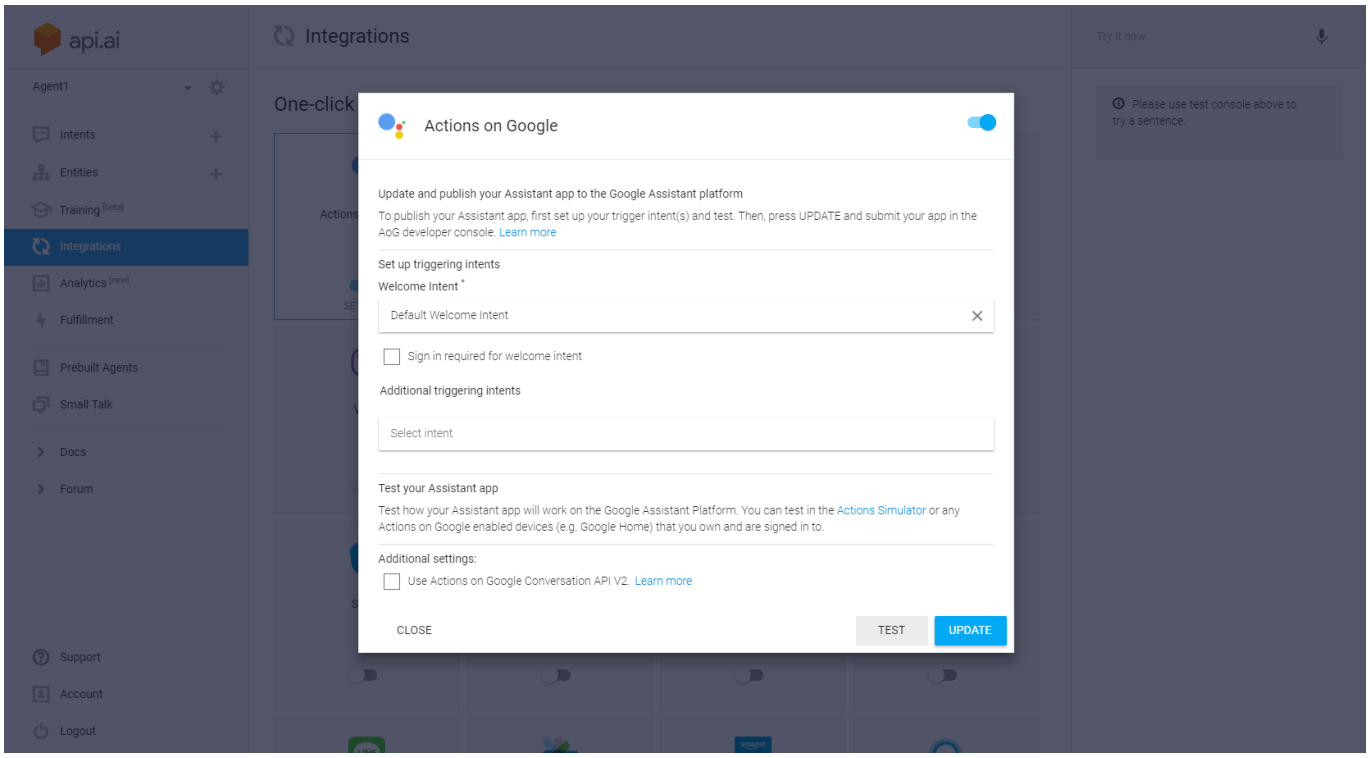

Figura 23: Enviar l'agent com a aplicació test.

Ara ja podem provar l'intent que acabem de crear en un dispositiu real que executi el software Google Assistant. En la figura 24 es mostra una captura de la prova en un smarthpone.

Per llençar l'aplicació és necessari dir la frase "talk to my test app", i un cop torna la salutació inicial per defecte es llança l'intent que hem creat dient el "hello" rebent la resposta pertinent.

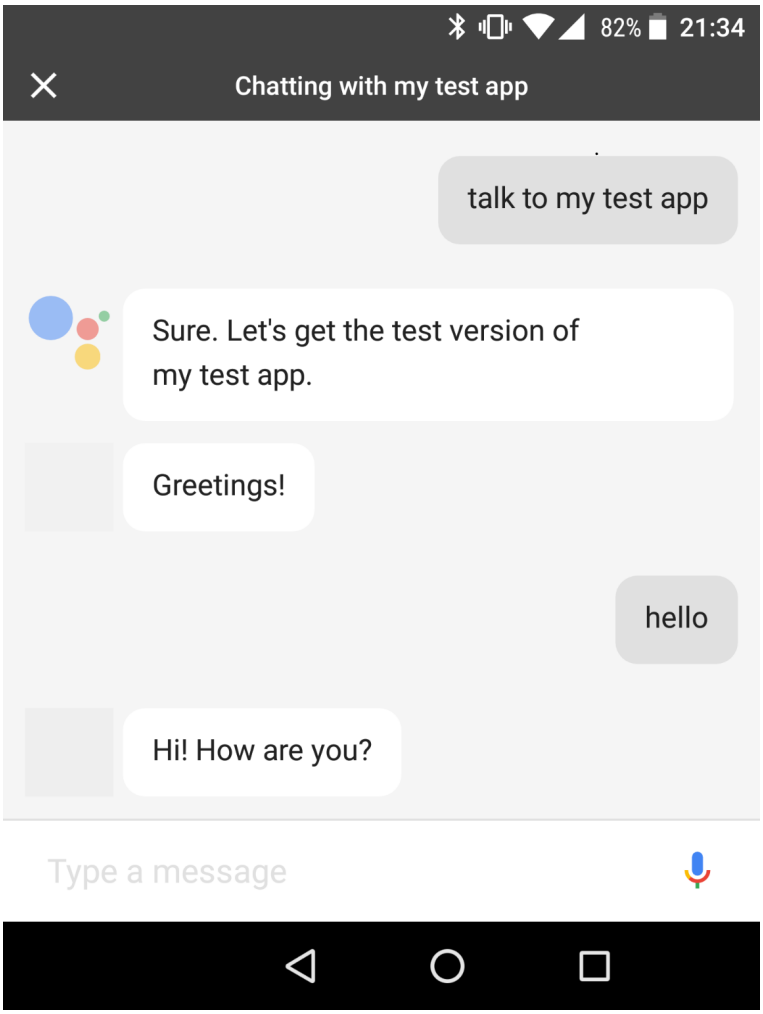

Figura 24: Prova de l'agent en un dispositiu real.

Fulfillment

La plataforma API.AI permet rebre informació externa del nostre *webhook*[21] fent peticions POST i rebent la informació actualitzada per tal d'inserir-la en la resposta que retornarà a l'usuari.

Hi ha situacions en què és necessari completar la resposta amb dades variables que hem d'anar a buscar en un servei extern, i per tant, no es pot posar una resposta predefinida a l'API.AI tal com mostra l'exemple anterior, com és el cas de l'estació meteorològica, on els valors dels sensors canvien en cada lectura. Aquesta acció d'emplenar la resposta s'anomena fulfillment.

Per realitzar el fulfillment necessitarem el servei extern esmentat anteriorment, que retorna les dades a l'API.AI, anomenat webhook. És un mètode que defineix *callbacks* (retorns d'una crida) *HTTP* definides per l'usuari.

A continuació es mostrarà com connectarem el webhook amb l'agent API.AI

En primer lloc, encara a l'API.AI introduirem la direcció del nostre webhook a l'apartat fulfillment (figura 25).

[21] És un mètode que defineix *callbacks* (retorns d'una crida) *HTTP* definides per l'usuari.

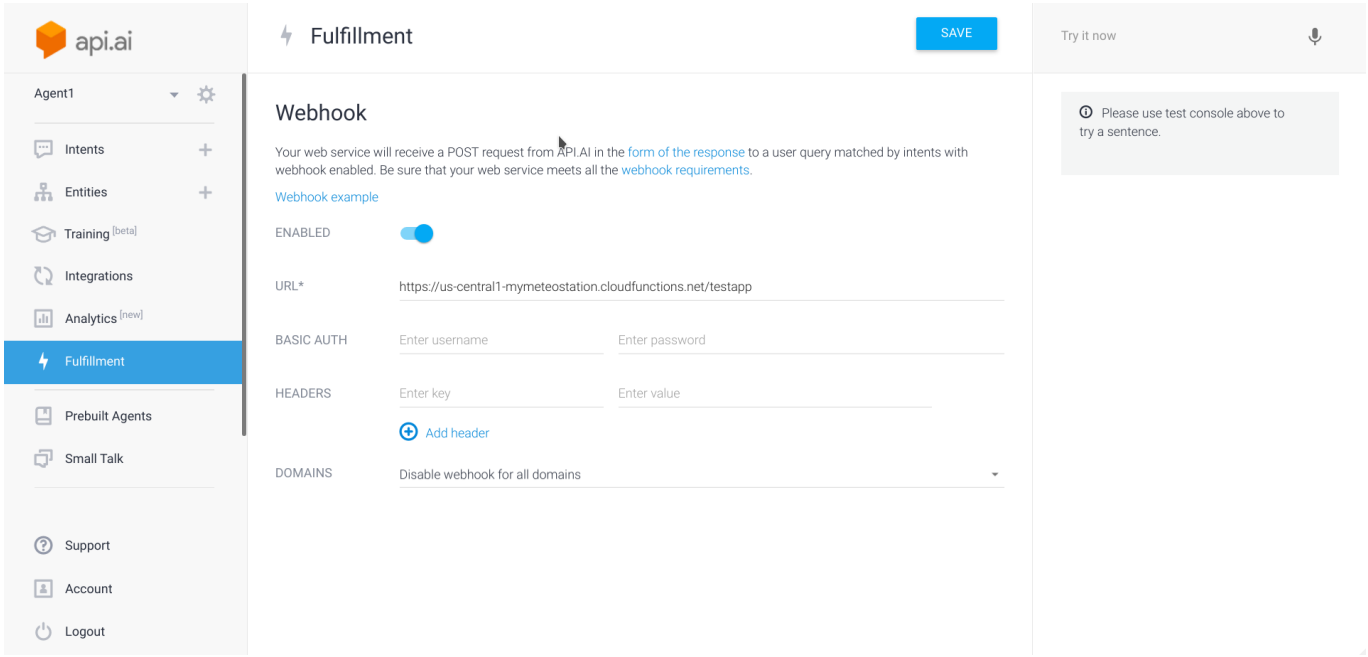

Figura 25: Secció per introduir la URL del webhook al projecte.

Ara crearem un intent d'exemple bàsic en què li dius un número i el nostre webhook repetirà el número que l'usuari ha dit. En la figura 26 es pot veure que l'intent es llença quan l'usuari diu "my favourite number is X". On x serà qualsevol numero, que es defineix com a variable (nombre enter) a enviar al webhook amb el nom de "number". Al quadre action hi definirem el nom de la funció que es cridarà al webhook i en l'apartat fulfillment se seleccionarà el quadre que habilita el webhook per aquest intent.

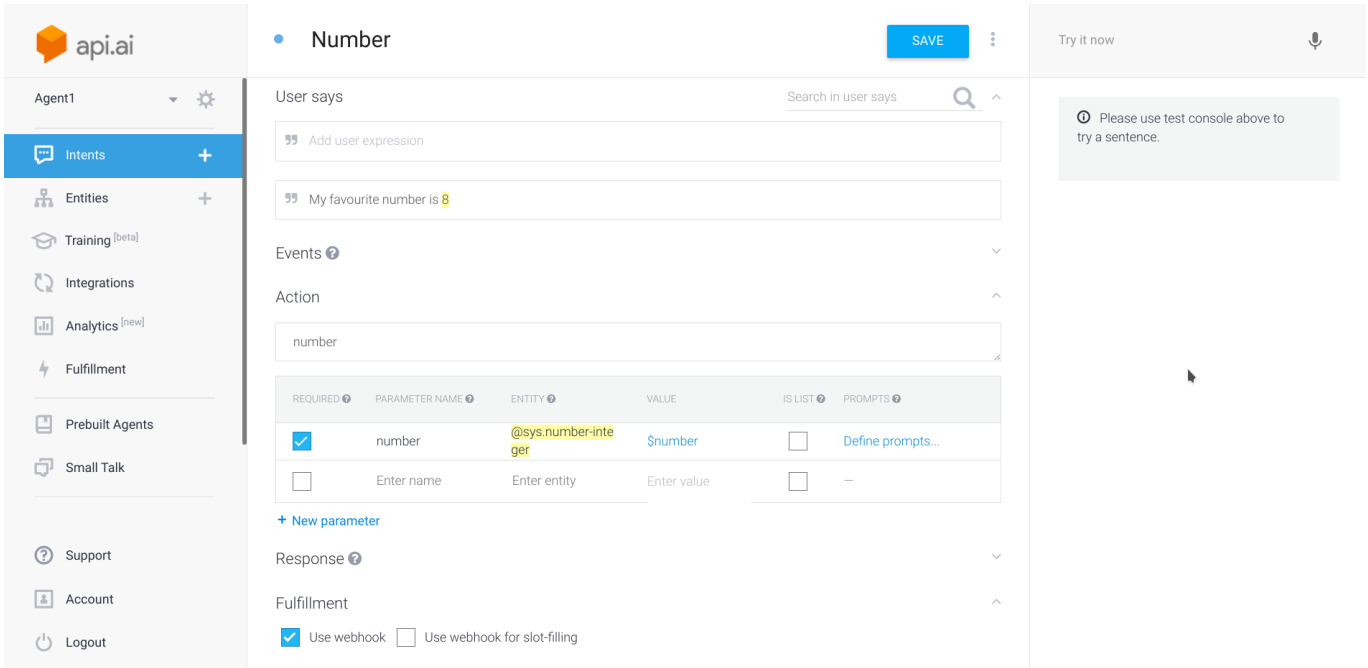

Figura 26: Intent de prova per testejar el webhook.

En la llista 3 es mostra el codi que executarà el webhook.

En el map, el primer paràmetre és el nom que hem introduït a action dins l'intent, que es relacionarà amb la funció que es cridarà, en aquest cas "numberIntent".

Aquesta funció primer rebrà el número que ha dit l'usuari, usant la funció app.getArgument() especificant-li el nom de la variable de l'intent de l'API.AI.

El webhook respondrà utilitzant la funció app.tell(), on s'hi definirà la resposta. Si es vol tornar a fer una pregunta a l'usuari, en lloc d'acabar la conversa s'utilitzarà app.ask().

```
'use strict';

process.env.DEBUG = 'actions-on-google:*';
const App = require('actions-on-google').ApiAiApp;
const functions = require('firebase-functions');


exports.testapp = functions.https.onRequest((request, response) => {
    const app = new App({request, response});

    function numberIntent (app) {
      // Nom del parametre
        let number = app.getArgument('number');

        app.tell('Your favourite number is ' + number);
    }

    let actionMap = new Map();
    // Nom de Action
    actionMap.set('number', numberIntent);

    app.handleRequest(actionMap);
});
```

Listing 3: Exemple de webhook

Ara podem testejar que l'intent i el webhook funcionin correctament de la mateixa forma que he provat anteriorment, tal com s'observa a la figura 27.

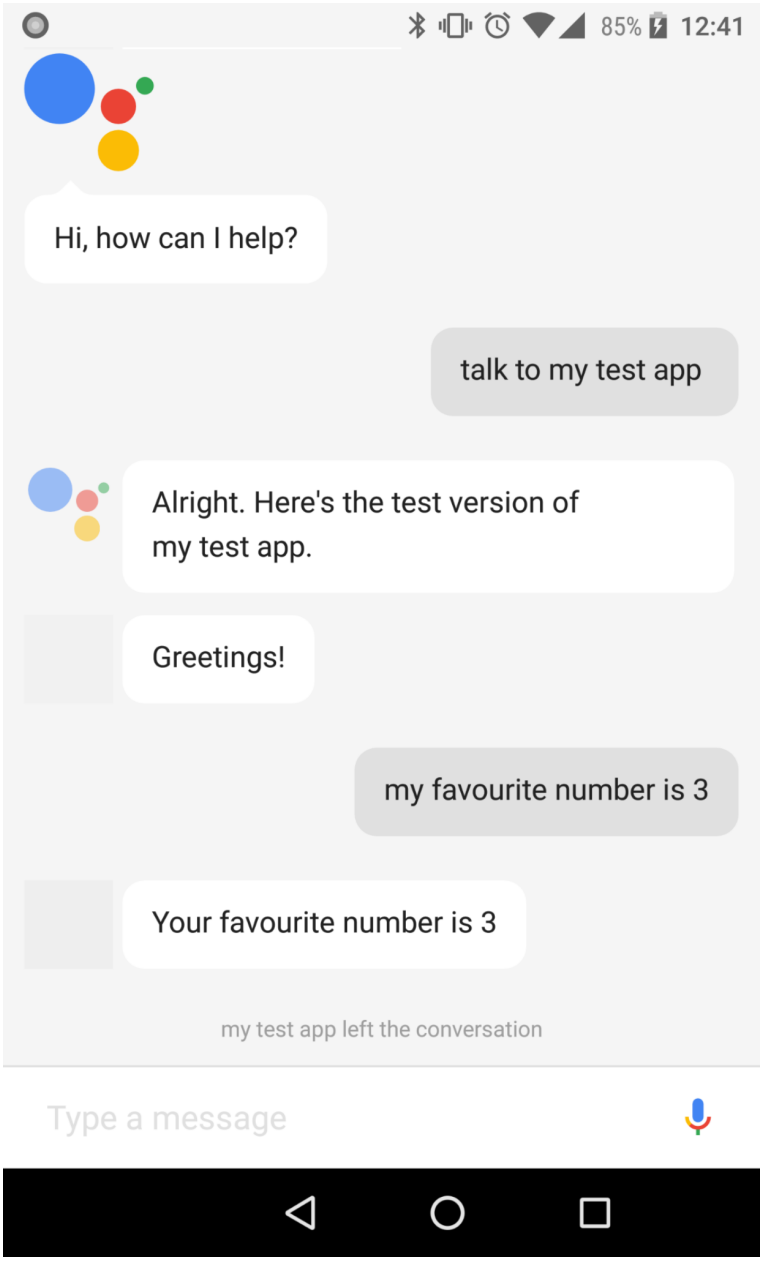

Figura 27: Prova de l'agent en un dispositiu real.

2.5 Node.js

Node.js es un entorn en temps d'execució de Javascript asíncron dissenyat per construir aplicacions de xarxa escalables, basat en el motor Chrome V8.

Pot servir múltiples connexions simultànies en un mateix fil d'execució, utilitzant l'entrada/sortida de manera no bloquejant, evitant així el cost que comporta utilitzar múltiples fils.

L'exemple més senzill d'una aplicació Node.js, que deixa escoltant el servidor en un port determinat, és el següent:

```
1 const http = require('http');
2
3 const hostname = '127.0.0.1';
```

```
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log('Server running at http://${hostname}:${port}/');
14 });
```

Listing 4: Exemple aplicació Node.js

2.5.1 Mòduls i NPM

Node.js disposa d'una col·lecció molt amplia de mòduls de tercers, que estenen la seva funcionalitat dotant d'un major valor a l'entorn.

A continuació es llistarà els diferents mòduls que s'utilitzaran en el projecte:

- Express: Proporciona un robust conjunt de funcions per a una aplicació web.

  Permet definir una llista de mètodes de sol·licitud HTTP essencials pel servidor:

  - GET: Sol·licita una representació del recurs especificat.
  - POST: Envia dades per a ser processades a un recurs específic. Les dades s'inclouen en el cos de la petició.
  - PUT: Modifica les dades del recurs especificat.
  - DELETE: Esborra el recurs especificat.
  - ...

  Els mètodes de resposta que permet express són els següents:

  - res.download(): Demana que es baixi un fitxer.
  - res.end(): Finalitza el procés de resposta.
  - res.json(): Envia una resposta JSON.
  - res.redirect(): Reencamina una sol·licitud.
  - res.render(): Renderitza una vista.
  - res.send(): Envia una resposta de diversos tipus.
  - res.sendFile(): Envia un fitxer en format d'octets.
  - res,sendStatus(): Estableix un codi d'estat a la resposta.

  En la llista 5 es mostra un exemple d'encaminament bàsic utilitzant el mètode GET, que respondrà un "hello world" en demanar la pàgina d'inici.

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('hello world')
})
```

Listing 5: Exemple d'encaminament bàsic en express utilitzant el mètode GET.

- Body-parser: Transforma una sol·licitud entrant en un middleware abans dels controladors, i el defineix sota la propietat req.body.

  A continuació es mostra un exemple de l'ús del mòdul accedint a la propietat del cos "username".

```
var express = require('express')
var bodyParser = require('body-parser')
var app = express()

var jsonParser = bodyParser.json()
var urlencodedParser = bodyParser.urlencoded({ extended: false })

app.post('/login', urlencodedParser, function (req, res) {
  if (!req.body) return res.sendStatus(400)
  res.send('welcome, ' + req.body.username)
})
```

Listing 6: Exemple d'ús de Body-parser.

- Firebase: Permet connectar amb un projecte Firebase per tal de fer ús de les seves característiques (base de dades, autenticació, ...). La seva explicació s'estén en el següent apartat.

Els mòduls Node es gestionen utilitzant l'eina NPM, que proporciona les funcions d'instal·lar, compartir i distribuir el codi, administrant totes les dependències del projecte.

Per tal d'instal·lar un mòdul i poder-lo utilitzar en el projecte realitzarem la següent comanda:

```
$ npm install <nom del modul> --save
```

2.5.2 Package.json

Una aplicació Node.js conte un fitxer anomenat "package.json" que conté:

- Nom, versió i descripció del projecte.
- La llista de dependències.

  S'introdueixen automàticament amb la comanda d'instal·lació anterior si s'introdueix la paraula clau "–save".

- La comanda d'inici per l'execució de l'aplicació.

  Per executar una aplicació node s'introdueix en un terminal:

```
$ node <nom del fitxer d'inici>.js
```

Definint aquesta comanda d'inici en el fitxer package.json simplificarem la sintaxi i podrem arrancar la nostra aplicació utilitzant la comanda:

```
$ npm start
```

En la llista 7 es mostra un exemple del contingut del fitxer package.json.

```
{
  "name": "my-meteorological-station",
  "version": "0.0.1",
  "description": "view data from RPi sensors",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "body-parser": "^1.17.2",
    "ejs": "^2.5.6",
    "express": "^4.15.2",
    "firebase": "^4.1.3",
    "pug": "^2.0.0-rc.2"
  }
}
```

Listing 7: Exemple del contingut del fitxer package.json

2.6 Firebase

Firebase és una plataforma de desenvolupament per a mòbils i aplicacions web. Ofereix un conjunt d'eines que faciliten el desenvolupament de la teva aplicació.

Les seves funcions principals són l'autenticació, la base de dades en temps real, emmagatzemament, funcions cloud, emmagatzemament de fitxers, etc.

A continuació s'ampliaran les funcionalitats que s'utilitzaran durant el projecte, l'autenticació i la base de dades.

2.6.1 Authentication

Si la teva aplicació necessita conèixer la identitat de l'usuari, Firebase ofereix un sistema d'autenticació perquè les dades de l'usuari es guardin de manera segura, i aquestes dades se sincronitzin en tots els seus dispositius.

Es permet l'autenticació dels diferents proveïdors, tal com es mostra la figura 28, que es llisten a continuació:

- Email/Password
- Phone
- Google

- Facebook
- Twitter
- Github
- Anonymous

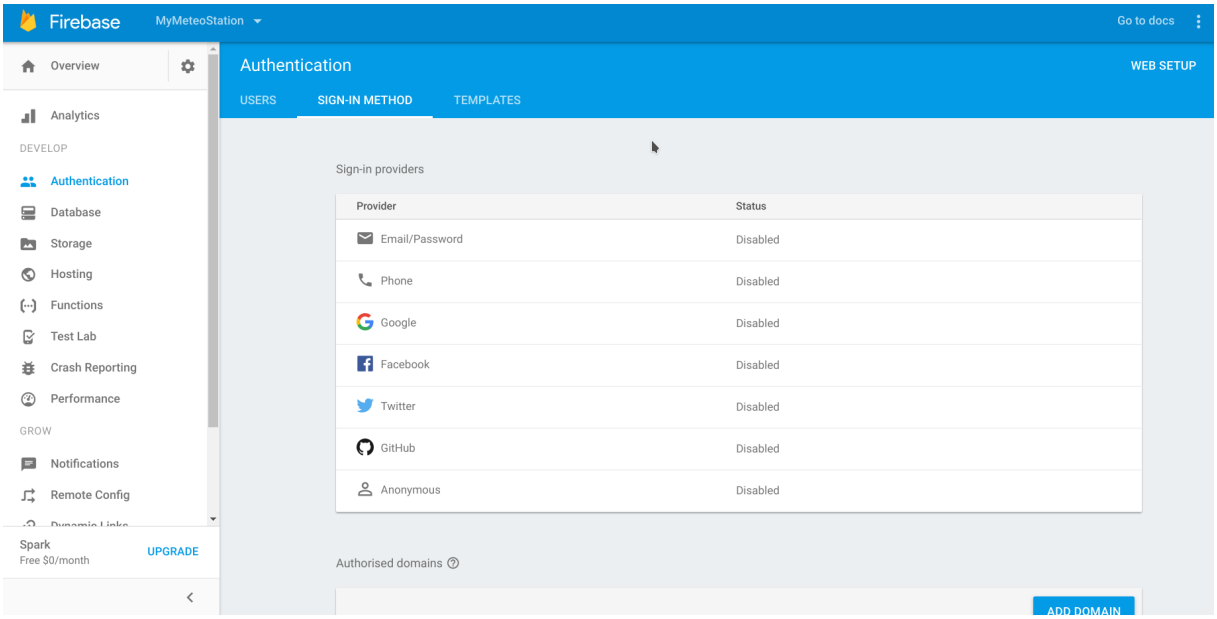

Figura 28: Pantalla de selecció de proveïdor de Firebase

A continuació es mostrarà com integrar l'autentificació de Firebase al nostre projecte.

Crear un usuari usant mail i contrasenya

```
firebase.auth().createUserWithEmailAndPassword(email, password).catch(function(error)
    {
  var errorCode = error.code;
  var errorMessage = error.message;
});
```

Listing 8: Exemple creació d'usuari en Firebase

Iniciar sessió usant mail i contrasenya

```
firebase.auth().signInWithEmailAndPassword(email, password).catch(function(error) {
  var errorCode = error.code;
  var errorMessage = error.message;
});
```

Listing 9: Exemple d'inici de sessió d'usuari en Firebase

Observar si l'usuari ha iniciat sessió

```
firebase.auth().onAuthStateChanged(function(user) {
  if (user) {
    // User is signed in.
  } else {
```

```
    // No user is signed in.
  }
});
```

Listing 10: Exemple d'observador del canvi d'estat de l'usuari en Firebase

Rebre la informació d'un perfil

```
var user = firebase.auth().currentUser;

if (user != null) {
  var name = user.displayName;
  var email = user.email;
  var photoUrl = user.photoURL;
  var emailVerified = user.emailVerified;
  var uid = user.uid;
}
```

Listing 11: Exemple de recepció de dades d'un usuari en Firebase

2.6.2 Realtime database

La informació de la base de dades està estructurada en arbres JSON (figura 29). Al contrari que una base de dades SQL, no hi ha ni taules ni relacions. El punt fort d'aquesta funció és que permet l'actualització de les dades en temps real al client quan aquestes canvien.

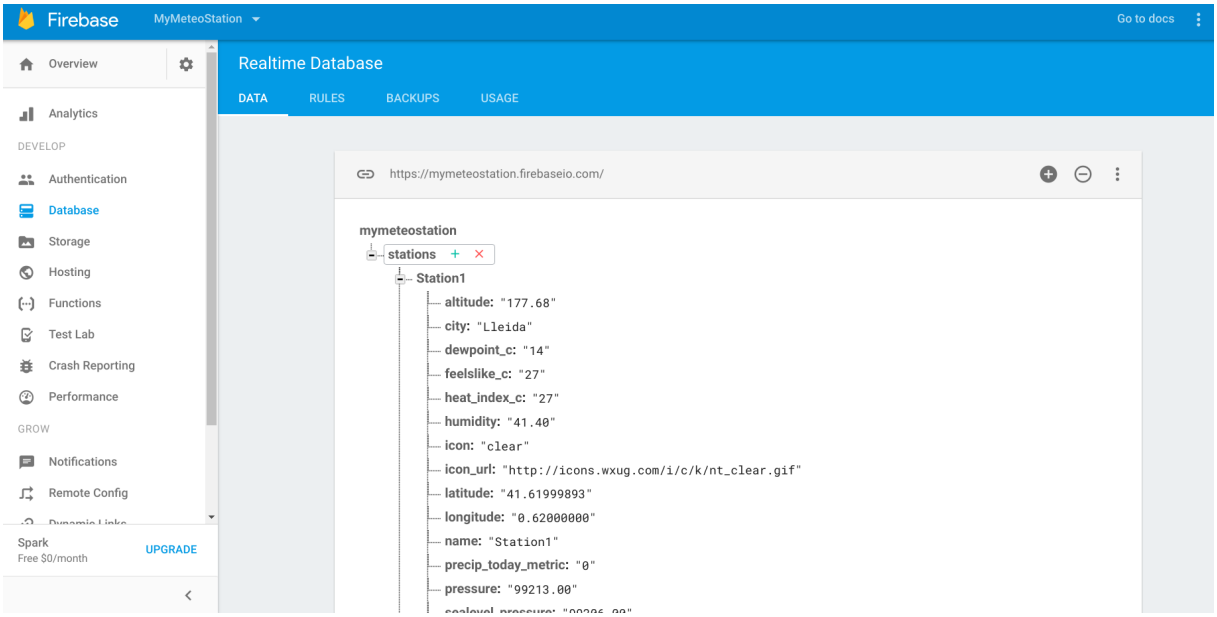

Figura 29: Captura de la base de dades de Firebase

En la llista següent es mostra un exemple de com les dades estan estructurades a la base de dades.

```
{
  "usuari": {
    "usuari1": {
      "name": "nom usuari 1",
      "contactes": { "usuari2": true },
```

```
    },
    "usuari2": { ... },
    "usuari3": { ... }
  }
}
```

Listing 12: Exemple d'estructura de la realtime database de Firebase

La base de dades conté una sèrie de regles de seguretat que permeten llegir o escriure les dades de manera pública, només a usuaris autenticats o d'un usuari en concret.

Tothom pot llegir i escriure a la base de dades

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Listing 13: Exemple de seguritat desactivada en la base de dades de Firebase

Només usuaris autenticats poden llegir i escriure a la base de dades.

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

Listing 14: Exemple de restricció d'usuaris no autenticats en la base de dades de Firebase

Només l'usuari pot accedir i modificar les seves dades.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

Listing 15: Exemple de restricció d'usuaris no propietaris de les dades en la base de dades de Firebase

Les funcions per tal d'utilitzar i administrar la base de dades són les següents:

Escriure dades a la base de dades

```
function writeUserData(userId, name, email, imageUrl) {
  firebase.database().ref('users/' + userId).set({
    username: name,
    email: email,
    profile_picture : imageUrl
  });
```

```
}
```

Listing 16: Exemple de inserció de dades en la base de dades de Firebase

El resultat en la base de dades seria el següent:

```
{
  "users": {
    "userId": {
      "username": "...",
      "email": "...",
      "profile_picture": "...",
    },
  }
}
```

Listing 17: Resultat de la inserció de dades de la realtime database de Firebase

Lectura de dades de la base de dades

```
var reference = firebase.database().ref('users/' + userId);
reference.on('value', function(snapshot) {
  console.log( snapshot.val() );
});
```

Listing 18: Exemple de lectura de dades en la base de dades de Firebase

2.7 Twitter

Twitter és una xarxa social que permet als seus usuaris d'enviar i llegir missatges de text d'una llargada màxima de 140 caràcters anomenats tuits.

Aquesta plataforma disposa d'una API que programàticament permet llegir o escriure tuits, llegir perfils d'usuari, dades de seguidors, etc.

En aquest projecte es voldrà escriure tuits de les actualitzacions de cada estació cada 24 hores. Per aconseguir això és necessari crear una *Twitter app* a https://apps.twitter.com (figura 30), que ens permetrà comunicar-nos des del nostre servidor al compte de Twitter.

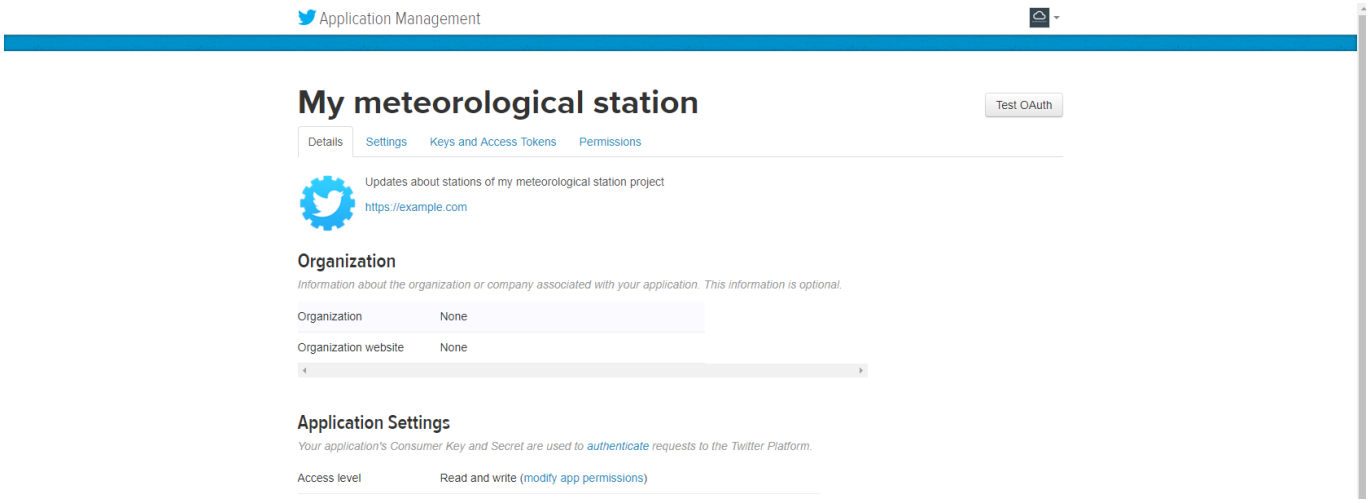

Figura 30: Pàgina de detalls d'una aplicació de Twitter

3 Anàlisis

A continuació s'analitzaran detalladament les parts i interaccions més importants, per tal d'identificar totes les necessitats i definir els límits del sistema.

3.1 Anàlisis de requeriments

En aquest apartat es detallaran els requeriments, que consta de les funcionalitats que ha d'oferir el sistema, les restriccions que sigui necessari respectar i reflectir les necessitats del client.

3.1.1 Requeriments funcionals

Els requeriments funcionals descriuen les funcionalitats i serveis que el sistema ha de fer, incloent-hi la resposta dels esdeveniments que rep de l'exterior. Seran els següents:

- **El sistema haurà de** rebre i emmagatzemar les dades provinents d'una o múltiples estacions meteorològiques.
- **El sistema haurà de** servir les dades de les estacions meteorològiques quan l'usuari les vulgui consultar.
- **El sistema haurà de** retornar les dades a Google Assistant perquè les pugui dictar a l'usuari.
- **El sistema haurà de** generar un KML amb les dades actuals i enviar-li al Liquid Galaxy perquè pugui mostrar-les.
- **L'estació meteorològica haurà d'**enviar periòdicament al sistema les dades actualitzades extretes dels sensors i de la API.
- **La balisa haurà d'**emetre la URL que serveix el sistema dins del radi de proximitat que permet la tecnologia bluetooth.

3.1.2 Requeriments no funcionals

Els requeriments no funcionals agrupen els atributs o propietats del software que ha de mostrar el sistema en realitzar la seva funció.

Producte

- **Portabilitat:** L'aplicació haurà de ser compatible amb els diferents navegadors existents, independentment del sistema operatiu.
- **Suport**
  - **Adaptabilitat:** Es requereix que la interfície s'adapti correctament a diferents mides de pantalla en diferents tipus de dispositius (escriptori, smartphone, tablet, ...).

- **Instal·lació:** No ha de requerir la instal·lació de cap software suplementari per fer funcionar l'aplicació des del navegador.

Extern

- **Privacitat:** Tant les dades com la localització d'una estació meteorològica serà publica. No es guardarà cap informació dels usuaris que consultin les dades i per tant serà anònima.
- **Legislatiu:** L'aplicació complirà la normativa i llei vigent de cada país en tots els seus aspectes.

3.1.3 Diagrama de casos d'ús

Els casos d'ús són fragments de funcionalitats que determinen quines són les funcions del sistema des del punt de vista de l'usuari o les entitats externes.

En el diagrama de la figura 31 es detallen les funcionalitats més importants i les relacions entre les seves entitats.

En primer lloc tenim la RPi o estació que envia les seves dades al servidor perquè les emmagatzemi. És l'únic cas d'ús on no intervé l'usuari.

L'usuari pot consultar les dades que demanarà al servidor i aquest les hi retornarà presentades en una pàgina web.

També pot consultar per veu a Google Assistant, on aquest demanarà les dades al servidor i les dictarà a l'usuari.

Per mostrar la informació al Liquid Galaxy, l'usuari donarà l'ordre al servidor, que generarà les dades en un fitxer kml i les hi enviarà.

En l'últim cas d'ús, es representa quan un smartphone detecta el senyal d'una balisa bluetooth, i mostra automàticament una notificació.

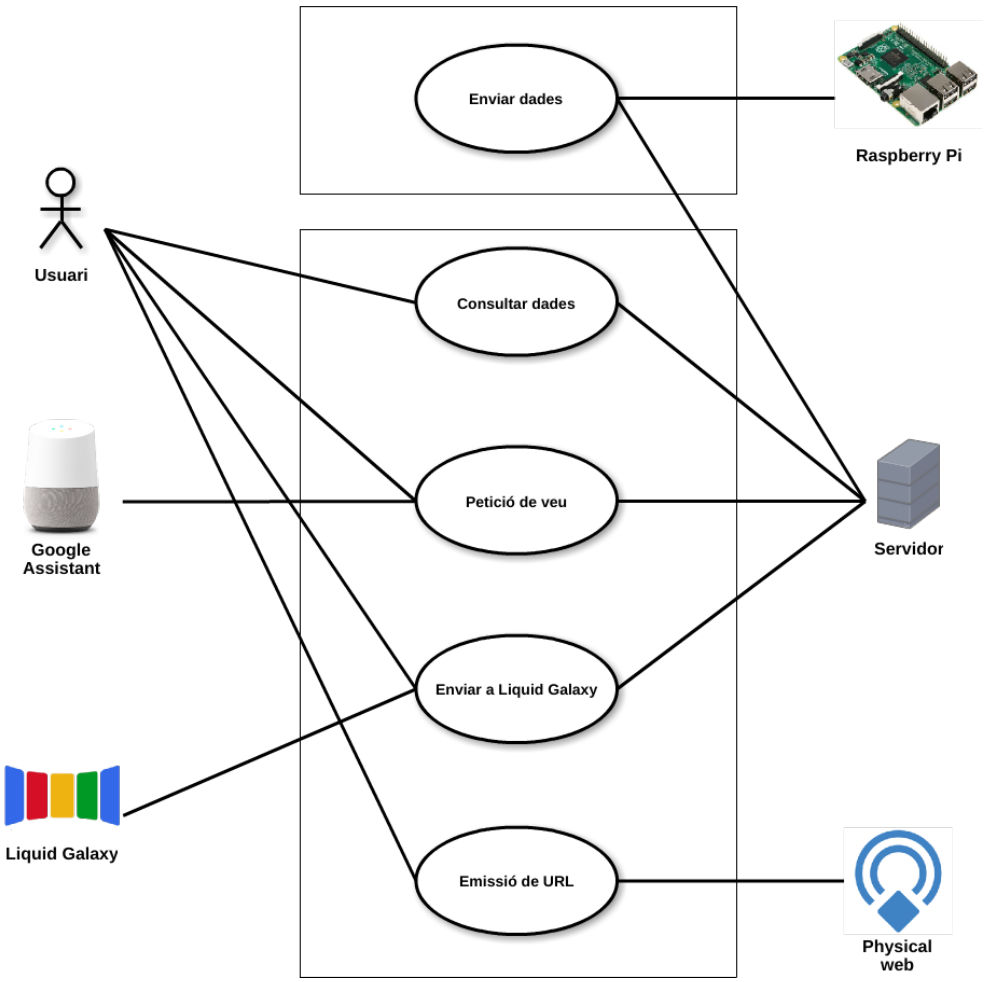

Figura 31: Diagrama de casos d'ús

3.1.4 Especificació dels casos d'ús

S'especificaran els casos d'ús separats per dos tipus, l'especificació d'alt nivell, que descriu breument l'escenari d'èxit del cas d'ús, i l'especificació expandida, que detalla els possibles escenaris del cas d'ús, amb totes les possibilitats d'èxit i fracàs.

Consultar dades

- **Especificació d'alt nivell**

Cas d'ús:	Consultar dades
Actors:	Principal: Usuari (iniciador) Recolzament: Servidor
Propòsit:	Consultar les dades de les estacions per part d'un usuari.
Descripció:	L'usuari accedeix a una URL que mostrarà la web que serveix el servidor i podrà consultar totes les dades interactuant amb l'aplicació web.

- **Especificació expandida**

Referències creuades:

Casos d'ús: Consultar dades

Curs típic d'esdeveniments:

Accions dels actors	**Respostes del sistema**
1. L'usuari obra un navegador web i ingressa la URL	
	2. El sistema rep la petició GET i retorna la web.
3. L'usuari visualitza la web al navegador i en consulta les dades.	

Emissió de URL

- **Especificació d'alt nivell**

Cas d'ús:	Emissió de URL
Actors:	Principal: Balisa PW Recolzament: Usuari
Propòsit:	Rebre la URL per accedir al sistema en una localització concreta.
Descripció:	L'usuari rebrà una notificació al seu smartphone quan estigui físicament dins del rang d'emissió de la balisa, que estarà situada en la mateixa localització que l'estació meteorològica. Aquesta notificació contindrà la URL del sistema per poder realitzar el cas d'ús 'consultar les dades'.

- **Especificació expandida**

Referències creuades:	
Casos d'ús: Emissió de URL	
Curs típic d'esdeveniments:	
Accions dels actors	**Respostes del sistema**
	1. La balisa està emetent indefinidament la URL dins del seu rang.
2. L'usuari es desplaça a dins de la zona d'emissió.	
3. L'usuari rep una notificació al seu smartphone, mostrant la URL del sistema.	
Requeriments especials:	
El bluetooth del smartphone necessita estar activat.	

Petició de veu

- **Especificació d'alt nivell**

Cas d'ús:	Petició de veu
Actors:	Principal: Usuari (iniciador) Recolzament: Google Assistant, Servidor
Propòsit:	Petició i consulta de les dades de les estacions mitjançant la veu, usant llenguatge natural.
Descripció:	L'usuari inicia la conversa, fent-li una pregunta sobre alguna dada de l'estació que desitja saber, a un dispositiu que executi el software Google Assistant (Google Home, smartphone Android, ...). L'aplicació Assistant demanarà la dada per la qual ha sigut preguntat al servidor i la retornarà de forma oral a l'usuari.

- **Especificació expandida**

Referències creuades:

Casos d'ús: Petició de veu

Curs típic d'esdeveniments:

Accions dels actors	**Respostes del sistema**
1. L'usuari realitza una pregunta al dispositiu amb Google Assistant.	
	2. El dispositiu processa la pregunta i demana les dades al servidor. 3. El servidor retorna les dades al Assistant. 4. Google Assistant contesta a l'usuari amb la resposta de la seva pregunta.
* L'usuari pregunta sobre alguna cosa que el dispositiu amb Assistant no sap contestar:	
	- El sistema informarà l'usuari que no és possible respondre la pregunta.

Enviar a Liquid Galaxy

- **Especificació d'alt nivell**

Cas d'ús:	Enviar a Liquid Galaxy
Actors:	Principal: Usuari (iniciador) Recolzament: Liquid Galaxy, Servidor
Propòsit:	Visualitzar les dades de l'estació en una instal·lació de Liquid Galaxy.
Descripció:	L'usuari està visualitzant les dades de l'estació meteorològica i dóna l'ordre d'enviar-les al Liquid Galaxy. El servidor genera el fitxer KML amb les dades actuals de l'estació i l'envia al Liquid Galaxy, on aquest les mostrarà dins d'un pop-up damunt de la ubicació de l'estació.

- **Especificació expandida**

Referències creuades:

Casos d'ús: Enviar a Liquid Galaxy

Curs típic d'esdeveniments:

Accions dels actors	**Respostes del sistema**
1. L'usuari dóna l'ordre d'enviar les dades al Liquid Galaxy.	
	2. El servidor genera el KML i l'envia al Liquid Galaxy.
	3. El Liquid Galaxy mostra les dades en la ubicació corresponent.

Requeriments especials:
És necessari que hi hagi una instal·lació de Liquid Galaxy en la mateixa xarxa que el servidor.

3.1.5 Diagrames de seqüencia

L'objectiu d'aquest apartat és representar gràficament el flux d'esdeveniments per a un escenari particular d'un cas d'ús.

Comunicació Raspberry Pi - Servidor

La figura 32 representa com l'estació envia les dades al sistema mitjançant una sol·licitud POST[22]. Un cop el servidor rep les dades, les emmagatzema a la base de dades.
Si hi ha un client connectat al sistema visualitzant les dades, actualitza els seus valors en temps real en cada POST.

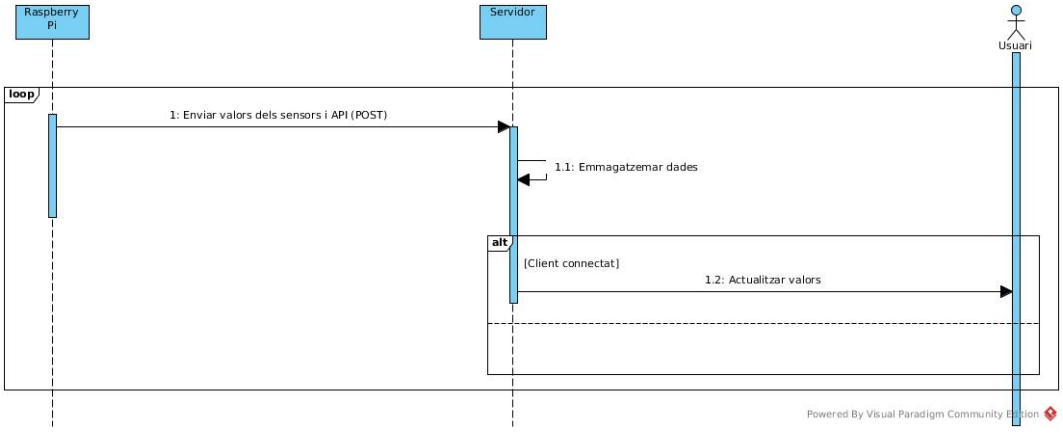

Figura 32: Diagrama de seqüencia entre la comunicació Raspberry Pi - Servidor

Physical web

En el diagrama 33 es mostra com l'usuari, quan entra dins del rang de la balisa Physical Web rep la

[22]És un mètode de l'arquitectura REST que s'utilitza per demanar que el servidor accepti un nou recurs.

notificació en el seu smartphone. En seleccionar la notificació, s'obra el navegador, enviant una petició al servidor i rebent la pagina web amb les dades actualitzades.

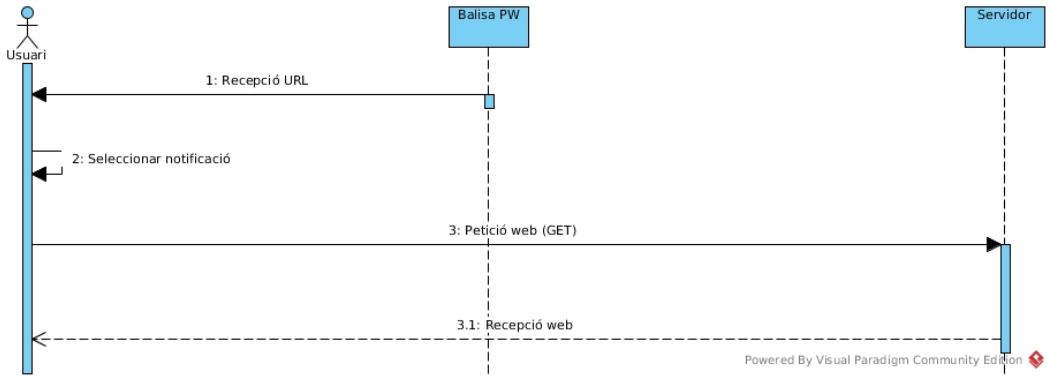

Figura 33: Diagrama de seqüencia Physical web

Liquid Galaxy

La figura 34 detalla quan l'usuari selecciona mostrar les dades de l'estació al Liquid Galaxy. El servidor genera el KML amb les dades actuals i l'envia al Liquid Galaxy que automàticament els mostrarà a les pantalles. Si les dades ja s'estan mostrant, el servidor envia l'ordre d'eliminar les dades i el Liquid Galaxy les elimina.

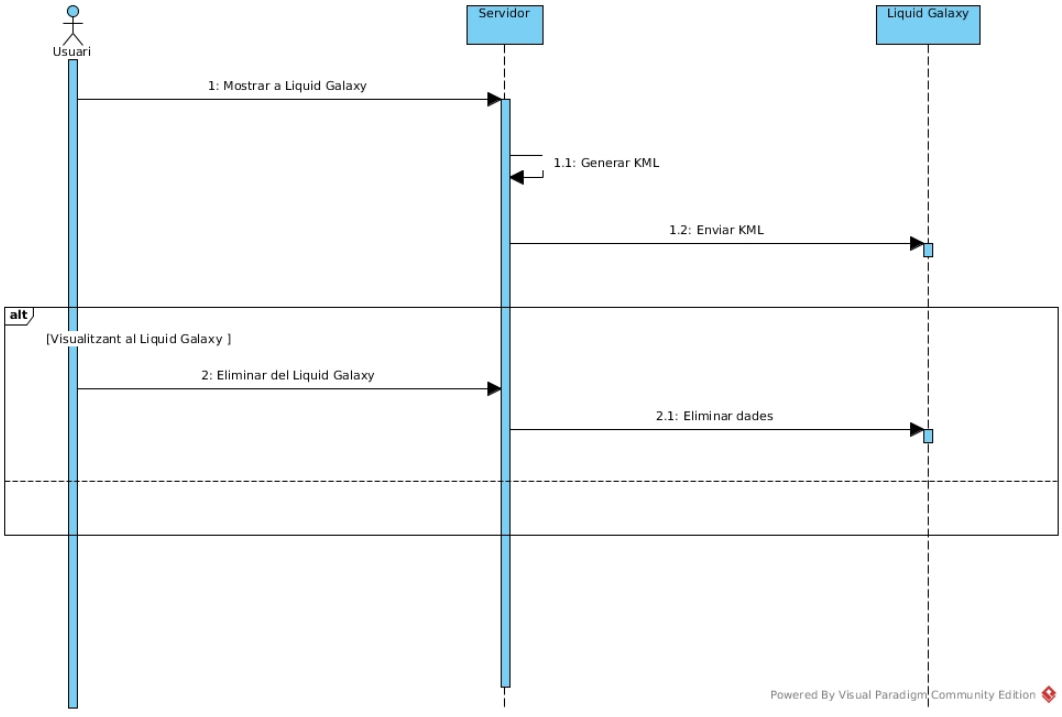

Figura 34: Diagrama de seqüencia Liquid Galaxy

Assistant

L'últim diagrama, en la figura 35 representa la interacció de l'usuari amb la plataforma Google Assistant, on l'usuari pregunta per veu les dades, l'aplicació Assistant consulta les dades al servidor i són retornades a l'usuari mitjançant llenguatge natural.

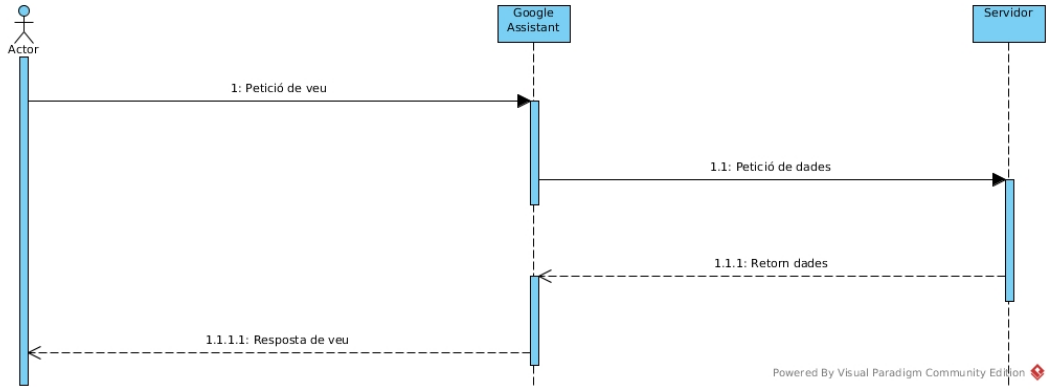

Figura 35: Diagrama de seqüencia Google Assistant

3.2 Anàlisis de costs

A continuació disposarem a analitzar el cost d'aquesta aplicació tenint en compte tant el material com les hores invertides en cada una de les seves parts.

El material utilitzat per cada estació amb el seu cost actual és el següent:

- Raspberry Pi 3B - 37 €
- Targeta de memòria micro SD - 10 €
- Transformador corrent - 10 €
- Sensor de temperatura i humitat DHT22 - 5 €
- Sensor de temperatura i pressió BMP180 - 9 €
- Cables de connexió als sensors - 6 €
- Balisa bluetooth - 15 €
- Google Home - 150 €

Total per estació: 242 €

També necessitarem un hosting per allotjar el servidor. En el nostre cas utilitzarem Google Cloud platform, però hi han altres possibilitats.

- Google Cloud - aprox. 60€/mes (variarà segons la càrrega).

Ara, ja podem calcular el temps a invertir a dissenyar i implementar el sistema. El preu de l'hora variarà en funció de la feina a realitzar. L'enginyer informàtic s'encarregarà de dissenyar i implementar les estacions i el back-end, el dissenyador web/gràfic realitzarà la part visual i del client, i per últim, un especialista a documentar tot el projecte i realitzar aquesta memòria.

- Enginyer informàtic - Preu per hora: 40 €
- Dissenyador gràfic - Preu per hora: 35 €
- Escriptor tècnic - Preu per hora: 30 €

Es contarà la suma d'hores totals de cada apartat d'aquest projecte segons quin professional realitzi la tasca.

Anàlisis

- Enginyer informàtic - 114 hores: 4.560 €

- Escriptor tècnic - 38 hores: 1.140 €

Disseny

- Enginyer informàtic - 70 hores: 2.800 €
- Dissenyador gràfic - 56 hores: 1.960 €
- Escriptor tècnic - 38 hores: 1.140 €

Implementació

- Enginyer informàtic - 100 hores: 4.000 €
- Dissenyador gràfic - 64 hores: 2.240 €
- Escriptor tècnic - 32 hores: 960 €

Validació

- Enginyer informàtic - 35 hores: 1.400 €
- Escriptor tècnic - 10 hores: 300 €

Per realitzar la quantitat total sumarem totes les parts, contant 1 any de hosting i una sola estació meteorològica.

Preu total del projecte: 21.462 €

A continuació es mostra una taula resum de tots els apartats de costs.

Taula resum	
Hardware (1 estació)	242 €
Servei de hosting (1 any)	720 €
Anàlisis	5.700 €
Disseny	5.900 €
Implementació	7.200 €
Validació	1.700 €
Total	21.462 €

4 Disseny

En aquest capítol es parlarà sobre com es guardaran, transmetran i visualitzaran les dades necessàries per al correcte funcionament del projecte.

4.1 Base de dades

Com ja hem vist, utilitzarem la base de dades de firebase estructurada en arbres JSON no relacional. L'estructura serà la que mostra el gràfic de la figura 36.

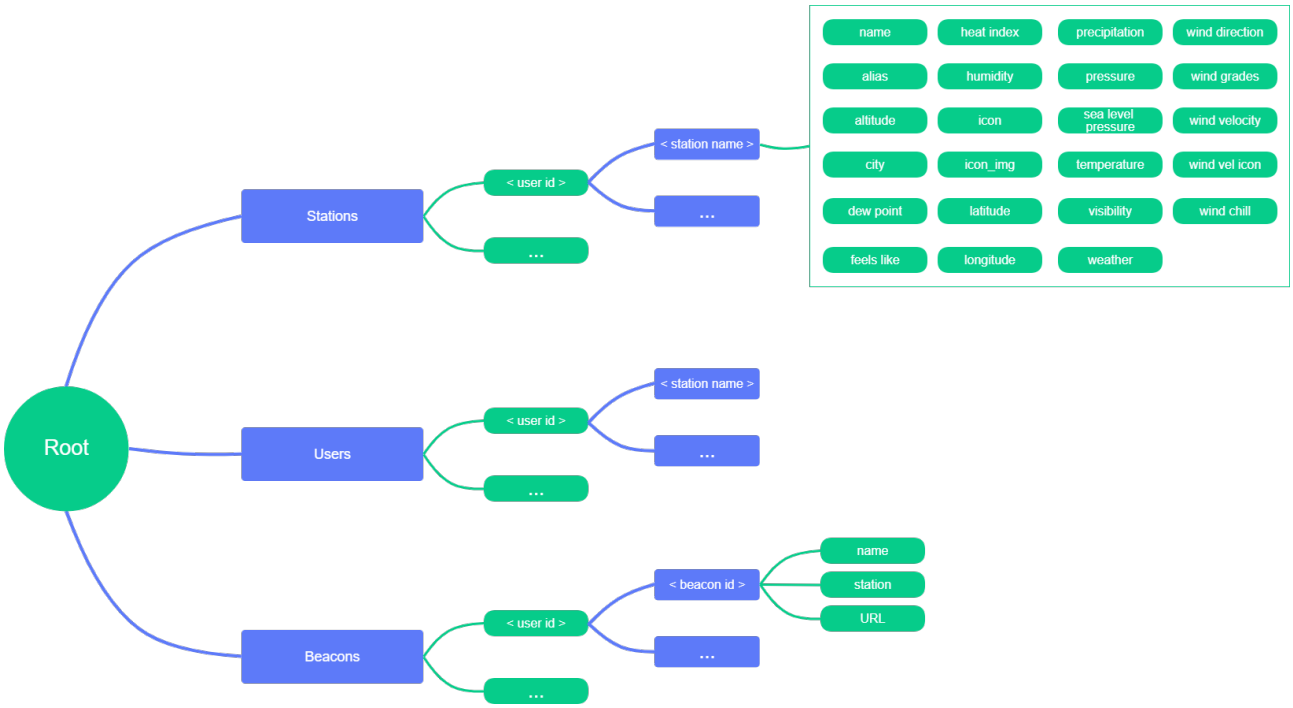

Figura 36: Estructura de la base de dades

La base de dades consta de tres entitats principals:

- **Stations** Conté tota la informació de l'estació.

  Cada fulla contindrà l'identificador de l'usuari on cada un contindrà totes les estacions pertanyents a l'usuari. Cada estació estarà formada pels següents camps:

  - **Name:** Nom que el sistema genera automàticament amb el format StationX, on X es un nombre enter incremental a partir d'1.
  - **Alias:** Nom que posa l'usuari per facilitar la identificació de l'estació.
  - **Altitude:** Altitud de l'estació en metres.
  - **City:** Ciutat on es troba l'estació
  - **Dewpoint:** Punt de rosada en graus centígrads.
  - **Feels like:** Sensació de temperatura en graus centígrads.

- **Heat index:** Temperatura de xafogor en graus centígrads.
- **Humidity:** Humitat de l'estació en percentatge.
- **Icon i icon image:** Indica el temps actual per mostrar a la imatge de la pantalla principal.
- **Latitude:** Latitud de la localització de l'estació per posicionar-la al Liquid Galaxy.
- **Longitude:** Longitud de la localització de l'estació per posicionar-la al Liquid Galaxy.
- **Precipitation:** Precipitació diària de l'estació en mil·límetres.
- **Pressure:** Pressió de l'estació en Pascals.
- **Sea level pressure:** Pressió del nivell del mar de l'estació en Pascals.
- **Temperature:** Temperatura de l'estació en graus centígrads.
- **Visibility:** Visibilitat en kilòmetres.
- **Weather:** Clima actual (per exemple, solejat, nuvolós, plujós, ...).
- **Wind direction:** Direcció del vent.
- **Wind grades:** Graus de la direcció del vent (per exemple, Nord es 0º, Est es 90º, ...)
- **Wind velocity:** Velocitat del vent en kilòmetres per hora.
- **Wind velocity icon:** Indica la imatge a mostrar en funció de la intensitat del vent.

- **Beacons** Conté la informació relativa a les balises.

  Cada fulla contindrà l'identificador de l'usuari on cada un contindrà totes les balises pertanyents a l'usuari. Cada balisa contindrà els següents camps:

  - **Name:** Nom de la balisa.
  - **Station:** Estació que emet.
  - **URL:** URL generada que conté la balisa.

- **Users** Conté la informació de l'usuari. Llista totes les estacions del qual és propietari cada usuari.

Permisos de la base de dades

Les regles de seguretat de la base de dades seran les que mostra el llistat 19. Totes les operacions realitzades pel servidor son en mode administrador, podent ignorar aquestes regles.

```
{
  "rules": {
    "beacons": {
        "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    },
    "stations": {
      ".read": true,
      ".write": false,
      "$uid": {
        ".read": true,
        ".write": "$uid === auth.uid"
```

```
      }
    },
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }

  }
}
```

Listing 19: Regles de seguretat de la base de dades

Això indica que:

- Les balises només podran ser llegides i escrites pel mateix usuari que les crea.
- Les estacions podran ser llegides per tothom, però només escrites per l'usuari al qual pertany.
- Les dades de l'usuari només podran ser llegides i escrites pel mateix usuari.

4.2 Fonts de dades

Les dades que es recollirà cada estació es rebran de dues fonts diferents, dels sensors i de la API.

4.2.1 Sensors

En primer lloc dels sensors connectats a la RPi, que seran els models següents i recolliran les següents dades:

- DHT22: Temperatura i humitat.
- BMP180: Temperatura, pressió atmosfèrica, pressió a nivell del mar i altitud.

Ja que tenim dos valors de temperatura, es realitzarà la mitja dels dos valors.

Connexió dels sensors

Tal com es pot observar en la numeració de la figura 37 disposarem a connectar els sensors de la següent manera:

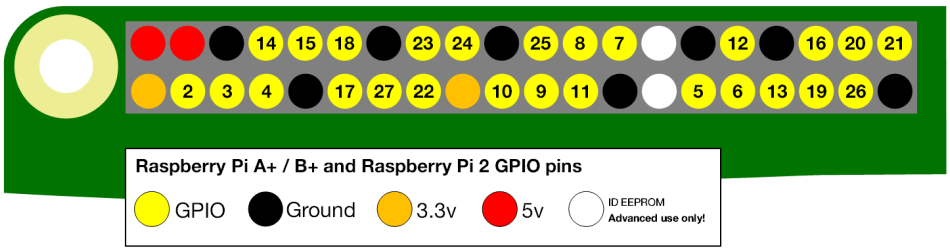

Figura 37: Numeració dels pins GPIO de la Raspberry Pi 2/3

- **Sensor DHT22 (figura 38)**

  Està format per 3 pins: +, out i -.

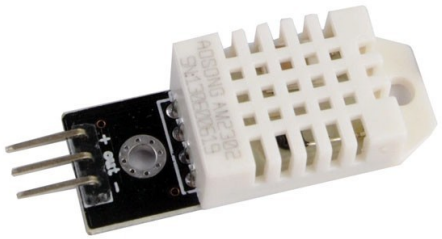

Figura 38: Sensor DHt22

  - \+ al pin de **3.3v**
  - **out** al pin **17**
  - **-** al pin **Ground**

- **Sensor BMP180 (figura 39)**

  Està format per 4 pins: VIN, GND, SCL, SDA.

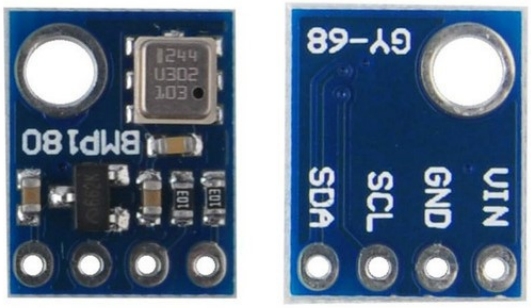

Figura 39: Sensor BMP180

- **VIN** al pin de **3.3v**
- **GND** al pin **Ground**
- **SLC** al pin **3**
- **SDA** al pin **2**

En la figura 40 es mostra un possible resultat final de la RPi amb els dos sensors connectats.

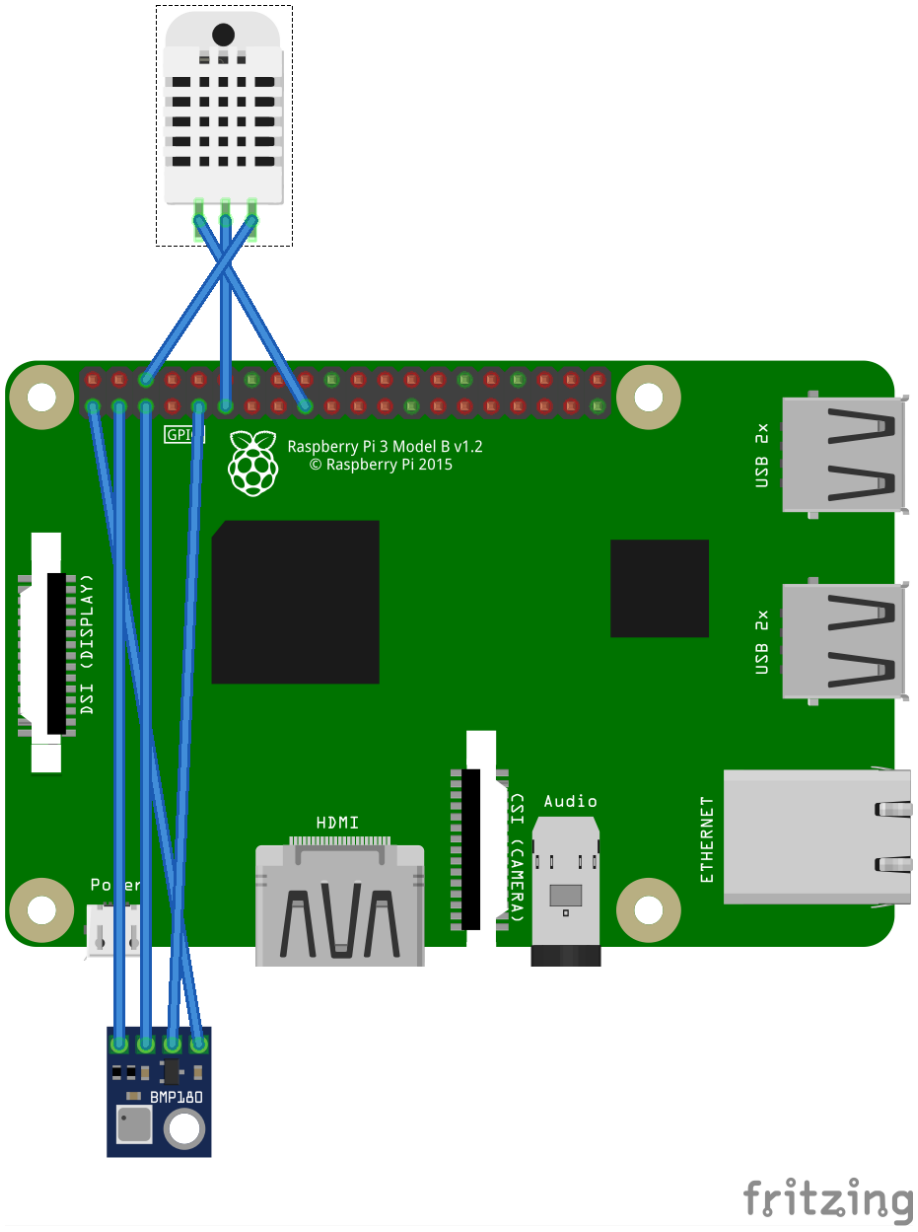

Figura 40: Esquema de la Raspberry Pi amb els senors connectats

4.2.2 API (Weather Underground)

Per completar les dades de l'estació, s'agafaran la resta de dades de l'API Weather Underground.

Es farà servir la característica: condicions, que retorna la temperatura actual, la condició del clima, la humitat, el vent, la temperatura "sensació de temperatura", la pressió baromètrica i la visibilitat.

L'usuari, en crear una estació, ha de seleccionar la localització d'aquesta. S'utilitzarà aquesta localització per rebre les dades de la zona.

El servei retornarà les dades en format JSON. Podem veure com fent una petició a l'URL:

http://api.wunderground.com/api/api_key/conditions/q/ES/Lleida.json

Retornarà les dades en el següent format:


```
{
  "response": {
    "version":"0.1",
    "termsofService":"http://www.wunderground.com/weather/api/d/terms.html",
    "features": {
      "conditions": 1
    }
  }
  , "current_observation": {
    "image": {
          "url":"http://icons.wxug.com/graphics/wu2/logo_130x80.png",
          "title":"Weather Underground",
          "link":"http://www.wunderground.com"
    },
    "display_location": {
          "full":"Lleida, Spain",
          "city":"Lleida",
          "state":"L",
          "state_name":"Spain",
          "country":"SP",
          "country_iso3166":"ES",
          "zip":"00000",
          "magic":"18",
          "wmo":"08171",
          "latitude":"41.61999893",
          "longitude":"0.62000000",
          "elevation":"123.1"
    },
    "observation_location": {
          "full":"Lleida, Lerida, L",
          "city":"Lleida, Lerida",
          "state":"L",
          "country":"SP",
          "country_iso3166":"ES",
          "latitude":"41.615494",
          "longitude":"0.635525",
          "elevation":"0 ft"
    },
    "estimated": {
    },
    "station_id":"ILLERIDA2",
    "observation_time":"Last Updated on August 30, 2:49 PM CEST",
    "observation_time_rfc822":"Wed, 30 Aug 2017 14:49:20 +0200",
    "observation_epoch":"1504097360",
    "local_time_rfc822":"Wed, 30 Aug 2017 14:54:00 +0200",
    "local_epoch":"1504097640",
    "local_tz_short":"CEST",
```

```
    "local_tz_long":"Europe/Madrid",
    "local_tz_offset":"+0200",
    "weather":"Scattered Clouds",
    "temperature_string":"79.2 F (26.2 C)",
    "temp_f":79.2,
    "temp_c":26.2,
    "relative_humidity":"59%",
    "wind_string":"From the WSW at 1.9 MPH Gusting to 16.8 MPH",
    "wind_dir":"WSW",
    "wind_degrees":251,
    "wind_mph":1.9,
    "wind_gust_mph":"16.8",
    "wind_kph":3.1,
    "wind_gust_kph":"27.0",
    "pressure_mb":"1014",
    "pressure_in":"29.93",
    "pressure_trend":"-",
    "dewpoint_string":"64 F (18 C)",
    "dewpoint_f":64,
    "dewpoint_c":18,
    "heat_index_string":"81 F (27 C)",
    "heat_index_f":81,
    "heat_index_c":27,
    "windchill_string":"NA",
    "windchill_f":"NA",
    "windchill_c":"NA",
    "feelslike_string":"81 F (27 C)",
    "feelslike_f":"81",
    "feelslike_c":"27",
    "visibility_mi":"34.0",
    "visibility_km":"55.0",
    "solarradiation":"--",
    "UV":"-1","precip_1hr_string":"-999.00 in ( 0 mm)",
    "precip_1hr_in":"-999.00",
    "precip_1hr_metric":" 0",
    "precip_today_string":"0.00 in (0 mm)",
    "precip_today_in":"0.00",
    "precip_today_metric":"0",
    "icon":"partlycloudy",
    "icon_url":"http://icons.wxug.com/i/c/k/partlycloudy.gif",
    "forecast_url":"http://www.wunderground.com/global/stations/08171.html",
    "history_url":"http://www.wunderground.com/weatherstation/WXDailyHistory.asp?ID=
    ILLERIDA2",
    "ob_url":"http://www.wunderground.com/cgi-bin/findweather/getForecast?query
    =41.615494,0.635525",
    "nowcast":""
  }
}
```


Listing 20: Resposta de l'API Wunderground a una petició

4.3 Enviament i recepció de dades

En la figura 41 es mostra el procés que es realitza en l'enviament, recepció i emmagatzemament de les dades.

La RPi executarà uns scripts que estaran indefinidament enviant les dades al servidor en un bucle infinit que arrancarà un servei automàticament al l'inici. Cada lectura enviarà una petició POST al nostre servidor amb les noves dades.

El script que rep les dades dels sensors realitzarà el POST a */sensors*. El que rep les dades de l'API el realitzarà a */api* i ho farà cada 3 minuts a causa de la limitació de peticions diàries de la clau gratuïta d'aquest servei.

En el moment que el servidor rebi les dades, les guardarà a la base de dades remota de firebase en la ruta *root/id usuari/nom estació/*.

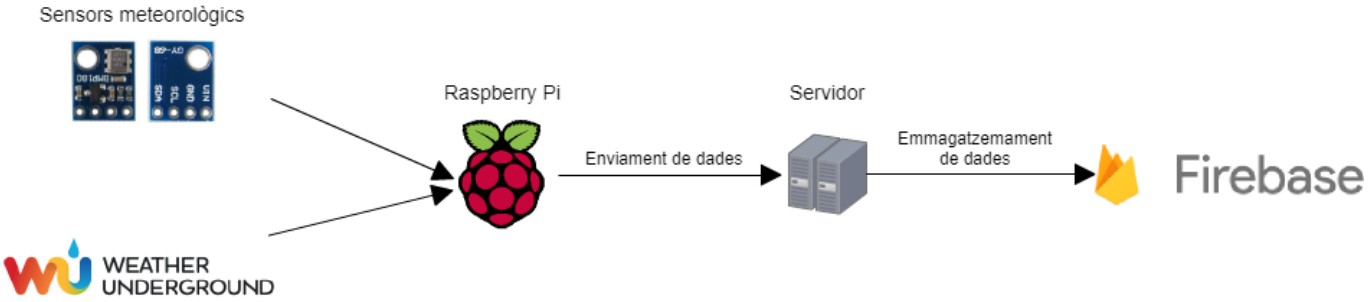

Figura 41: Enviament, recepció i emmagatzemament de les dades

4.4 Visualització de les dades

A continuació es recolliran les diferents formes en què l'usuari pot visualitzar les dades.

4.4.1 Interfície web

El sistema disposarà d'una aplicació web on es podran visualitzar les dades de les estacions.

Tal com s'observa en la figura 42, la interfície consta de:

- Una barra lateral amb totes les categories.
- Una barra horitzontal superior amb les opcions d'autentificació d'usuari, i la llista d'estacions mostrades en diferents pestanyes.
- L'espai central on es representarà totes les dades de l'estació en temps real.

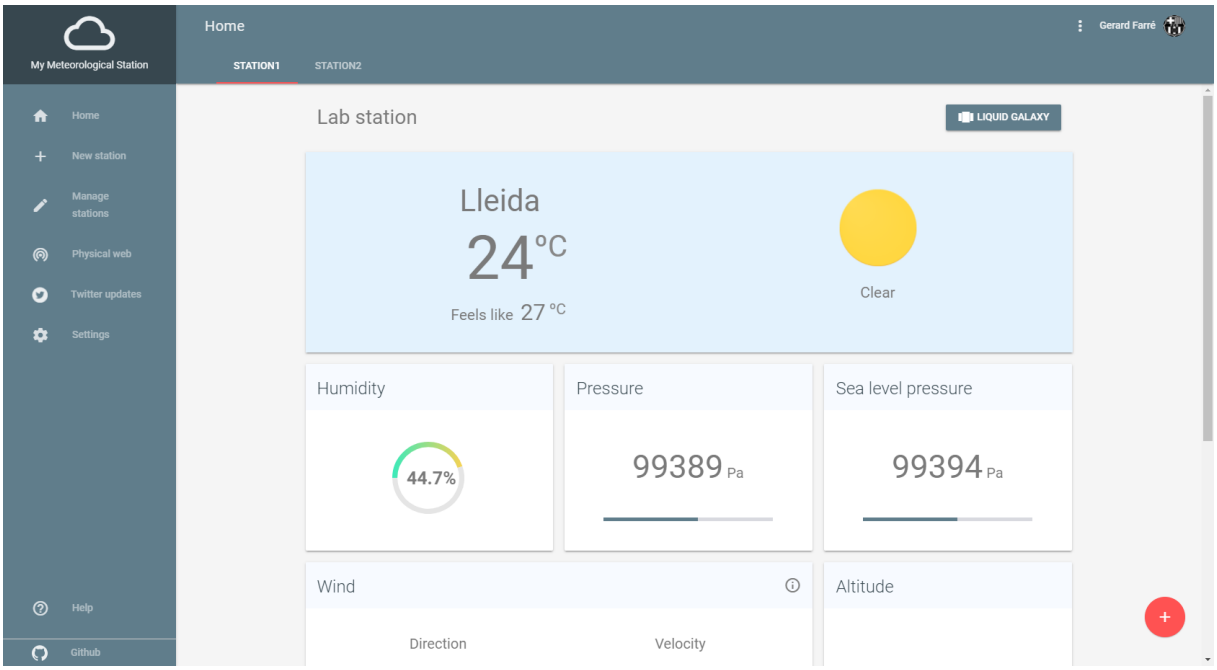

Figura 42: Pantalla principal de l'aplicació web

El sistema s'haurà d'adaptar a diferents tipus de mides de pantalla així com en diferents dispositius. En la figura 43 es mostra la interfície adaptada a un smartphone de 5,7 polzades.

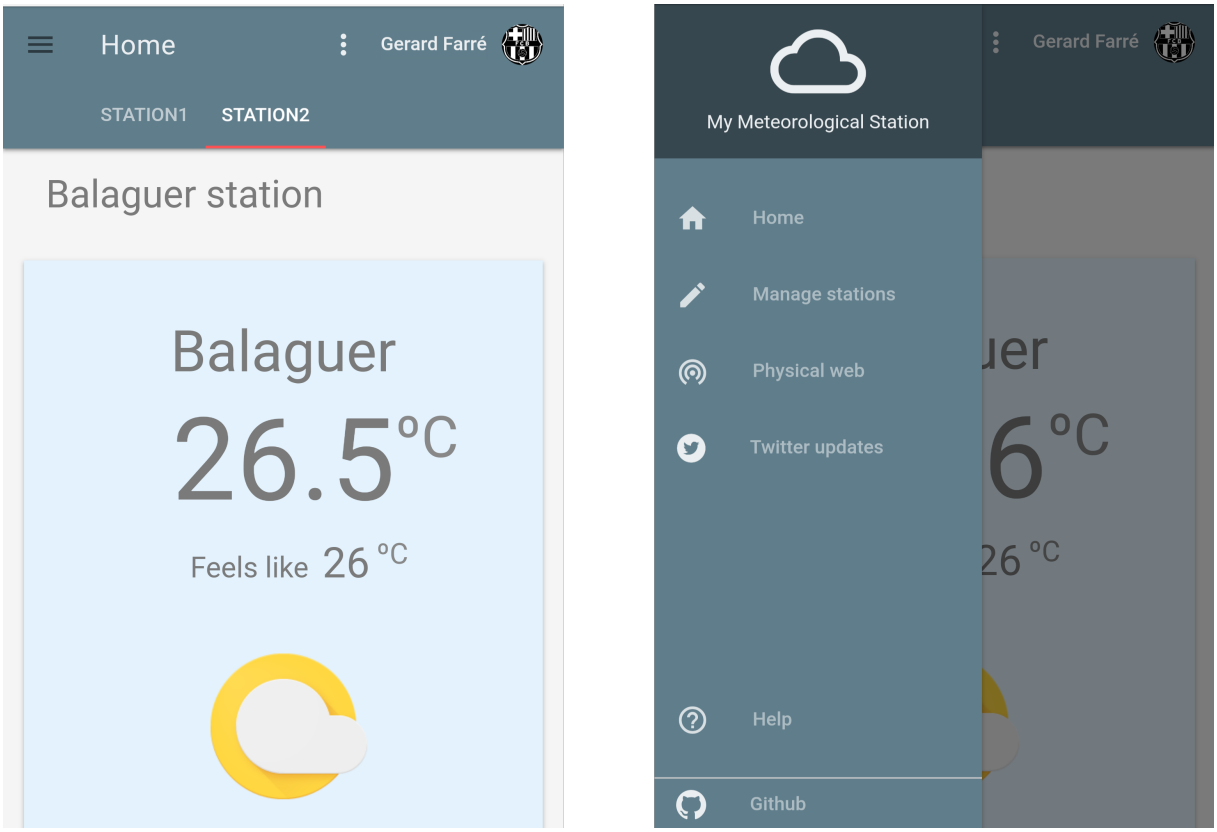

Figura 43: Pantalla principal de l'aplicació web en mode smartphone

4.4.2 Physical web

Una forma que l'usuari pugui accedir fàcilment a l'aplicació web per visualitzar les dades és mitjançant les balises. Aquest sistema el gestiona el sistema operatiu Android per defecte.

Quan l'usuari estigui a prop de l'estació, rebrà una notificació com la que s'observa en la figura 44. En el moment que l'usuari cliqui aquesta notificació, s'obrirà el navegador web que mostrarà la interfície de la nostra aplicació directament a la pestanya de l'estació propera.

Un usuari podrà crear una balisa i assignar-li una estació per emetre. L'estació que emet la balisa es podrà modificar des de l'aplicació sense necessitat de reescriure la balisa, essent gestionades pel servidor.

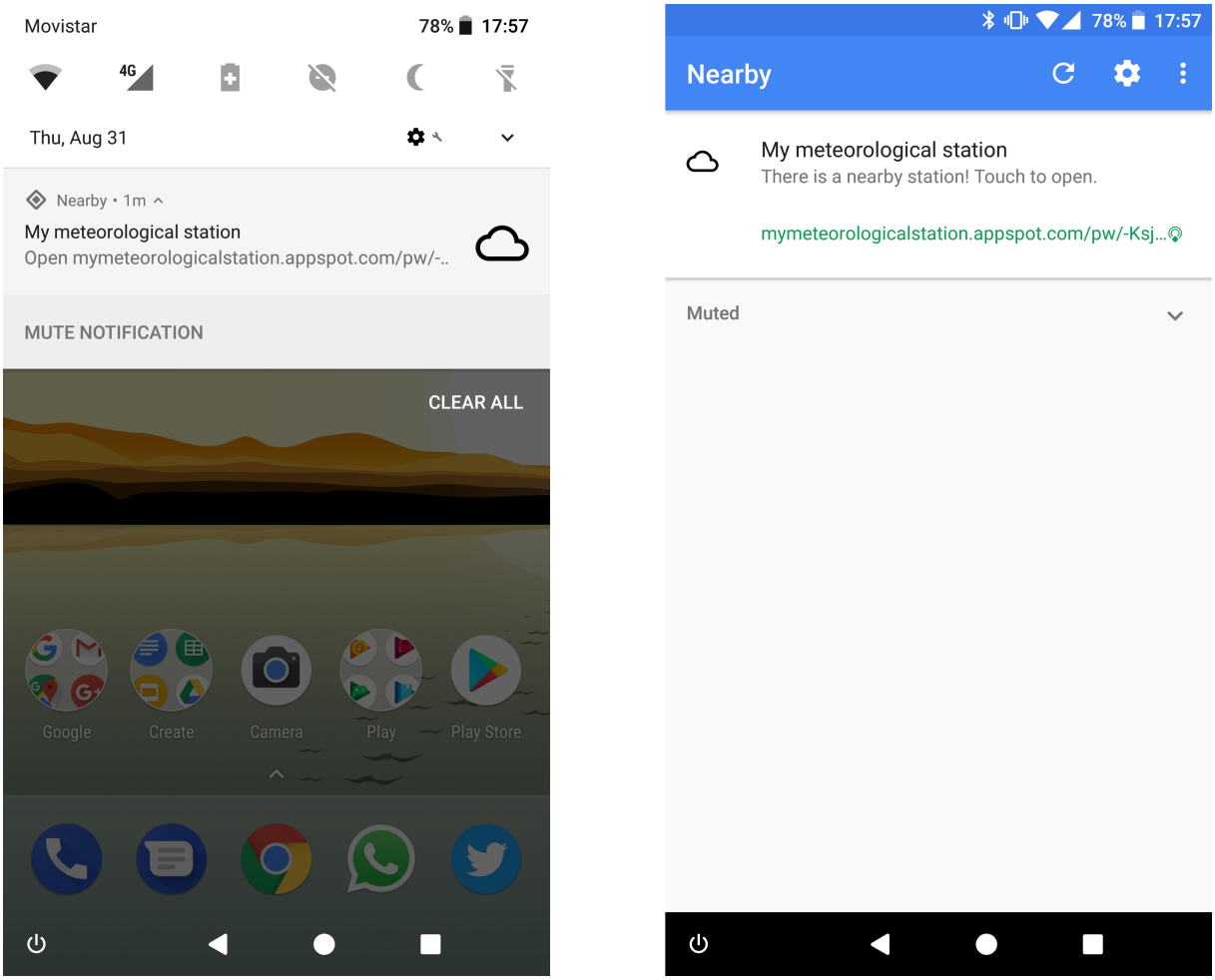

Figura 44: Notificació Physical Web

4.4.3 Liquid Galaxy

També es podrà visualitzar les dades de l'estació en el Liquid Galaxy. En la figura 45 es veu com des de l'aplicació web hi haurà un botó desplegable amb les diferents opcions de visualització.

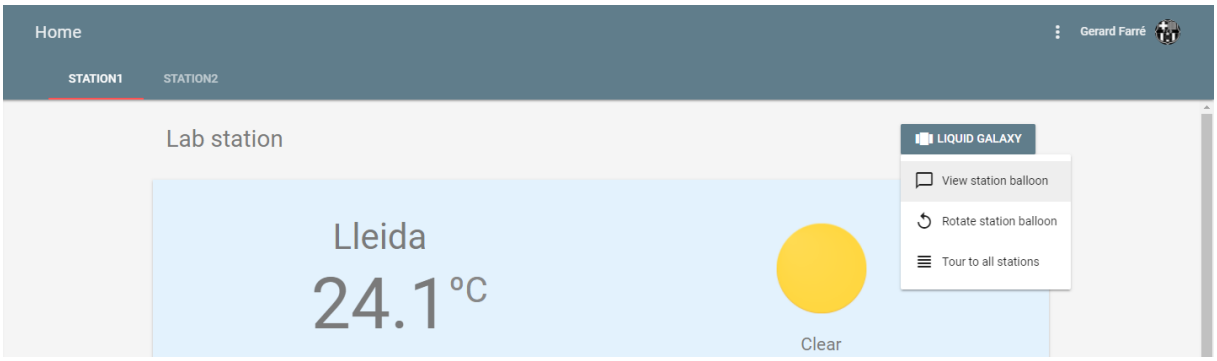

Figura 45: Menú desplegable per al Liquid Galaxy

- **Mostrar un globus de l'estació:** Vola fins al punt on hi ha l'estació i hi mostra un globus amb tots els valors de l'estació, tal com es veu a la figura 46.
- **Rotació en òrbita del globus:** Realitza una òrbita de 360° sobre el globus.
- **Tour a totes les estacions:** Fa un recorregut de totes les estacions disponibles, parant uns segons en cada estació i mostrant el seu globus.

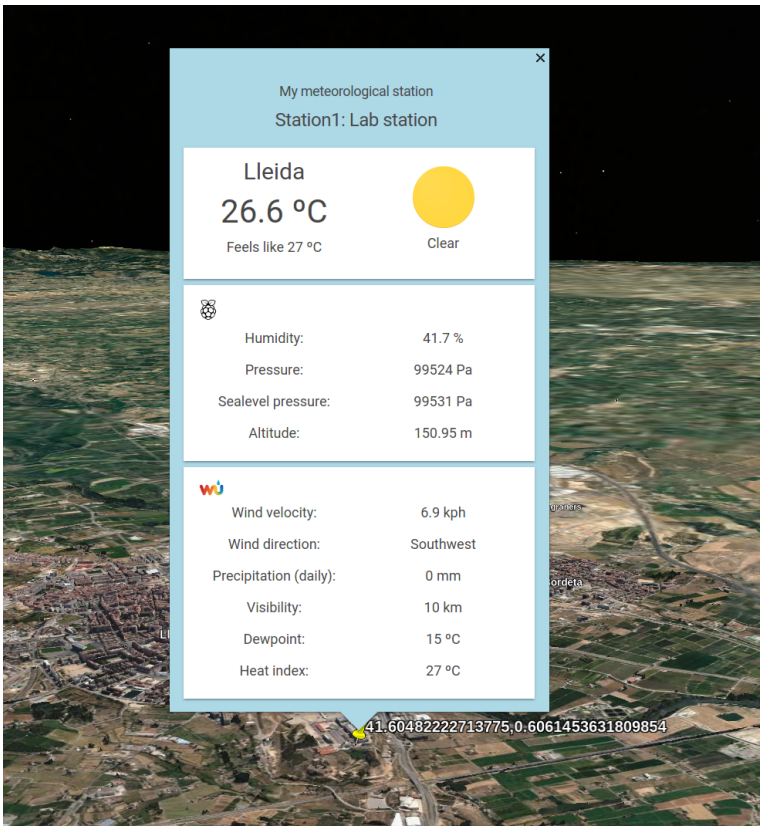

Figura 46: Globus d'una estació mostrada al Liquid Galaxy

El globus separa en targetes les dades que rep dels sensors i de l'API i els identifica mitjançant una icona a la part superior esquerra (excepte la primera targeta, que mescla dades de tots dos).

4.4.4 Google Assistant

Des de la tecnologia Google Assistant podrem consultar dades de les estacions així com enviar dades al Liquid Galaxy.

Com ja s'ha comentat anteriorment, aquesta tecnologia només suporta dos idiomes actualment, l'Anglès i l'Alemany. S'han anunciat nous idiomes que afegiran pròximament però encara no estan disponibles. A causa d'aquestes circumstàncies, la realització dels diàlegs es realitzarà en l'idioma Anglès.

Es pot preguntar per:

- **Un valor d'una estació concreta: (Figura 47)**

  What's the **{value}** of the **{station name}**?

  On value pot ser:

  - Temperature
  - Humidity
  - City
  - Weather
  - Pressure
  - Sea level pressure
  - Altitude
  - Wind velocity

  Exemple: What's the temperature of the Station 1?

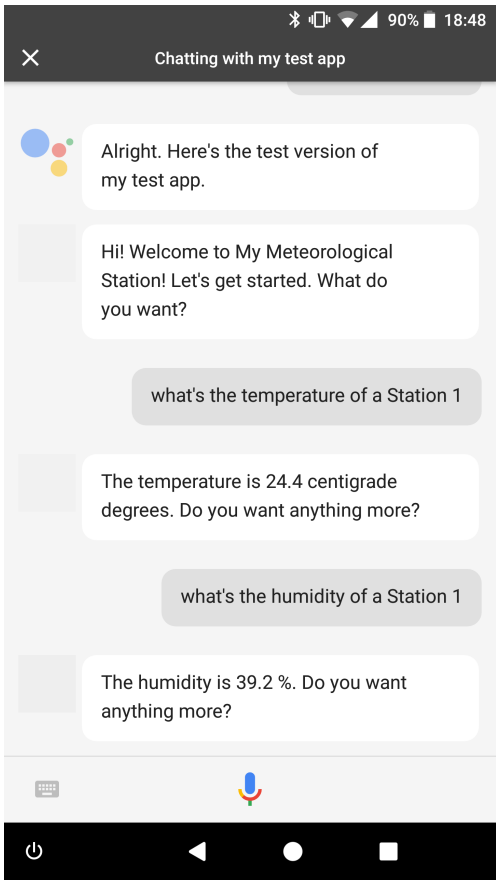

Figura 47: Conversa amb Google Assistant preguntant per un valor concret d'una estació

- **Tots els valors d'una estació concreta:**
  What's the status of the **{station name}**?
  Exemple: What's the status of the station 1?
- **Tots els valors de totes les estacions:**
  All the stations information

Accions per el Liquid Galaxy:

- **Mostrar el globus d'una estació:**
  Show me the **{station name}** balloon
  Exemple: Show me the station 3 balloon?
- **Rotació en orbita del globus:**
  Rotate **{station name}**
  Exemple: Rotate station 1?
- **Tour a totes les estacions:**
  Make a complete tour
- **Neteja les dades del Liquid Galaxy**
  Clean Liquid Galaxy

4.4.5 Twitter

Per la publicació de les dades de l'estació en el compte de Twitter es definiran les següents característiques:

- Es publicarà un tuit per cada estació.
- Es publicaran els tuits de totes les estacions cada 24 hores.

El format d'un tuit d'una estació serà el següent:

Station1: Lab station
City: Lleida
Weather: Clear
Temperature: 25.8 ºC
Humidity: 44.8 %
Pressure: 99445 Pa

Només es mostraran aquests camps a causa de la limitació de 140 caràcters per tuit.

5 Implementació

En aquest capítol s'implementaran totes les parts d'aquest projecte, definides al llarg de la memòria.

5.1 Estructura del projecte

A continuació es mostra un resum de l'estructura del projecte i que conté cada directori.

- `/`
  - `package.json` ..... Conté tota la informació i dependències del projecte.
  - `app.js` ..... Fitxer principal per arrencar el servidor.
  - `views` ..... Conté totes les plantilles de les vistes.
    - `scripts` ..... Emmagatzema tots els scripts que s'executaran al client.
  - `routes` ..... Estableix totes les rutes del servidor per a les peticions GET, POST, ...
  - `rpi-configuration` ..... Conté el codi de configuració automàtic per l'estació.
    - `scripts` ..... Emmagatzema els scripts que es copiaran a l'estació.
      - `services` ..... Conté les plantilles per generar els serveis.
  - `public` ..... Guarda les dades estàtiques del servidor.
    - `css` ..... Conté els estils per les vistes.
    - `img` ..... Conté totes les imatges del projecte.
    - `templates` ..... Guarda les plantilles usades per generar els fitxers KML.
  - `firebase` ..... Conté totes les funcions de comunicació amb la base de dades.
  - `liquidgalaxy` ..... Conté el codi de comunicació i generació de KML del Liquid Galaxy.
  - `assistant` ..... Conté tot el codi necessari pel Webhook de Google Assistant.
    - `agent` ..... Projecte de l'API.AI exportat (per poder importar-lo en un altre agent).
  - `twitter` ..... Codi relatiu a l'aplicació de Twitter.

5.2 Servidor

L'aplicació serà realitzada amb Node.js utilitzant el mòdul express.

El servidor arrancarà al port 3000 o en la variable d'entorn PORT (utilitzat per alguns entorns com Heroku[23]), tal com es mostra en el codi del llistat 21.

```
1 http.listen(process.env.PORT || 3000, function(){
2     console.log('listening on *:3000');
3 });
```

Listing 21: Codi d'arrencada del servidor

5.3 Firebase

Per tal que l'aplicació es pugui comunicar amb el projecte firebase necessitarem descarregar la configuració del projecte a utilitzar per al client, i la configuració d'administrador pel servidor.

Primer es crearà el fitxer **firebase-config.json** a l'arrel del projecte amb el següent contingut:

```
1 {
```

[23] Heroku es una plataforma de servei de computació al nuvol

```
  "apiKey": "",
  "authDomain": "",
  "databaseURL": "",
  "projectId": "",
  "storageBucket": "",
  "messagingSenderId": ""
}
```

Listing 22: Contingut del fitxer firebase-config.json

Dins de visió general, clicarem el botó: Afegeix firebase,a la teva aplicació web, tal com s'observa a la figura 48, i ens mostrarà la configuració amb la qual hem d'emplenar els camps del fitxer que acabem de crear.

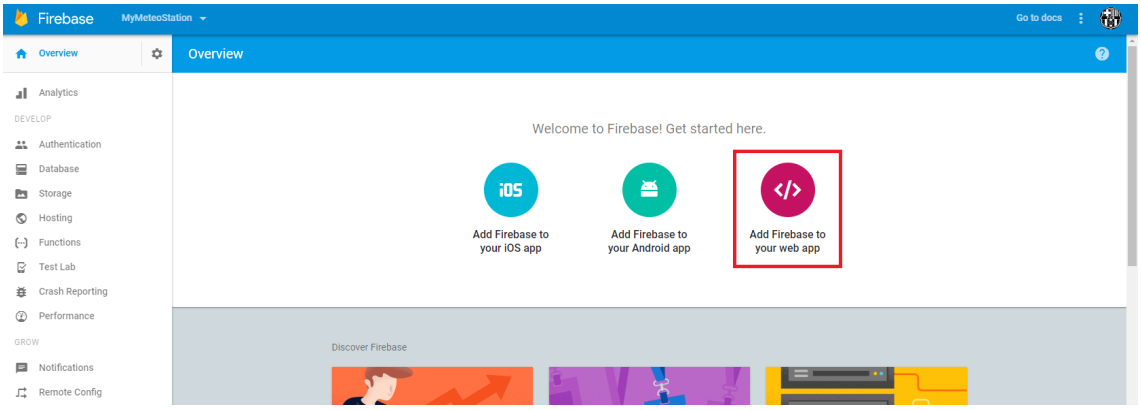

Figura 48: Botó per rebre la configuració de firebase del client

Per rebre la configuració d'administrador, com es pot veure a la figura 49, ens situarem a la configuració del projecte, en la pestanya comptes de servei i generarem una nova clau privada, i ens descarregarà un arxiu que canviarem de nom a **firebase-admin.json** i el guardarem a l'arrel del projecte.

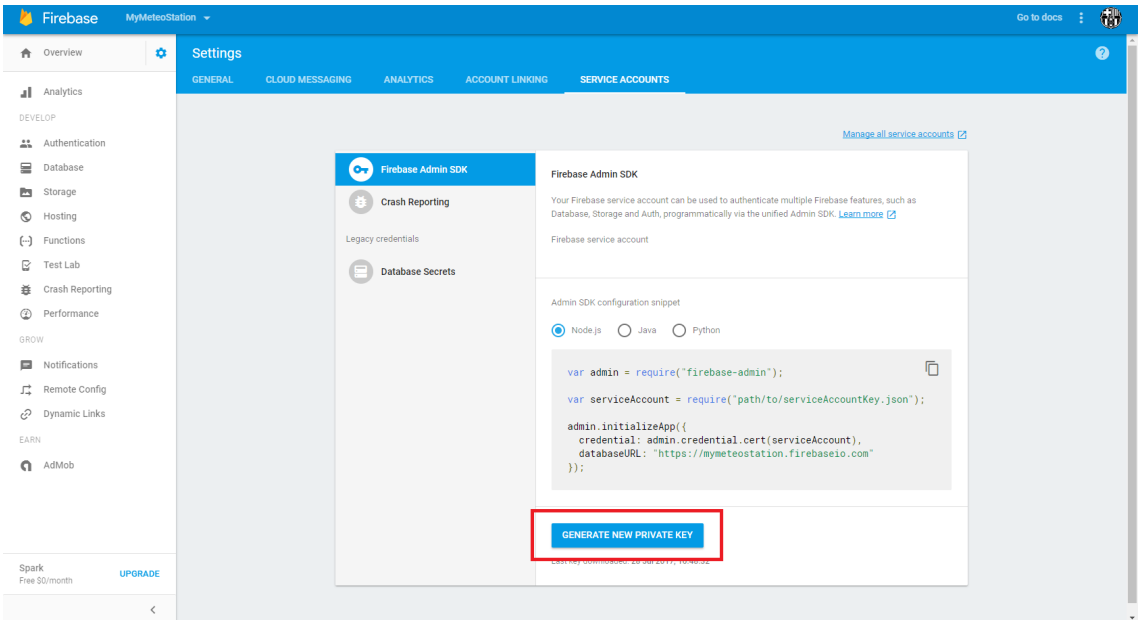

Figura 49: Botó per rebre la configuració de Firebase en administrador

Per carregar i inicialitzar la configuració del client utilitzarem el següent codi:

```
const config = require('./firebase-config.json');
firebase.initializeApp(!{JSON.stringify(config)});
```

Listing 23: Carrega i inicialització de la configuració de Firebase

I per la configuració d'administrador:

```
const serviceAccount = require("./firebase-admin.json");
admin.initializeApp({
    credential: admin.credential.cert(serviceAccount),
    databaseURL: config.databaseURL
});
```

Listing 24: Carrega i inicialització de la configuració de Firebase en administrador

Ara que ja es té el servidor preparat per gestionar la base de dades es llistaran les funcions més importants per llegir i escriure les dades del projecte i l'autenticació d'usuaris.

5.3.1 Lectura de dades

Així doncs, per llegir totes les estacions disponibles, farem referencia a la ruta de la base de dades "/stations/", que ens retornarà tots els userId amb totes les seves estacions, per tant els iterarem i guardarem en una llista, que retornarem.

```
function readStations () {

    return new Promise(function (resolve, reject) {
        admin.database().ref('/stations/').once('value').then(function (snapshot) {

            let data = [];
            for (const user in snapshot.val()) {
                for (const station in snapshot.child(user).val()) {
                    data.push(snapshot.child(user).child(station).val());
                }
            }
            return resolve(data);

        }).catch(function (error) {
            return reject(error);
        });
    });

}
```

Listing 25: Funció per rebre les dades de totes les estacions

S'utilitza una promesa pel fet que les crides a Firebase són asíncrones, i es voldrà executar cert codi només si s'han rebut les dades. Així, la crida de la funció seguirà aquest format:

```
firebase.readStations().then( data => {
  // Realitza qualsevol accio amb les dades de data
});
```

Listing 26: Crida a la funció de lectura de totes les estacions

Utilitzant la mateixa idea, es pot llistar les dades d'una estació concreta que coneixem el nom, retornant les dades quan l'estació coincideixi amb el nom que busquem en lloc de guardar-les a una llista.

```
function readStationData (name) {
    return new Promise(function (resolve, reject) {
        admin.database().ref('/stations/').once('value').then(function(snapshot) {
            for (const user in snapshot.val()) {
                for (const station in snapshot.child(user).val()) {
                    if (station === name) {
                        return resolve(snapshot.child(user).child(station).val());
                    }
                }
            }
        }).catch(function (error) {
            return reject(error);
        });
    });
}
```

Listing 27: Funció que retrona les dades d'una estació

O també podem llistar totes les estacions que pertanyen a un usuari utilitzant la informació de la taula '/users/' seguint el mateix procediment.

```
function getUserStations (uid) {
    return new Promise(function (resolve, reject) {
        admin.database().ref('/users/' + uid).once('value').then(function(snapshot) {

            let data = [];
            for (const station in snapshot.val()) {
                data.push(snapshot.child(station).val());
            }
            return resolve(data);

        }).catch(function (error) {
            return reject(error);
        });
    });
}
```

Listing 28: Funció que retrona les estacions que pertanyen a un usuari

5.3.2 Escriptura de dades

L'escriptura de dades es realitzarà utilitzant la funció update, que actualitzarà les dades si ja existeixen, o les crearà en cas contrari. La ruta on es guardaran serà '/stations/{userId}/{station name}'.

```
function writeStationSensors (data) {

    isValidStation(data.uid, data.name).then(valid => {
        if (valid) {
            admin.database().ref('stations/' + data.uid + '/' + data.name).update({
                name: data.name,
                temperature: utils.round(
                  (Number(data.temperature)+Number(data.temperature2))/2
```

```
                ),
                humidity: utils.round(data.humidity),
                pressure: utils.round(data.pressure),
                sealevel_pressure: utils.round(data.sealevel_pressure),
                altitude: data.altitude
            });
        }
    });

}
```

Listing 29: Funció que guarda els valors dels sensors a la base de dades

Es verifica abans de guardar les dades, que l'usuari que vol modificar les dades sigui el propietari de l'estació cridant la funció 'isValidStation()'.

Utilitzarem el mateix sistema per guardar les dades de l'API a la base de dades.

5.4 Interfície

La pàgina inicial se servirà en la direcció arrel '/'. Es renderitzarà passant-li les dades de totes les estacions.

```
router.get('/', function(req, res){

    firebase.readStations().then( data => {

        res.render('index',{
            title: 'Home',
            data: data,
            env: env,
            config: config,
            id: req.query.id
        });

    });

});
```

Listing 30: Renderització de la pagina inicial

També es passarà la variable env, que indica si el servidor està corrent en producció (al cloud) o en local.

```
const env = process.env.NODE_ENV || 'development';

if (env !== 'production') //Running in production
else //Running in local
```

Això permetrà a la interfície mostrar o amagar opcions segons on s'estigui executant el servidor. Per exemple, tant l'opció de crear i configurar una estació, com la d'enviar dades al Liquid Galaxy, es realitzen per ssh, i un requisit d'aquest és que estiguin els dispositius en la mateixa LAN. Si el servidor es troba en mode producció les opcions de Nova estació i Liquid Galaxy no apareixeran.

5.4.1 Google Sign in

L'autenticació d'usuaris amb compte de Google també és gestionada per Firebase.

Afegirem el codi de sign in al botó.

```
$('#signinbtn').on('click', function (event) {

  const provider = new firebase.auth.GoogleAuthProvider();

  firebase.auth().signInWithPopup(provider).then(function (result) {
  }).catch(function (error) {
  console.log(error);
  });

});
```

Per sortir cridarem la funció signOut.

L'aplicació està en tot moment escoltant un canvi d'autenticació, això permet mostrar els menús de creació d'una estació o una balisa quan l'usuari s'ha autenticat, o amagar-los quan surt.

```
firebase.auth().onAuthStateChanged(function (user) {
  if (user) {
  // User is signed in.
  $('#managestations').css({display: 'block'});
  ...

  } else {
  // No user is signed in.
  $('#managestations').css({display: 'none'});
  ...
  }
});
```

Listing 31: Funció que escolta els canvis d'autenticació

5.5 Estacions

En la següent subsecció es mostrarà tant la interfície d'usuari que s'utilitzarà per crear una estació, com les tasques que generarà el sistema en la creació d'aquesta.

5.5.1 Interfície d'usuari

Per crear una estació ens dirigirem a l'apartat *New station* i *Create a station*, com es pot veure a la figura 50.

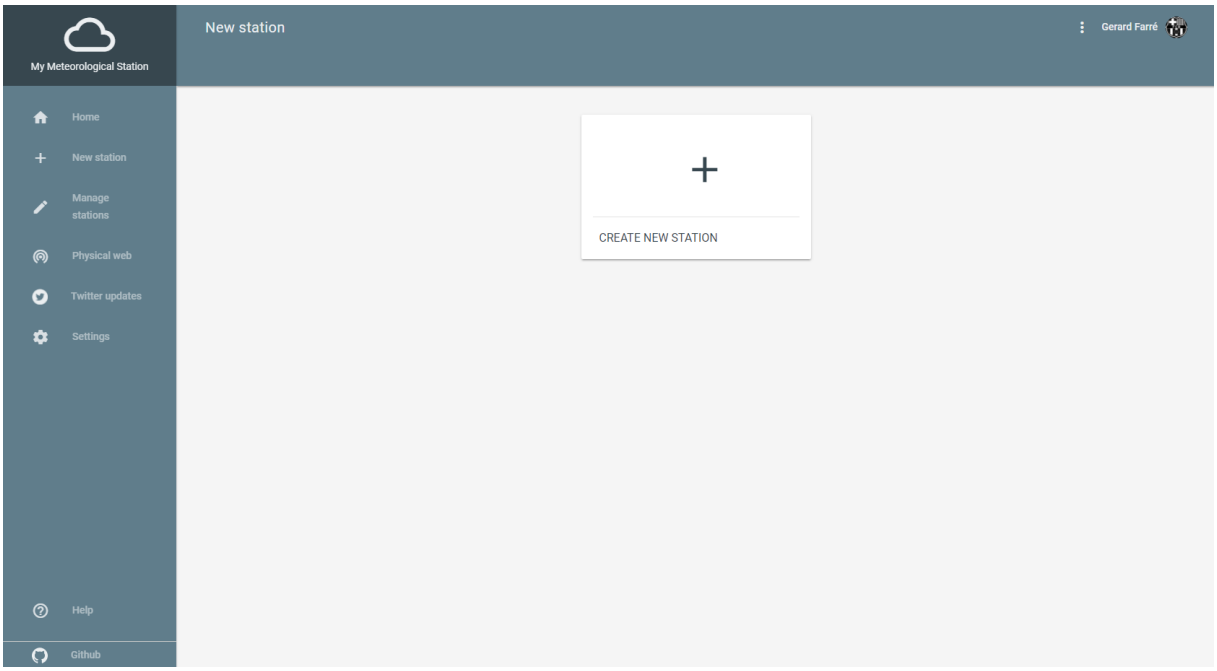

Figura 50: Apartat de creació d'una nova estació

Necessitarem omplir un formulari amb les dades que ens demana (fig. 51).

- Estació: L'àlies i la localització de l'estació seleccionant un punt en el mapa.
- Raspberry Pi: El nom d'usuari, la contrasenya d'usuari i la ip
- Servidor: S'introduirà la URL del servidor si està en producció o la ip:port si està en local. És la direcció on l'estació realitzarà les crides POST per enviar-li les dades.
- Wunderground API; Clau de la Api proporcionada per Weather Underground

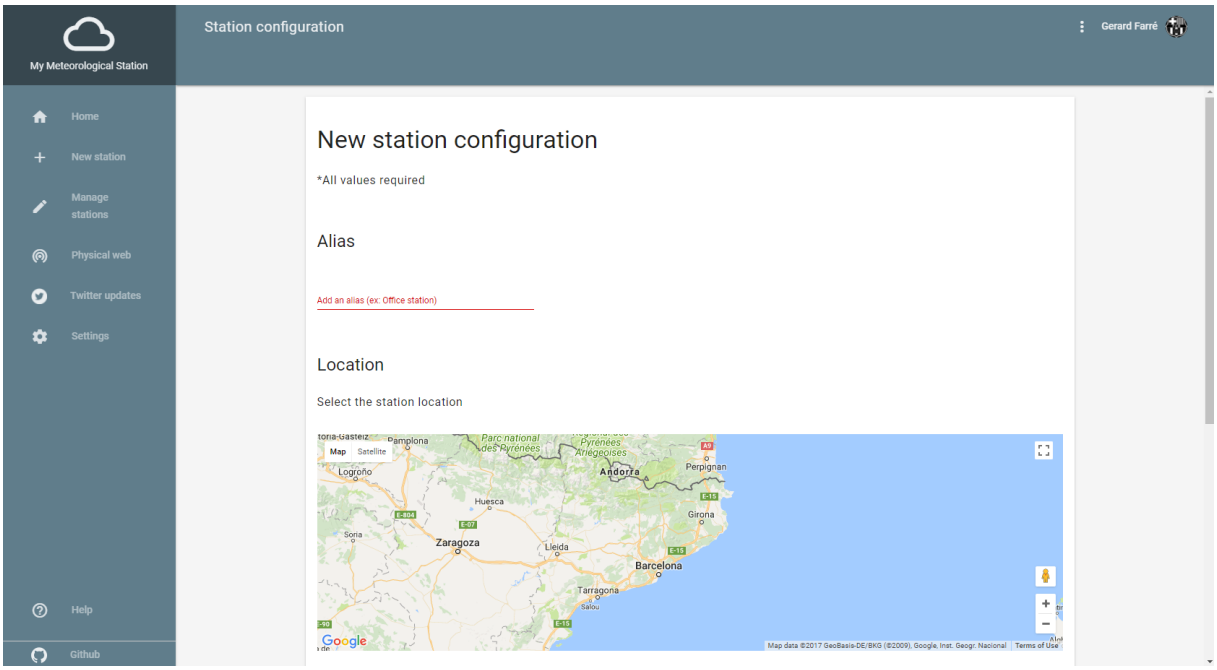

Figura 51: Formulari de creació d'una nova estació

L'usuari podrà editar o eliminar l'estació en l'apartat "manage stations", com es veu a la figura 52. L'edició de l'estació es realitzarà seguint el mateix procediment que en la creació. Per eliminar una estació serà necessari confirmar-ho en un diàleg (fig. 53) per evitar clics indesitjats.

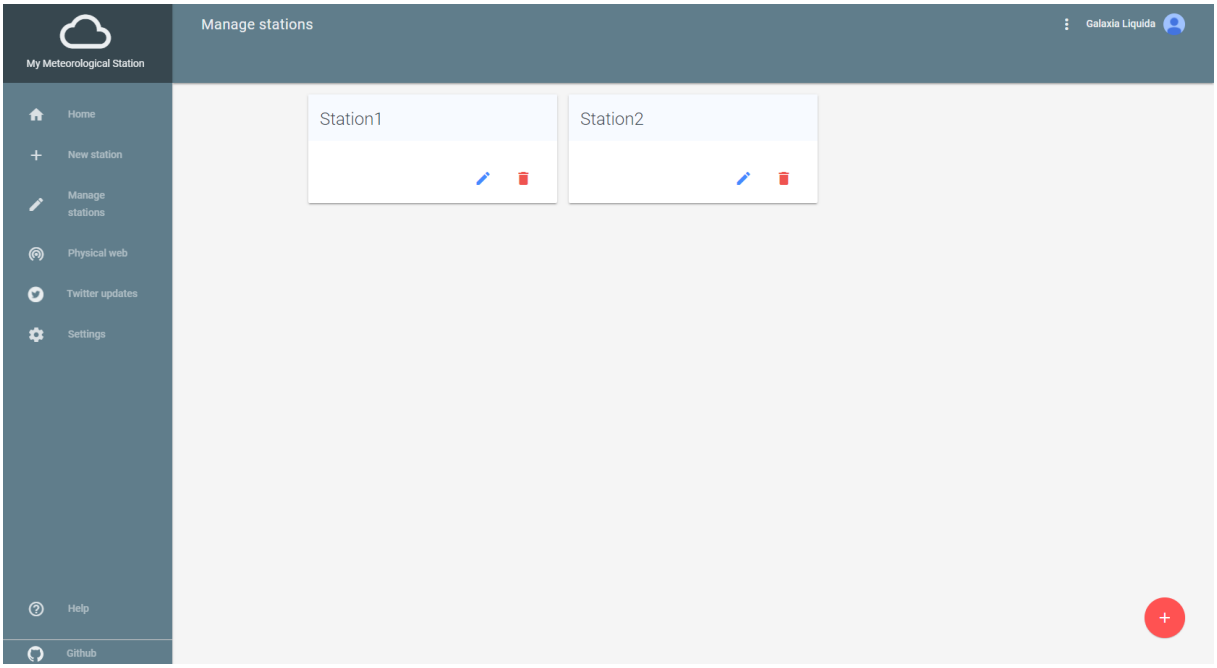

Figura 52: Apartat d'administració de les estacions

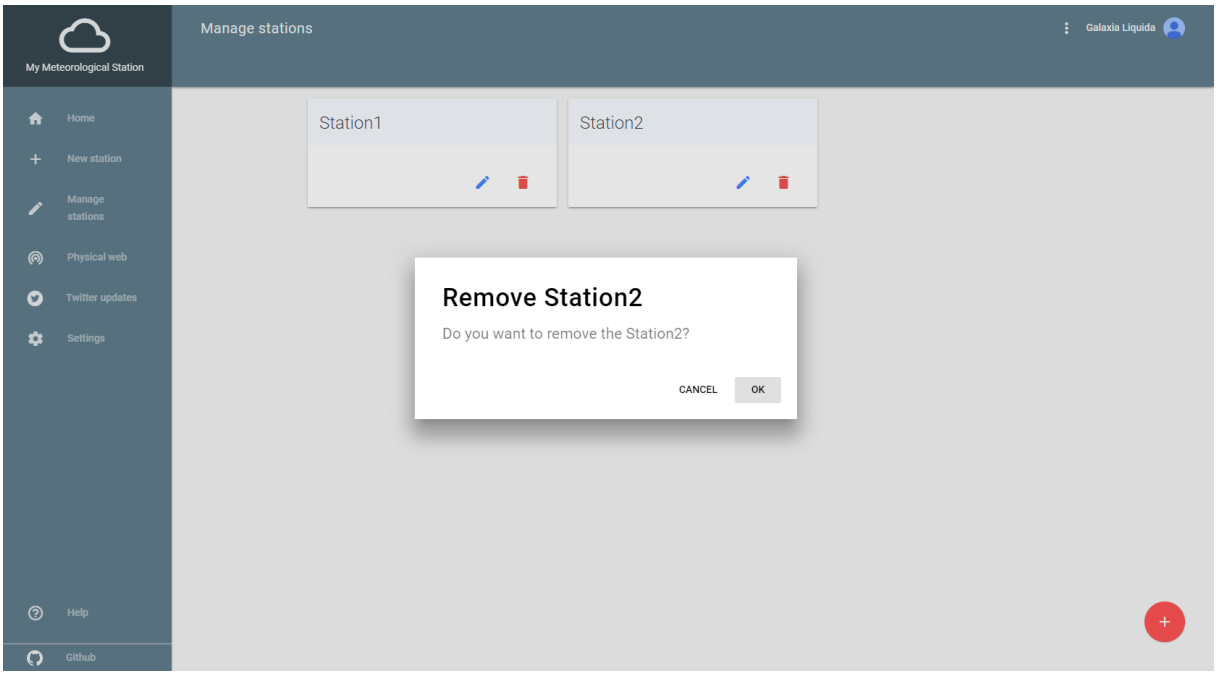

Figura 53: Diàleg de confirmació per eliminar una estació

5.5.2 Còpia de la clau Maps API

Per tal de mostrar l'opció del mapa, s'utilitza la Google Maps API. Un requisit perquè el servei funcioni correctament és afegir al projecte l'API key.

Per això crearem un fitxer anomenat **maps-key.json** a l'arrel del projecte amb el següent contingut que emplenarem amb la nostra clau:

```
1 {
2 "apikey": ""
3 }
```

Listing 32: Contingut del fitxer maps-key.json

5.5.3 Configuració estació - servidor

En el moment que creem l'estació, s'afegirà a la llista d'estacions de l'usuari a la base de dades i es disposarà a configurar automaticament la RPi utilitzant SSH/SCP. Els passos que durà a terme són:

- **Generar i copiar els fitxers de servei:**

  Com ja hem comentat, la RPi executa un servei "systemd"a l'arrencada del sistema. Generarà i copiara al home de l'usuari un fitxer ".service"pels sensors i un altre per la API.

  Per això, utilitzarem una plantilla per generar el servei amb les variables que ha introduït l'usuari en crear l'estació.

  La plantilla tindrà el següent contingut:

```
[Unit]
Description=Start weather sensors
After=network.target multi-user.target
Wants=network-online.target

[Service]
ExecStart=/usr/bin/python /home/<%=user%>/read_sensors.py <%=name%> <%=serverip
    %> <%=uid%>
Restart=on-failure

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

Listing 33: Contingut de la plantilla del servei dels sensors

Indica que en arrancar el sistema quan la xarxa estigui disponible executi el nostre script, i el torni a arrencar en cas de fallada. Les variables com el nom de l'estació, la ip del servidor o el id de l'usuari s'introduiran en el fitxer generat en aquest pas.

Els fitxers de servei es mouran al directori "/lib/systemd/system/"que emmagatzema tots els serveis del sistema.

- **Copia dels scripts:**

  La RPi executa dos scripts, en llenguatge Python, que reben les dades de la font i realitzen el POST amb les dades al servidor. Es copiaran aquests scripts al directori home de l'usuari, perquè els serveis els puguin executar.

- **Instal·lació de les dependències:**

  S'instal·laran els paquets necessaris:

```
sudo apt-get update
sudo apt-get -y install git build-essential python-dev
pip install -r requirements.txt
```

  S'instal·laran també les llibreries dels sensors:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
cd Adafruit_Python_DHT
sudo python setup.py install

git clone https://github.com/adafruit/Adafruit_Python_BMP.git
cd Adafruit_Python_BMP
sudo python setup.py install
```

- **Activació dels serveis:**

  Ara ja podem activar el servei perquè s'activi en iniciar el sistema.

```
sudo systemctl enable weathersensors.service
sudo systemctl enable weatherapi.service
```

  I posar-lo en marxa.

```
sudo systemctl start weathersensors.service
sudo systemctl start weatherapi.service
```

Un cop el servidor hagi acabat de configurar la RPi, ja començarà a enviar peticions POST amb les dades que rebrà el servidor.

```
router.post('/sensors', function(req, res){
    firebase.writeStationSensors(req.body);
    res.json(req.body);
});

router.post('/api', function(req, res){
    firebase.writeStationAPI(req.body);
    res.json(req.body);
});
```

Listing 34: Mètodes POST del servidor per rebre les dades

El servidor comprova que la id del usuari sigui propietari de l'estació i en cas afirmatiu crearà l'estructura a la base de dades i hi guardarà els valors.

```
function writeStationSensors (data) {
    isValidStation(data.uid, data.name).then(valid => {
        if (valid) {
            admin.database().ref('stations/' + data.uid + '/' + data.name).update({
                name: data.name,
                humidity: utils.round(data.humidity),
                ...
            });
        }
    });
}

function isValidStation (uid, name) {
    return new Promise(function (resolve, reject) {
        admin.database().ref('/users/' + uid).once('value', function (snapshot) {
            for (const station in snapshot.val()) {
                if (name === snapshot.child(station).val()) {
                    return resolve(true);
                }
            }
            return resolve(false);
        });
    });
}
```

Listing 35: Funcions per guardar els valors dels sensors i comprovar que la estació sigui vàlida

En editar una estació, realitzarà el mateix procés excepte la instal·lació de dependències, que no serà necessària tornar a repetir.

En eliminar una estació s'esborren totes les dades de l'estació. També s'elimina l'estació de la llista d'estacions de l'usuari, permetent que en cas que l'estació continués activa enviant POSTS al servidor, aquestes peticions serien ignorades, ja que no passaria la comprovació de propietari de l'estació.

Totes aquestes funcions requereixen que l'usuari estigui autentificat per crear, editar o eliminar una estació i el servidor corri en mode de desenvolupament tenint l'estació en la mateixa xarxa per l'autoconfiguració en SSH (tant en la creació com en l'edició).

5.6 Balises

A continuació es detallarà com l'usuari interacciona amb la interfície del sistema i com funcionen internament les seves parts.

5.6.1 Interfície d'usuari

El sistema permet gestionar les balises podent seleccionar quina estació ha d'emetre utilitzant la redirecció d'URL.

Per crear una balisa entrarem a l'apartat *Physical Web* en la secció *New beacon* (figura 54).

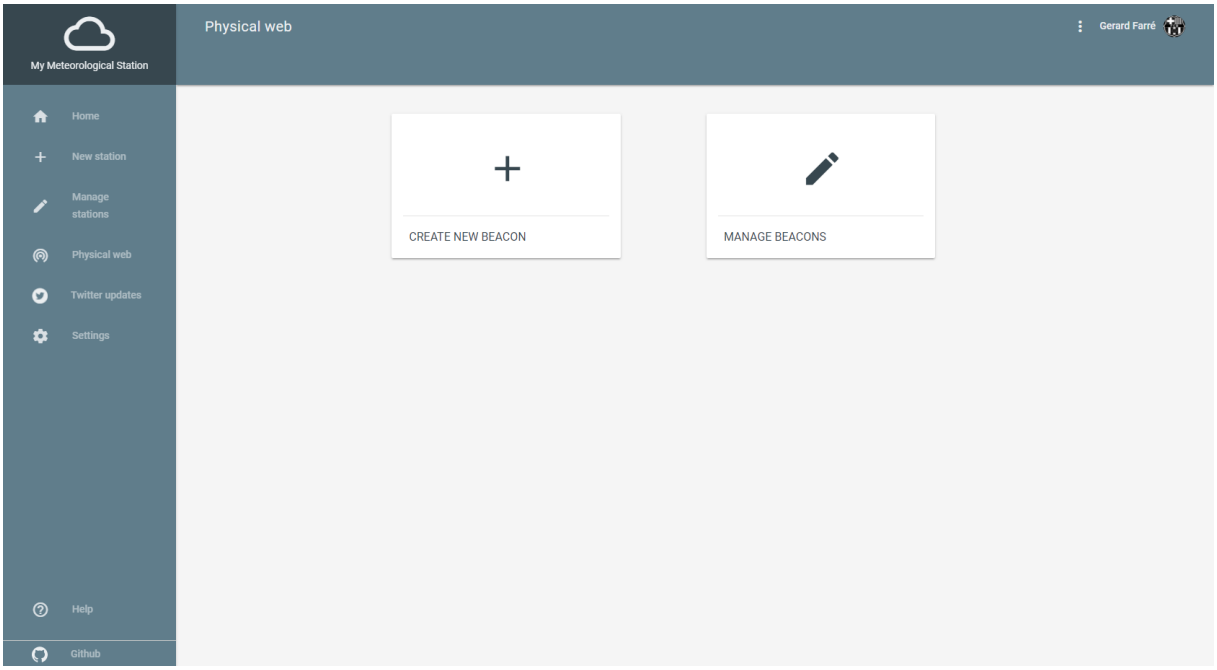

Figura 54: Secció Physical Web

Posarem el nom de la balisa i seleccionarem l'estació a emetre (posteriorment modificable), tal com es mostra a la figura 55.

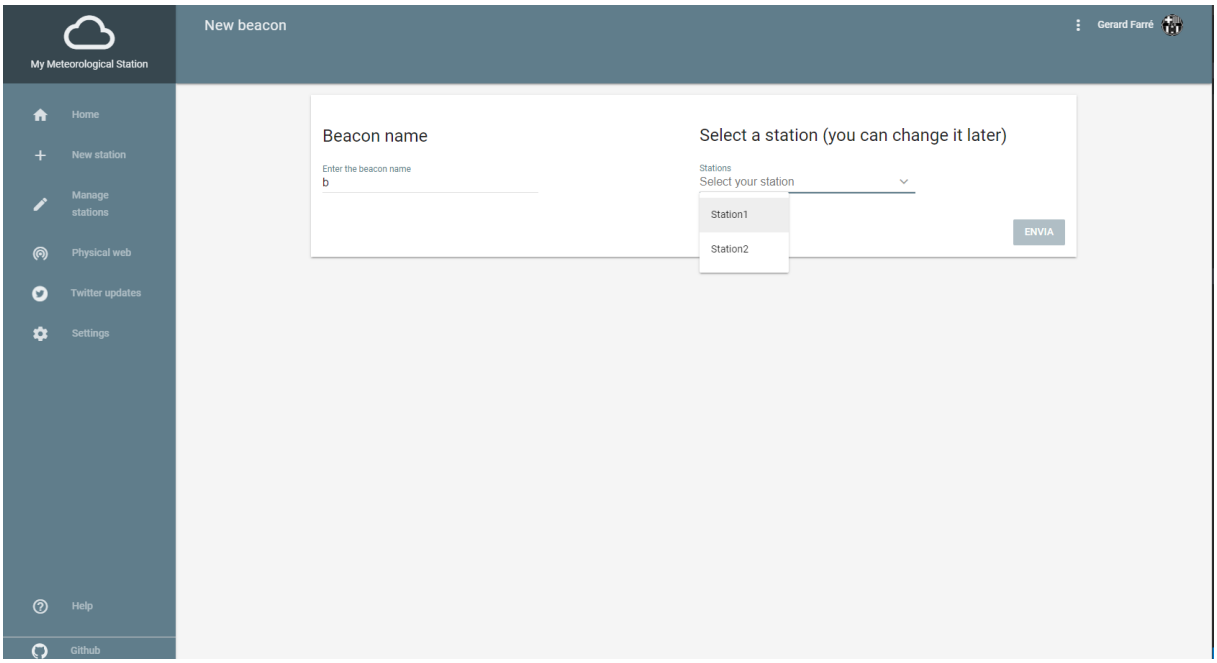

Figura 55: Formulari de creació d'una balisa

Un cop creada, generarà la URL que identifica la balisa i que s'escurçarà utilitzant el reductor de URL is.gd.

Aquesta URL s'haurà de gravar a la balisa utilitzant la utilitat web oficial de la marca, tal com mostra el procediment de les figures 56, 57 i 58.

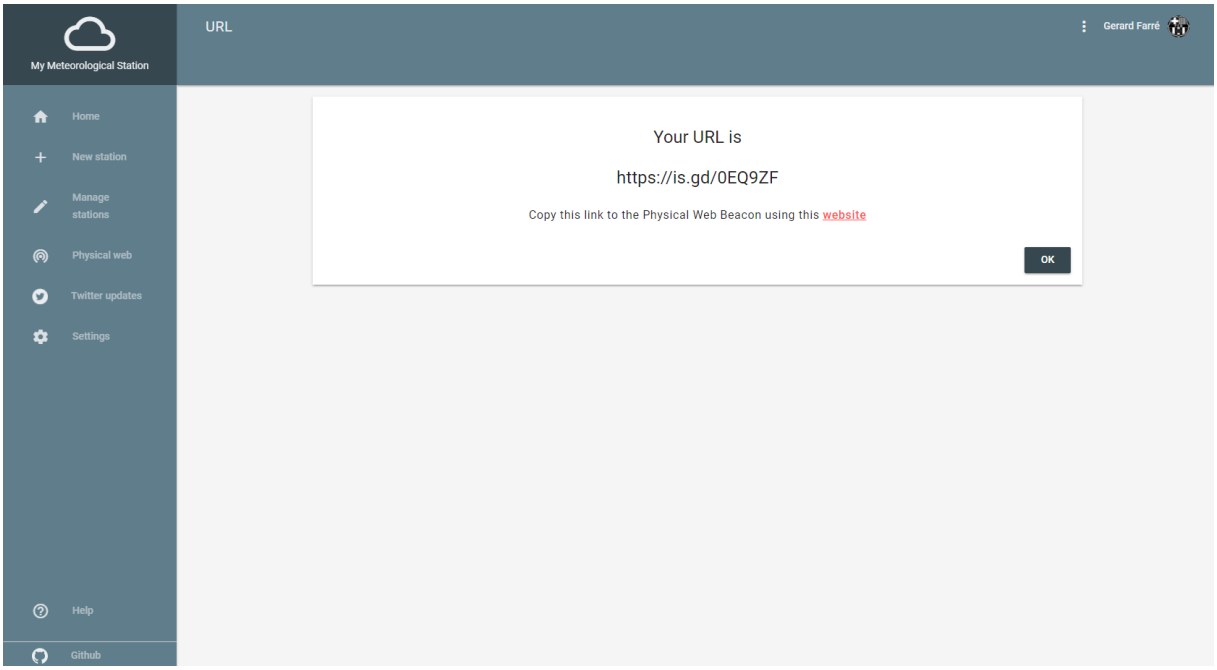

Figura 56: Pantalla que retorna la URL un cop creada la balisa

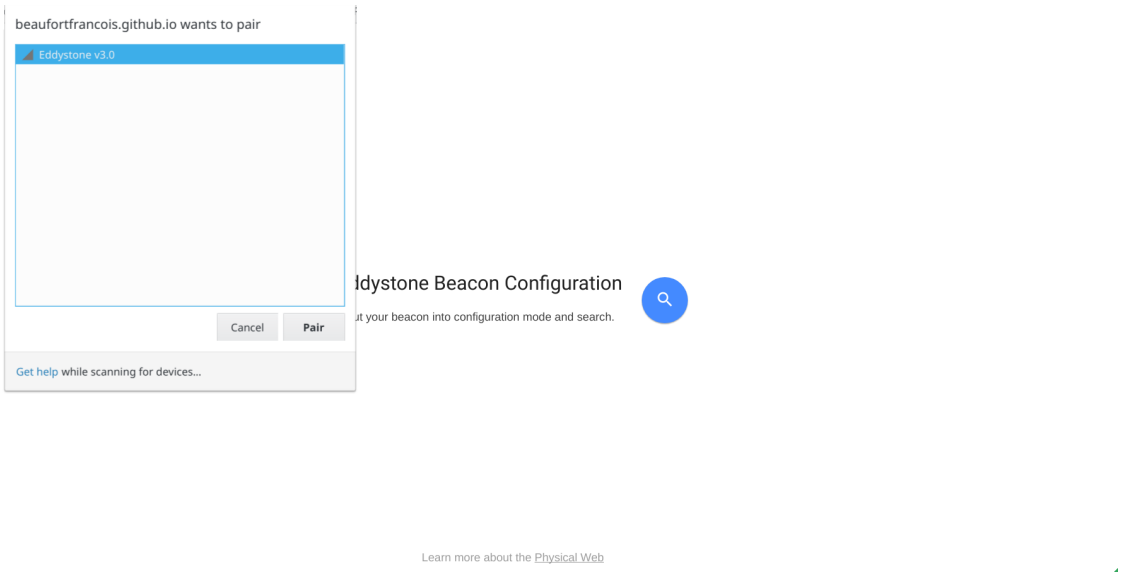

Figura 57: Utilitat per emparellar la balisa mitjançant bluetooth

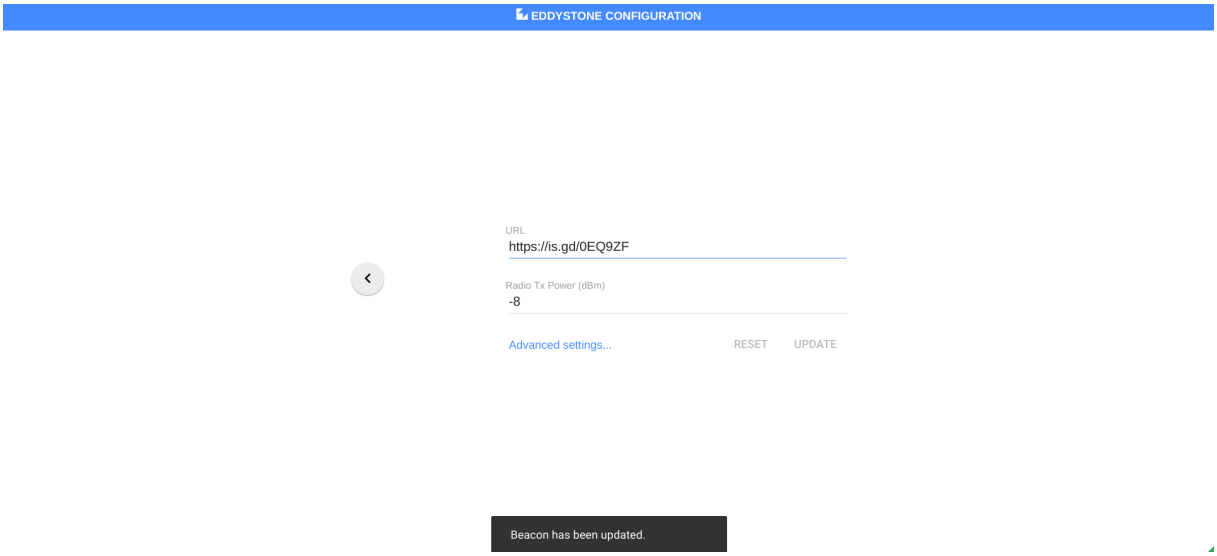

Figura 58: URL introduïda i balisa actualitzada correctament

En la figura 59 es mostra la secció d'administració de la balisa, on es pot observar la URL assignada, l'estació que emet, editar els seus camps o eliminar-la.

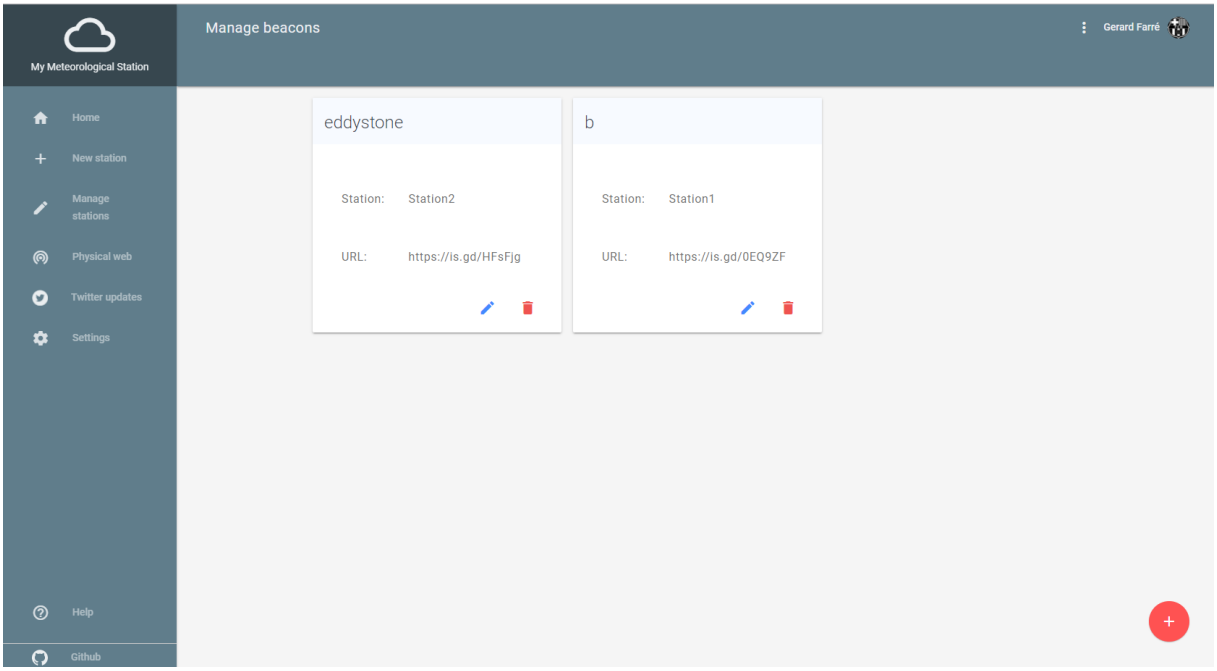

Figura 59: Secció d'administració de les balises

5.6.2 Escurçador de URL

A causa de la limitació de caràcters de l'URL que guarda una balisa, s'ha d'utilitzar un servei per escurçar-la.

Per tal d'escurçar l'URL generada, s'utilitzarà el servei gratuït is.gd i el mòdul npm isgd, usant el codi mostrat a continuació.

```
1 const UrlShorter = require('isgd');
2
3 UrlShorter.shorten(longUrl, url => {
4   id.update({url: url});
5 });
```

Listing 36: Funció per escurçar la URL

5.6.3 Redirecció de URL

El sistema que utilitza la redirecció és el següent:

Posem el cas que tenim una balisa amb id "balisa1"que emet la "estacio1". La URL escurçada apuntarà a "/pw/balisa1", que és la direcció que obrirà cada cop que se seleccioni la notificació.

La URL anterior rebrà de la base de dades l'estació que ha d'emetre, en aquest cas la 'estacio1' i redirigirà el navegador directament a la seva pàgina, "/#estacio1".

```
1 router.get('/pw/:id', function (req, res){
2     firebase.getBeacon(req.params.id).then(data => {
3         res.render('pwcontent', {
4             title: 'My meteorological station',
```

```
            description: 'There is a nearby station! Touch to open.',
            url: 'https://' + url + '/#' + data.station
        });
    });
});
```

Listing 37: Mètode que gestiona la notificació de la balisa

Així, en el moment que s'edita la balisa, i es canvia d'estació a emetre a la "estacio2", en tornar a seleccionar la notificació rebrà la nova balisa a emetre i redirigirà a "/#estacio2".

```
html
    head
        title #{title}
        meta(name='description', content=description)
        link(rel='icon' type='image/png' href='/img/cloud.png')
        script.
            window.location.href = '#{url}';
```

Listing 38: Vista de la notificació Physical Web

5.7 Liquid Galaxy

En aquesta secció descriurem com es realitza la comunicació amb el Liquid Galaxy i la generació dels KML.

5.7.1 Comunicació

La comunicació amb el Liquid Galaxy es realitzarà per SSH, per tant només es podrà fer ús d'aquesta funció tenint el servidor corrent en la mateixa xarxa local que el Liquid Galaxy.

Per moure't a una posició, podem realitzar consultes al fitxer **query.txt** que usarem amb les coordenades de l'estació.

```
function flyTo (lgip, lgpass, latitude, longitude) {
    const message = "echo 'search="+latitude+","+longitude+"' > /tmp/query.txt";
    communicate(lgip, lgpass, message);
}

function communicate (lgip, lgpass, message) {
    const command = "sshpass -p '"+lgpass+"' ssh lg@"+lgip+" \""+message+"\"";
    execute_command(command);
}
```

Listing 39: Funció per desplaçar-te a una localització al Liquid Galaxy

També podem enviar KML utilitzant el fitxer **kmls.txt** que està escoltant el Liquid Galaxy. Aquest fitxer pot contenir el contingut d'un fitxer KML, o la URL d'un o varis KML. Utilitzant qualsevol de les dues maneres el LG s'encarregarà de mostrar-los a les seves pantalles, encara que en aquest projecte s'utilitzarà el sistema de la URL.

El servidor servirà en estàtic el fitxer KML, de tal manera que s'escriurà la URL a la que apunta a un fitxer que generarà el servidor anomenat kmls.txt.

```
function send_single_kml (lgip, lgpass, name, route, tour) {
    const content = 'http://' + get_server_ip() + ':3000/kml/' + name + '\n';
    const command = "echo '" + content + "' > "+route+"/kmls.txt";
    child = exec( command, function (error, stdout, stderr) {
        if (tour) {
            send_galaxy_tour(lgip, lgpass, route);
        } else {
            send_galaxy(lgip, lgpass, route);
        }
    });
}
```

Listing 40: Escriu la URL del fitxer KML a kmls.txt

Quan ja tenim el fitxer kmls.txt generat, l'enviarem al LG.

```
function send_galaxy (lgip, lgpass, route) {
    const file_path = route+'/kmls.txt';
    const server_path = '/var/www/html/kmls.txt';
    const command = "sshpass -p '" + lgpass + "' scp " + file_path + " lg@" + lgip +
    ":" + server_path;
    execute_command(command);
}
```

Listing 41: Envia kmls.txt al Liquid Galaxy

Per netejar les dades del Liquid Galaxy només s'haurà de netejar tot el contingut del fitxer kmls.txt, deixant-lo buit.

5.7.2 Emmagatzemament de la configuració del Liquid Galaxy

Per poder establir comunicació mitjançant SSH amb el Liquid Galaxy necessitarem conèixer la seva ip i la contrasenya de l'usuari. En la secció *Settings*, tal com es pot veure a la figura 60, hi haurà dues caixes d'introducció de text per a guardar aquests camps.

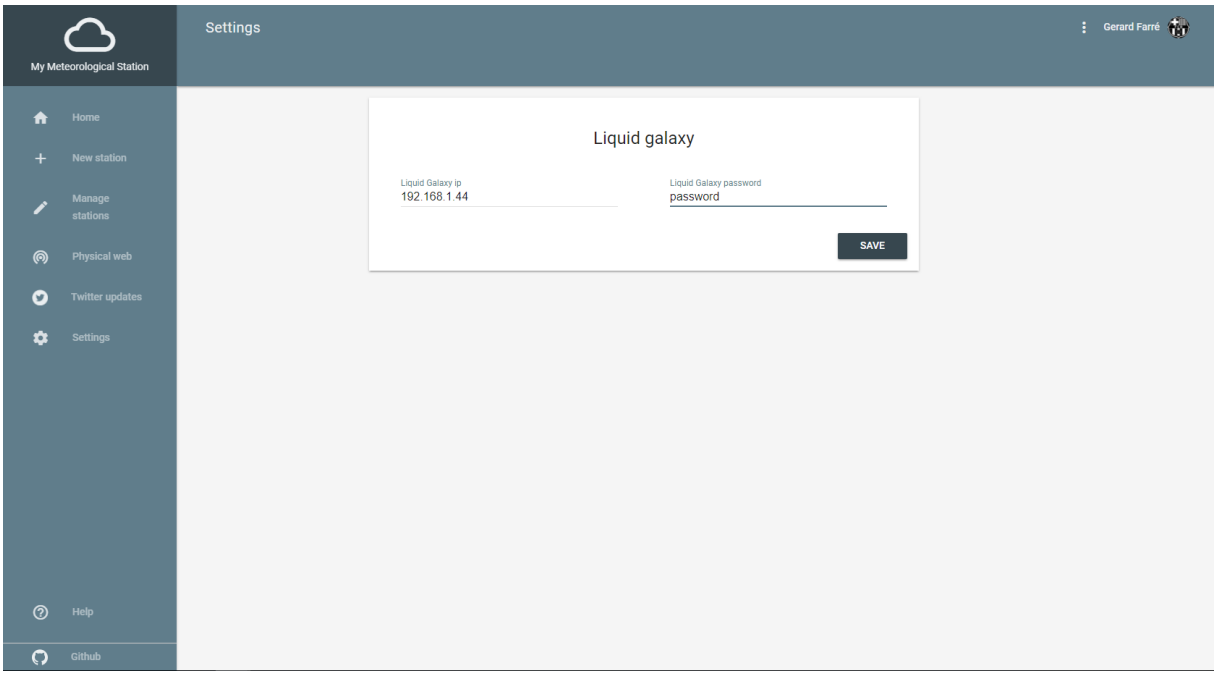

Figura 60: Secció de configuració dels valors del Liquid Galaxy

Aquests valors es guardaran en un fitxer de text local al mateix servidor utilitzant l'eina node persist, ja que s'ha considerat que són unes dades poc variables i pertanyents només al servidor (ni de l'usuari, ni de les estacions). Aquestes dades es guarden de la següent manera:

```
storage.init().then(function() {
  storage.setItem('lgsettings',{
    ip: req.body.ip,
    pass: req.body.pass
  });
});
```

Listing 42: Emmagatzemament de les dades usant node persist

I les podem recuperar utilitzant *getItem*:

```
storage.init().then(function() {
  storage.getItem('lgsettings').then(lgsettings => {
    // lgsettings.ip
    // lgsettings.pass
  })
});
```

Listing 43: Recuperació de dades emmagatzemades localment usant node persist

5.7.3 Generació de KML

Per tal de generar els fitxers KML utilitzarem el mateix sistema que els serveis, les plantilles.

Aquesta plantilla tindrà com a variables, la descripció del globus, que contindrà tots els valors de l'estació i les coordenades on se situa el globus.

La plantilla més bàsica, que mostra un globus en un punt, és la següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext
    /2.2">
    <Document id="feat_1">
        <Style id="stylesel_0">
            <BalloonStyle>
                <bgColor>ffe6d8ad</bgColor>
                <displayMode>default</displayMode>
                <text>$[description]</text>
            </BalloonStyle>
        </Style>
        <Placemark id="feat_2">
            <description><%=description%></description>
            <styleUrl>#stylesel_0</styleUrl>
            <gx:balloonVisibility>1</gx:balloonVisibility>
            <Point id="geom_0">
                <coordinates><%=coordinates%>,0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

Listing 44: Plantilla que mostra un globus en un punt

Les variables *description* i *coordinates* s'emplenaran en la generació del fitxer, agafant les dades de la base de dades, que conté les últimes dades disponibles.

```
firebase.readStationData(req.body.name).then(data => {

  lg.show_kml_balloon(lgsettings.ip, lgsettings.pass, data);

});
```

Listing 45: Recull les dades de Firebase i crida la funció per generar el KML

```
function show_kml_balloon (lgip, lgpass, data) {
    const contentString = content.getContent(data);

    const values = {
        description: contentString,
        coordinates: data.longitude+','+data.latitude,
    };

    renderFile(data.city, values, template).then(data => {
        send_single_kml(lgip, lgpass, data[0], data[1], tour);
    });
}
```

Listing 46: Renderitza la plantilla KML amb les dades rebudes

Per realitzar les diferents funcions del Liquid Galaxy només canviarà la plantilla a generar en cada cas.

5.8 Google Assistant

En primer lloc es crearà un agent a la plataforma API.AI.

Tal com s'observa a la figura 61 es crearà l'entitat "weather-values"que contindrà tots els valors els quals pots preguntar.

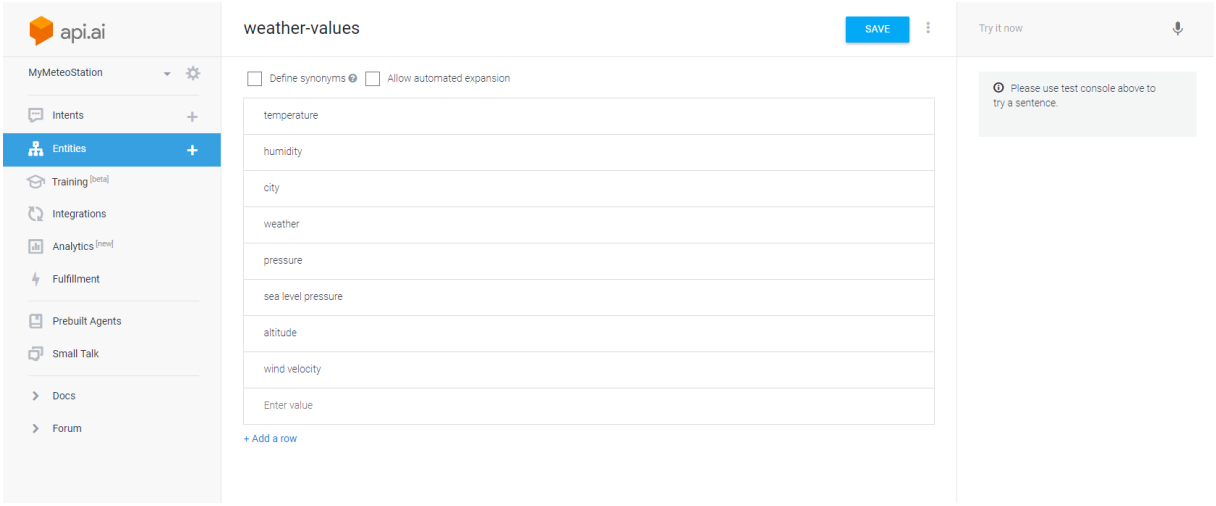

Figura 61: Entitat weather-values a l'API.AI

Ara ja es poden generar les preguntes que pot realitzar l'usuari. Ens disposarem a detallar, la pregunta que retorna el valor d'una estació. *What's the* ***{value}*** *of the* ***{station name}****?*

Tal com es veu a la figura 62, la pregunta es pot formular de diferents formes. Marcarem la paraula *temperature* com paràmetre amb entitat *weather-values* i obligatòria. El número d'estació requerirà ser un nombre enter.

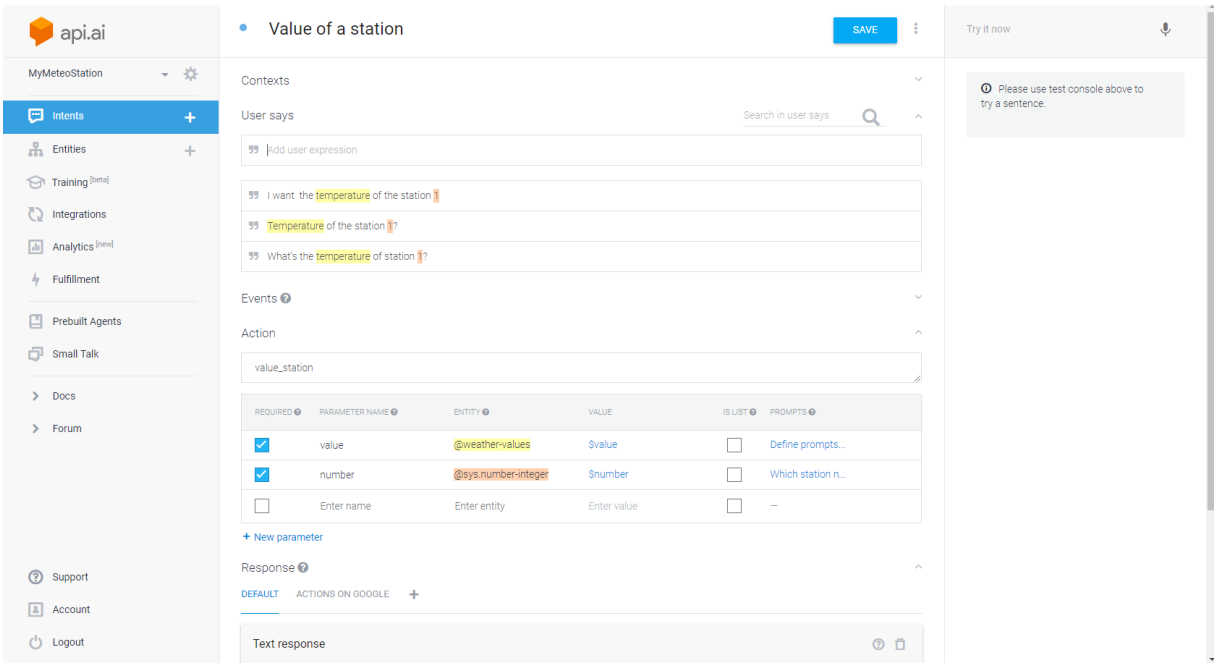

Figura 62: Pagina de detall de l'intent a l'API.AI

En l'apartat *fulfillment* serà obligatori afegir el webhook, que gestionarà el nostre servidor a la ruta "/assistant".

```
router.post('/assistant', function(req, res){
    assistant.webhook(req, res);
});
```

Listing 47: Mètode POST que gestiona el webhook

Un cop la crida arribi a la funció pertinent, es rebran els arguments, es buscaran a la base de dades Firebase i es construirà la resposta amb les dades de retorn. Quan es tingui la resposta completa, es retornarà a l'usuari cridant a la funció app.ask.

```
function valueStationIntent (app) {
  const value = app.getArgument(VALUE_ARGUMENT);
  const number = app.getArgument(NUMBER_ARGUMENT);

  const station = 'Station'+ number;
  let val = '';
  if (value !== null) val = assistant_utils.getKeyAndUnit(value);

  firebase.readStationData(station).then(data => {
    let answer = '<speak>';
    if (val.valueId in data) {
      answer += 'The ' + value + ' is ' + data[val.valueId] + val.measureUnit +
                '. Do you want anything more?';
    } else {
      answer += 'The ' + station + ' doesn\'t have the '+ value + ' value. Try again!
    ';
    }
    answer += '</speak>';
    app.ask(answer);
  });
}
```

Listing 48: Funció que retorna el valor d'una estació a Google Assistant

D'aquesta manera, podem generar un intent preguntant per tots els valors d'una estació, utilitzant una sola variable, la de número d'estació, i la funció del webhook pertinent que construeixi la resposta amb tots els valors consultats de la base de dades. O tots els valors de totes les estacions, que no implica cap variable, ja que simplement es crea la resposta amb totes les dades existents.

També es permet controlar el Liquid Galaxy mitjançant Google Assistant.

Per mostrar un globus de l'estació, rebrem com a paràmetre l'estació, que cridarà les funcions que s'han mencionat en l'apartat anterior. Així doncs, la seva funció en el *webhook* contindrà el següent codi:

```
function stationBalloon (app) {

  const number = app.getArgument(NUMBER_ARGUMENT);
  const station = 'Station'+ number;
  let answer = '<speak>';

  storage.init().then(function() {
    storage.getItem('lgsettings').then(function(lgsettings) {
```

```
9       firebase.readStationData(station).then(data => {
10
11        lg.flyTo(lgsettings.ip, lgsettings.pass, data.latitude, data.longitude);
12        lg.show_kml_balloon(lgsettings.ip, lgsettings.pass, data, false);
13
14      });
15      answer += 'Let\'s go, showing information about '+ station +
16              ' to liquid galaxy. <break time="2" /> ';
17      answer += 'Do you want anything more?</speak>';
18      app.ask(answer);
19    })
20  });
21 }
```

Listing 49: Funció per Google Assistant que mostra un globus al Liquid Galaxy

Podem observar que l'única diferencia entre mostrar informació al LG des del botó o des de l'Assistant és en la recepció de l'argument i en la generació de la resposta que es retorna a l'usuari.

5.9 Twitter

Per tal que el servidor es pugui comunicar amb l'aplicació Twitter necessitem la configuració de l'aplicació.

Crearem el fitxer **twitter-config.json** a l'arrel del projecte amb el següent contingut:

```
1 {
2   "consumer_key": "",
3   "consumer_secret": "",
4   "access_token": "",
5   "access_token_secret": ""
6 }
```

Listing 50: Contingut del fitxer twitter-config.json

L'haurem d'emplenar agafant els valors de l'administració de l'aplicació en la pestanya *Keys and Access Tokens* com es mostra a la figura 63.

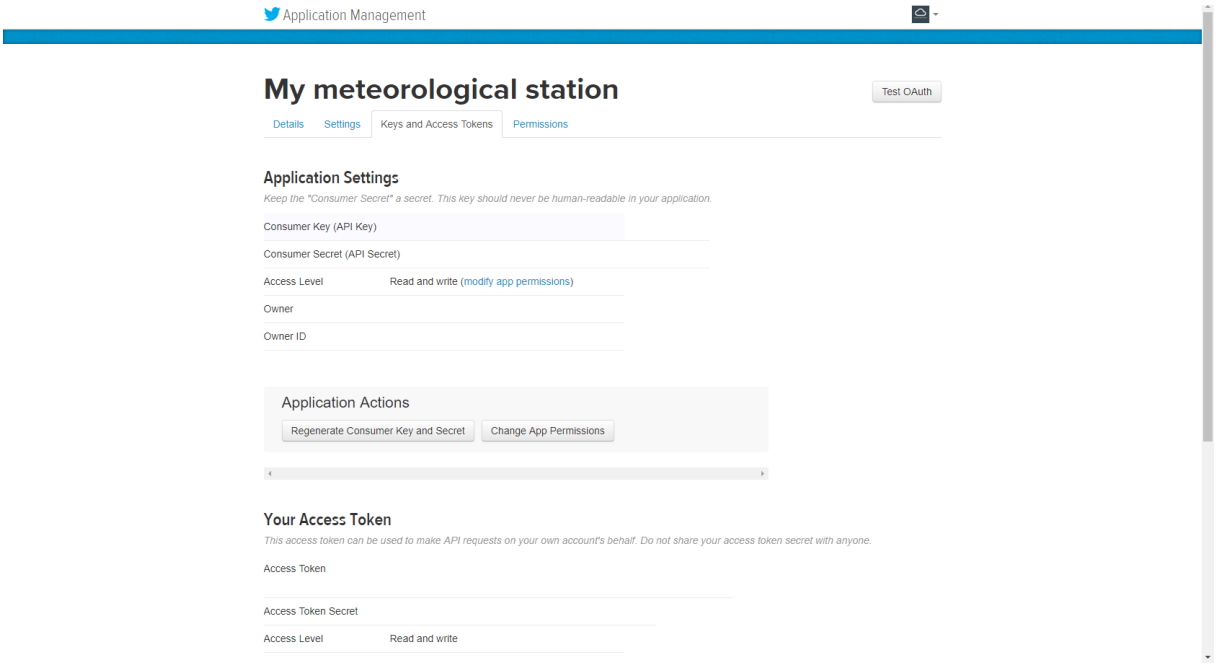

Figura 63: Pagina d'administració de claus de l'aplicació de Twitter

Ara el servidor ja pot publicar tuits en el compte de Twitter carregant la configuració i inicialitzant-la.

```
const twitterConfig = require('../twitter-config.json');
const twitter = new Twit(twitterConfig);
```

Listing 51: Inicialització de l'aplicació de Twitter

Per tal de generar el tuit, es rebrà totes les dades de totes les estacions, i per cada estació construirà el missatge i el publicarà utilitzant la funció *post*.

```
function tweetUpdate () {

    firebase.readStations().then(data => {

        for (const station in data) {
            const msg = data[station].name + ': ' + data[station].alias +' \n' +
                'City: ' + data[station].city + '\n' +
                'Weather: ' + data[station].weather + '\n' +
                'Temperature: ' + data[station].temperature + ' C\n' +
                'Humidity: ' + data[station].humidity + ' %\n' +
                'Pressure: ' + data[station].pressure + ' Pa\n';

            twitter.post('statuses/update', {status: msg},
              function (err, data, response) { console.log(data); }
            );
        }

    });

}
```

Listing 52: Funció que realitza la publicació de les dades a Twitter

Per tal que aquesta funció s'executi cada 24 hores utilitzarem la funció *setInterval*[24] subministrada per Node.js.

```
1 setInterval(twitter.tweetUpdate, 86400000);
```

Listing 53: Executa l'actualització a Twitter cada 24 hores

[24]És un mètode intern de node que crida a una funció cada cert període de temps.

6 Conclusions i línies obertes

En aquest capítol final s'explicaran les conclusions personals que he extret de la realització d'aquest projecte i les línies que queden obertes per tal que el projecte segueixi creixent i millorant en un futur.

6.1 Conclusions

En primer lloc he pogut aprofundir en el món de la Raspberry i el seu port GPIO, que té un gran potencial donada la gran varietat de sensors que té disponible. La selecció i connexió dels sensors (amb alguna petita soldadura pel mig), ha sigut una bona experiència.

El principal atractiu de la realització del projecte ha sigut integrar un conjunt de tecnologies diferents entre si en un mateix sistema. Aquestes tecnologies s'han utilitzat per fer arribar de diferents formes les dades a l'usuari, tant de manera visual com auditiva, donant-li un valor afegit al projecte.

He après tecnologies que no coneixia com la comunicació amb el Liquid Galaxy, la representació de dades a Google Earth o el funcionament de les balises bluetooth. M'ha permès aprofundir i adquirir coneixements en una tecnologia de futur i molt interessant com Google Assistant. Degut el gran creixement que estan tenint en els últims temps els assistents intel·ligents (Assistant, Siri, Cortana, Alexa, ...), he trobat una gran oportunitat haver treballat en aquesta plataforma.

Per realitzar l'aplicació he utilitzat Node.js, un framework que vam començar a donar l'últim curs del grau i que em va cridar l'atenció, així doncs ha sigut l'escollit per tal d'aprofundir-hi. Passa una cosa similar amb la plataforma Firebase (també la vàrem utilitzar l'últim curs del grau) que ofereix una gran simplificació a l'hora de construir una aplicació web. L'administració de la base de dades i el control d'usuaris és un gran al·licient i resol diferents problemes que et trobes durant la realització del projecte.

Hi ha diferents parts del projecte que es podrien haver dissenyat o implementat d'una manera més eficient o elegant, però la falta de temps i la proximitat de l'entrega del projecte m'ha obligat a centrar-me en part més importants o amb una funcionalitat més visible. De totes maneres estic molt content amb el resultat final del projecte i crec que he complert la major part dels objectius que m'havia proposat.

6.2 Línies obertes

A continuació es nomenaran una llista d'idees d'ampliació del projecte per un futur, sense contar la correcció dels petits errors que poden aparèixer.

- **Multi idioma:** Actualment només està en anglès, es podria afegir diferents idiomes i que automàticament les vistes mostressin l'idioma del lloc de procedència de l'usuari.
- **Diferents unitats de mesura per les dades de l'estació:** El sistema només tracta les dades en graus centígrads, metres i pascals. Es tractaria que l'usuari pogués escollir en la configuració diferent unitats de mesura per representar la informació.

- **Estadístiques:** Es podria afegir diferents estadístiques que siguin útils a l'usuari, com temperatures màximes i mínimes, gràfiques dels valors segons el dia/setmana/mes anterior, entre d'altres. Això implicaria anar acumulant dades a la base de dades en lloc de reemplaçar-les com es realitza actualment, incrementant l'espai necessari en la base de dades.
- **Més sensors a l'estació:** La idea seria anar afegint cada cop més sensors a l'estació fins al punt de reemplaçar totes les dades que es reben de l'API i així no dependre de cap servei extern, tenint totes les dades reals d'aquell punt concret on es troba l'estació.
- **Feedback a l'usuari en la configuració de la RPi:** Mentre s'està configurant una estació, l'usuari no sap que està passant per darrere ni si el procés ha acabat correctament. Seria necessari anar informant l'usuari de què està succeint a la RPi en tot moment.
- **Desactivar temporalment estacions fora de línia:** Si una estació fa uns dies que està desconnectada, es mostra igual amb els últims valors que va guardar a la base de dades, mostrant unes dades que no són reals actualment. S'hauria de detectar si una estació no envia dades al servidor i desactivar-la fins que torni a estar disponible.
- **Millorar l'experiència d'usuari:** Com l'usuari interactua amb l'aplicació és una part complexa, ja que hi ha molts tipus d'usuaris amb molts perfils diferents. Es podria realitzar un estudi detectant i millorant els punts negres de la interfície.
- **I molt més...**

Referències

[1] Actions sdk. https://developers.google.com/actions/develop/sdk/.

[2] Api.ai. https://api.ai/.

[3] Google assistant. https://assistant.google.com/.

[4] Google assistant sdk. https://developers.google.com/assistant/sdk/.

[5] Keyhole markup language. https://developers.google.com/kml/.

[6] Liquid galaxy. https://github.com/LiquidGalaxy.

[7] Liquid galaxy lab. https://github.com/LiquidGalaxyLAB.

[8] Node.js documentation. https://nodejs.org/en/docs/.

[9] Physical web. https://google.github.io/physical-web/.

[10] Raspberry pi. https://www.raspberrypi.org/.

[11] Weather underground api. https://www.wunderground.com/.