



“SISTEMA PARA EL MONITOREO, DETECCIÓN Y ALERTA DE
SOMNOLENCIA DEL CONDUCTOR MEDIANTE VISIÓN ARTIFICIAL,
COMUNICACIÓN INALÁMBRICA Y GEOLOCALIZACIÓN”

Tercer Reporte Parcial

Lista de actividades

- Implementación del sistema de Geolocalización
- Despliegue de la aplicación web
- Realizar la conexión de la aplicación web con el Módulo Central de Procesamiento
- Implementación del modelo de red neuronal para la detección de somnolencia en la Jetson Nano
- Realizar la integración de los periféricos al Módulo Central de Procesamiento

Autores:

Alan Eduardo Gamboa Del
Ángel
Maite Paulette Díaz Martínez

Asesores:

M.en C. Niels Henrik Navarrete
Manzanilla
Dr. Rodolfo Vera Amaro

19 de Mayo 2023

Índice

1. Implementación del Sistema de Geolocalización	4
1.1. Objetivo	4
1.2. Descripción	4
1.3. Resultados	8
2. Despliegue de la aplicación web	10
2.1. Objetivo	10
2.2. Descripción	10
2.3. Resultados	11
3. Realizar la conexión de la aplicación web con el Módulo Central de Procesamiento	12
3.1. Objetivo	12
3.2. Descripción	12
3.3. Resultados	13
4. Implementación del modelo de red neuronal para la detección de somnolencia en la Jetson Nano	14
4.1. Objetivo	14
4.2. Descripción	14
4.3. Resultados	17
5. Realizar la integración de los periféricos al Módulo Central de Procesamiento	19
5.1. Objetivo	19
5.2. Descripción	19
5.3. Resultados	20
6. Conclusiones	21
7. Bibliografía	22

Índice de figuras

1.	Código que permite obtener la posición GPS parte 1	5
2.	Código que permite obtener la posición GPS parte 2	5
3.	Código que permite oibtener la posición GPS parte 3	6
4.	Código que permite obtener la posición GPS parte 4	6
5.	Mapa creado en AWS - MyMap	7
6.	Código para obtener la ubicación geográfica en la aplicación web	8
7.	coordenadas geográficas en la Jetson Nano	8
8.	Detalles del mapa creado en AWS	9
9.	coordenadas geográficas que recibe la página web	9
10.	Amplify Hosting	10
11.	ramificación de repositorio de Github	11
12.	Acceso a la página web desde el buscador	11
13.	Red 4G LTE activa por medio de la interfaz wwan0	13
14.	Conexión hacía Aws	13
15.	Código de somnolencia parte 1	14
16.	Código de somnolencia parte 2	15
17.	Código de somnolencia parte 3	15
18.	Código de somnolencia parte 4	16
19.	Grabación del video y el envió del video hacía la página web	16
20.	Prueba de somnolencia ojos abiertos	17
21.	Prueba de somnolencia ojos cerrados	17
22.	Prueba de la metrica MOR 1	18
23.	Prueba de la metrica MOR 2	18
24.	Carcasa sin armar	19
25.	Carcasa armada con la Jetson Nano y periféricos integrados	20

Índice de tablas

1. Implementación del Sistema de Geolocalización

1.1. Objetivo

Implementar el sistema de geolocalización en la Jetson Nano. Además de mostrar en la aplicación web la ubicación en tiempo real.

1.2. Descripción

Habilitar GPS y obtener datos de posición GPS desde la Jetson Nano

Se inicio habilitando el GPS en la Jetson Nano a través de la terminal:

- Configuración del puerto serial:

```
sudo stty -F /dev/ttyTHS1 115200
```

/dev/ttyTHS1 es el dispositivo del puerto serial utilizado y 115200 es la velocidad de transmisión.

- Habilitar el GPS: el comando echo envia comandos AT al módulo GPS y habilita su funcionalidad.

```
echo "AT+CGNSPWR=1" > /dev/ttyTHS1
```

- Verificar el estado del GPS: El comando cat, lee la salida del puerto serial y verificar el estado del GPS.

```
cat /dev/ttyTHS1
```

Al ejecutar el comando cat /dev/ttyTHS1 en la terminal, se mostrará la salida del puerto serial /dev/ttyTHS1. Si el GPS se habilitó correctamente, se observará la respuesta del módulo GPS indicando que está listo para recibir datos de posición. Esta respuesta confirmará que el GPS está funcionando correctamente y listo para proporcionar información de ubicación.

Para obtener los datos de posición GPS, fue necesario interactuar con el módulo SIM7600X a través del programa en Python:

```

GPS.py > getGpsPosition
1  #!/usr/bin/python
2
3  import Jetson.GPIO as GPIO
4  import serial
5  import time
6
7  ser = serial.Serial('/dev/ttyTHS1',115200)
8  ser.flushInput()
9
10 powerKey = 6
11 rec_buff = ''
12 rec_buff2 = ''
13 time_count = 0
14
15 def sendAt(command,back,timeout):
16     #rec_buff = ''
17     ser.write((command+'\r\n').encode())
18     time.sleep(timeout)
19     if ser.inWaiting():
20         time.sleep(0.01)
21         rec_buff = ser.read(ser.inWaiting())
22     if rec_buff != '':
23         if back not in rec_buff.decode():
24             print(command + ' ERROR')
25             print(command + ' back:\t' + rec_buff.decode())
26             return 0
27         else:
28             print(rec_buff.decode())
29             return 1
30     else:
31         print('GPS is not ready')
32         return 0

```

Figura 1: Código que permite obtener la posición GPS parte 1

```

GPS.py > ...
33
34 def getGpsPosition():
35     rec_null = True
36     answer = 0
37     print('Start GPS session...')
38     rec_buff = ''
39     sendAt('AT+CGPS=1,1','OK',1)
40     time.sleep(2)
41     while rec_null:
42         answer = sendAt('AT+CGPSINFO','+CGPSINFO: ',1)
43         if 1 == answer:
44             answer = 0
45             if '+++++' in rec_buff:
46                 print('GPS is not ready,wait 10 seconds')
47                 rec_null = False
48                 time.sleep(10)
49             else:
50                 print('error %d'%answer)
51                 rec_buff = ''
52                 sendAt('AT+CGPS=0','OK',1)
53                 return False
54         time.sleep(1.5)
55     #整理数据内容
56
57 def powerOn(powerKey):
58     print('SIM7600X is starting:')
59     GPIO.setmode(GPIO.BCM)
60     GPIO.setwarnings(False)
61     GPIO.setup(powerKey,GPIO.OUT)
62     time.sleep(0.1)
63     GPIO.output(powerKey,GPIO.HIGH)
64

```

Figura 2: Código que permite obtener la posición GPS parte 2

```

GPS.py > getGpsPosition
63     time.sleep(0.1)
64     GPIO.output(powerKey,GPIO.HIGH)
65     time.sleep(2)
66     GPIO.output(powerKey,GPIO.LOW)
67     time.sleep(20)
68     ser.flushInput()
69     print('SIM7600X is ready')
70
71     def powerDown(powerKey):
72         GPIO.setmode(GPIO.BCM)
73         GPIO.setwarnings(False)
74         GPIO.setup(powerKey,GPIO.OUT)
75         print('SIM7600X is logging off:')
76         GPIO.output(powerKey,GPIO.HIGH)
77         time.sleep(3)
78         GPIO.output(powerKey,GPIO.LOW)
79         time.sleep(18)
80         print('Good bye')
81
82     def checkStart():
83         while True:
84             ser.write( ('AT\r\n').encode() )
85             time.sleep(0.1);
86             if ser.inWaiting():
87                 time.sleep(0.01)
88                 recBuff = ser.read(ser.inWaiting())
89                 print( 'try to start\r\n' + recBuff.decode() )
90                 if 'OK' in recBuff.decode():
91                     recBuff = ''
92                     return
93             else:

```

Figura 3: Código que permite oibtener la posición GPS parte 3

```

93     else:
94         powerOn(powerKey)
95         time.sleep(1)
96     try:
97         checkStart()
98         getGpsPosition()
99         powerDown(powerKey)
100 except:
101     if ser != None:
102         ser.close()
103         powerDown(powerKey)
104         GPIO.cleanup()
105 if ser != None:
106     ser.close()
107     GPIO.cleanup()
108

```

Figura 4: Código que permite obtener la posición GPS parte 4

AWS Maps

AWS Maps es un servicio de Amazon Web Services que proporciona capacidades de mapas y geo-localización en la nube. Permite a crear, visualizar y analizar datos geoespaciales en aplicaciones y servicios.

Se creó un mapa en AWS Maps donde se configuró la apariencia y funcionalidad del mismo, incluyendo los estilos de visualización, las capas de datos y las interacciones del usuario. El propósito de este mapa es mostrar información geográfica, como ubicaciones, rutas, geovallas, puntos de interés, entre otros. La configuración del mapa permitió personalizar su aspecto y adaptarlo a las

necesidades del proyecto.

Para crear el mapa se inició sesión en la Consola de administración de AWS. Luego, se accedió a la página de AWS Maps Service en la consola de administración. A continuación, se hizo clic en "Crear mapa" para comenzar a crear el mapa. En el proceso de creación, se seleccionó un nombre para el mapa y se especificó la región de AWS donde se deseaba crearlo. Seguidamente, se eligió el estilo de mapa y posteriormente se configuraron las capas del mapa, pudiendo agregar capas para mostrar datos como carreteras, edificios, ríos, y también pudo agregar sus propios datos geoespaciales como capas personalizadas, finalmente, se creó el mapa.

Una vez que creado el mapa en AWS Maps, se agregó y configuró un rastreador para realizar el seguimiento del dispositivo llamado "mydevice" en el mapa. Los rastreadores se encargaron de recopilar y actualizar la ubicación de los dispositivos, brindando información en tiempo real.

A continuación, se presenta un resumen de cómo se creó un rastreador en AWS Maps:

En la página del mapa en la consola de administración de AWS Maps, se hizo clic en "Crear rastreador". Se seleccionó un nombre para el rastreador y se especificó la región de AWS donde se deseaba crear. Se procedió a configurar los detalles del rastreador, como la frecuencia de actualización de la ubicación de los dispositivos en este caso cada 10m envía las coordenadas. Finalmente, se hizo clic en "Crear rastreador" para finalizar la configuración del mismo.

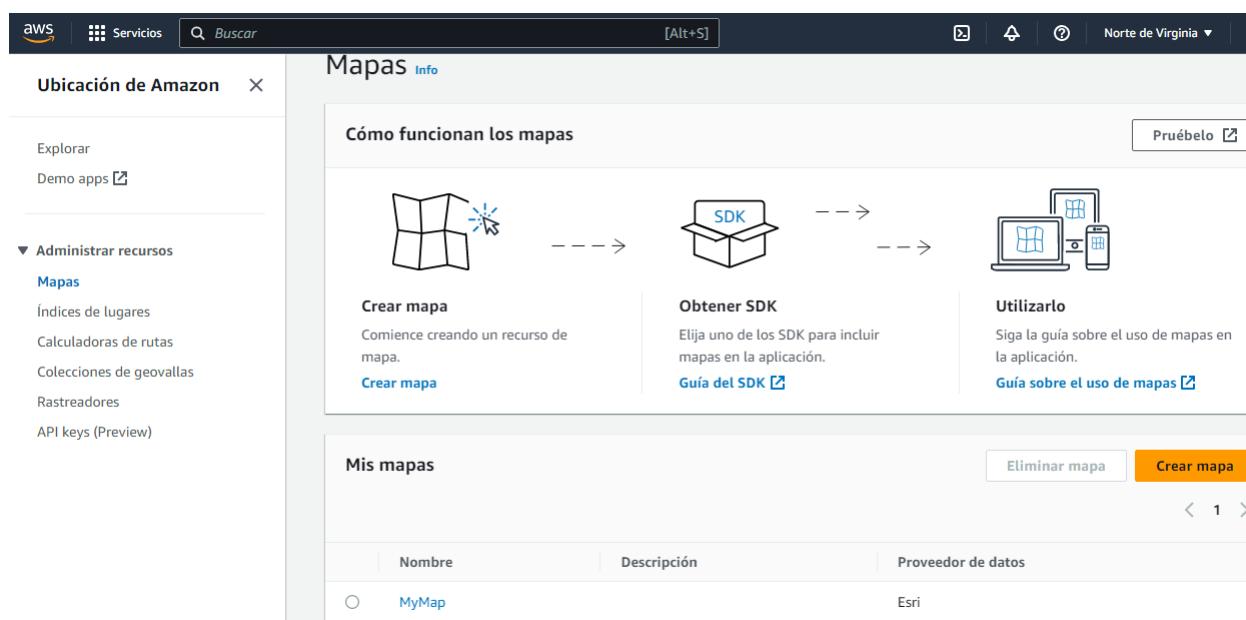


Figura 5: Mapa creado en AWS - MyMap

Ubicación geográfica en la Aplicación web

Para obtener el historial de posiciones de un dispositivo, se definió la función llamada "getPosition". Posteriormente, se utilizó la función `client.getPositionHistory` con el argumento "params" para solicitar el historial de posiciones del dispositivo. Después, se creó la variable "tempPosMarkers" para almacenar los resultados obtenidos al mapear los elementos de "data.DevicePositions". Para cada elemento "devPos" en "data.DevicePositions", se imprimió su posición en la consola y se generó un nuevo objeto con propiedades "index", "Longz", "lat", que representaban los valores de la posición del dispositivo. Este archivo se guardó con la extensión .jsx.



Figura 6: Código para obtener la ubicación geográfica en la aplicación web

1.3. Resultados

Obtención de coordenadas geográficas

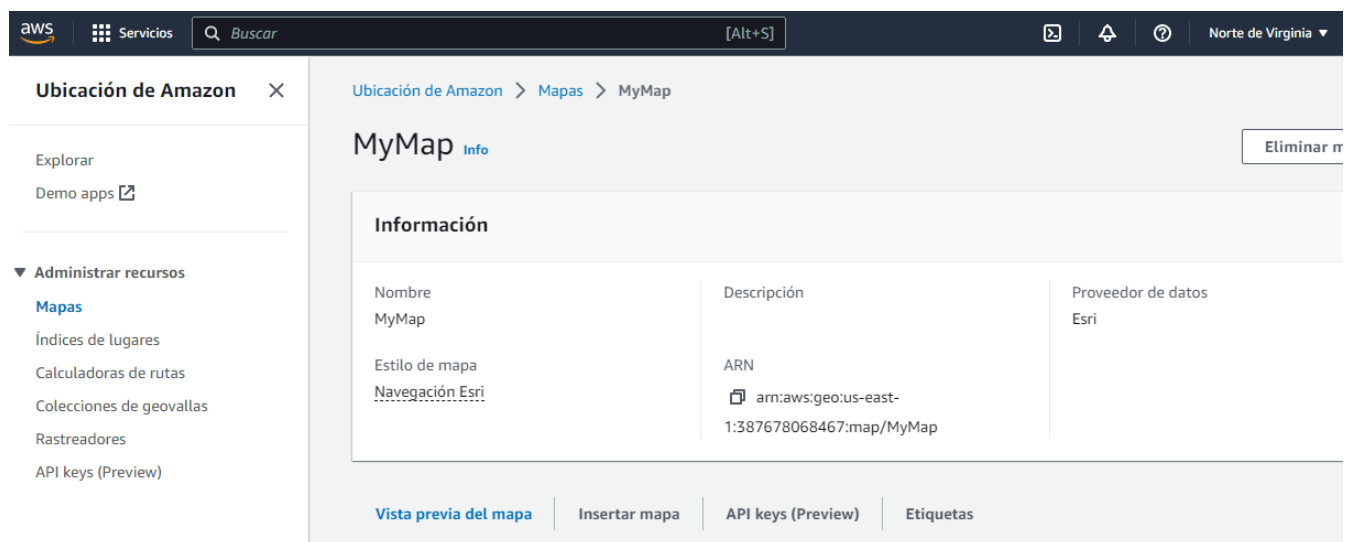


Figura 7: coordenadas geográficas en la Jetson Nano

Mapa creado en AWS Maps

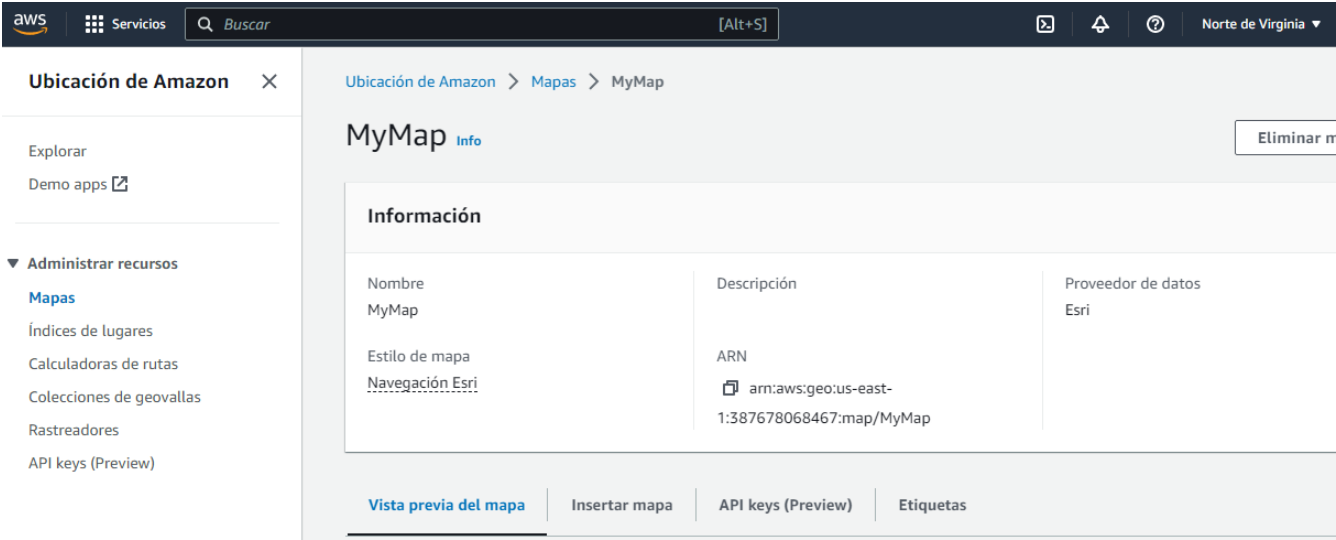


Figura 8: Detalles del mapa creado en AWS

coordenadas geográficas que recibe la página web

```

▶ {DevicePositions: Array(1)} RealTimeMap.jsx:122
-99.14154126666666 19.4703304 RealTimeMap.jsx:124
▶ {DevicePositions: Array(1)} RealTimeMap.jsx:122
-99.14154126666666 19.4703304 RealTimeMap.jsx:124
> |
```

Figura 9: coordenadas geográficas que recibe la página web

2. Despliegue de la aplicación web

2.1. Objetivo

Desplegar el frontend y backend de la aplicación en su totalidad en un servidor dedicado para que pueda ser accesible desde cualquier dispositivo con acceso a internet y con un navegador.

2.2. Descripción

Para poder alojar y desplegar la aplicación web en un servicio de *hosting*, se utilizará Amplify Hosting.

Se necesita ingresar a la consola de AWS y seleccionar el servicio de Amplify Hosting

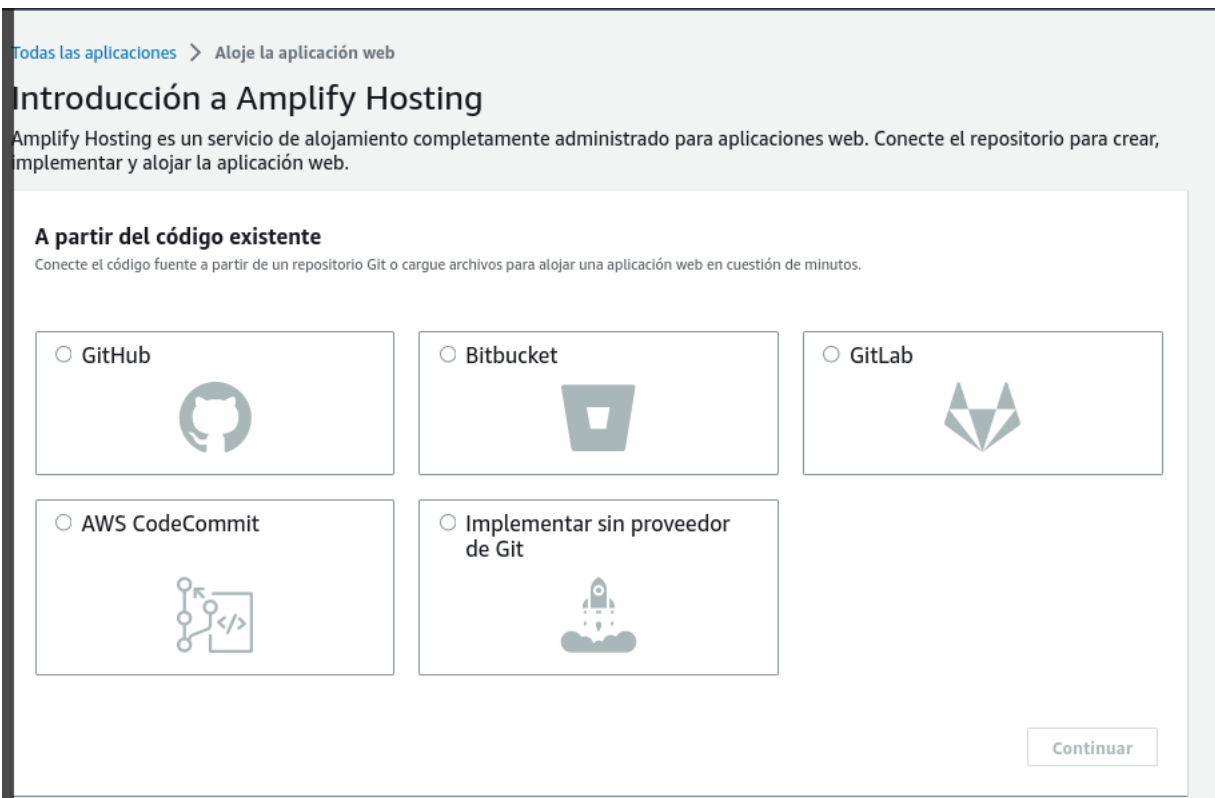


Figura 10: Amplify Hosting

Hosting requiere alguna fuente de alojamiento de código para poder desplegar la aplicación web. En este caso, la aplicación web fue alojada utilizando *Github*, por lo tanto, se selecciona esta opción. Posteriormente, se requiere seleccionar el repositorio el cual será alojado, en este caso, el nombre de dicho repositorio es *eb*. A su vez, se selecciona la ramificación *main* para ser desplegada.

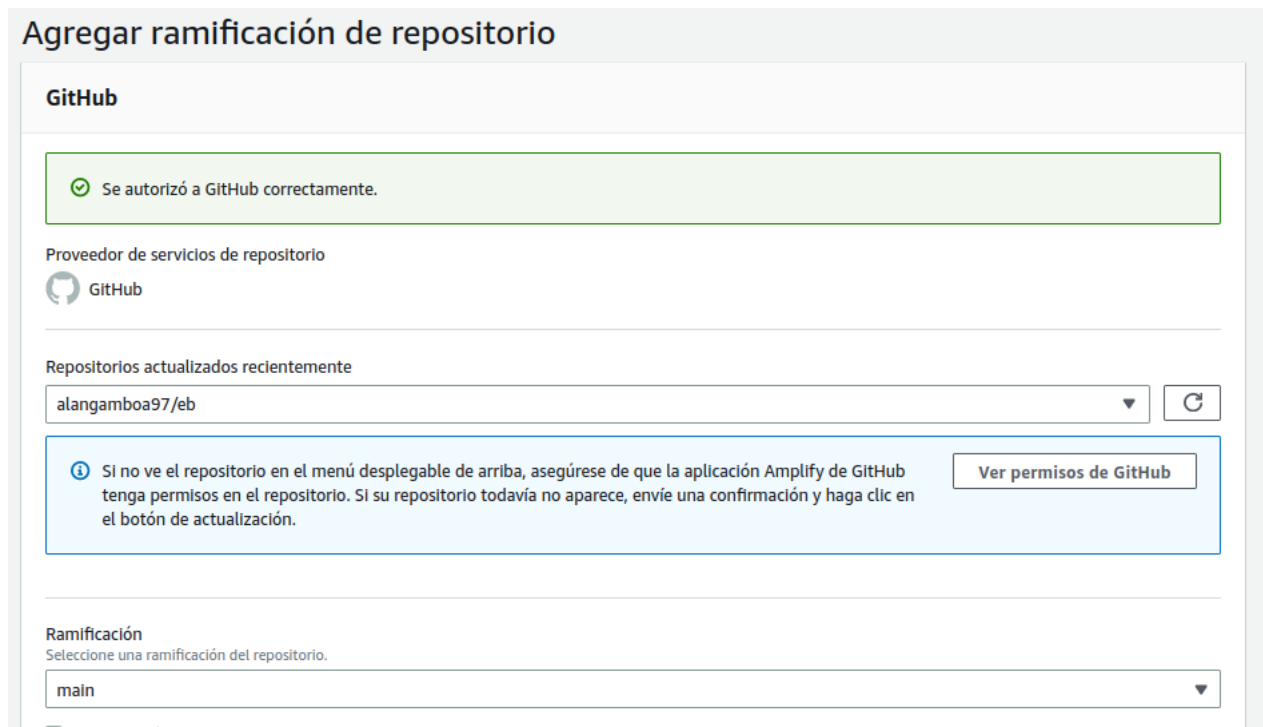


Figura 11: ramificación de repositorio de Github

2.3. Resultados

Acceso a la pagina web desde el buscador con la siguiente URL:
<https://main.dcst1c83llut3.amplifyapp.com/login>

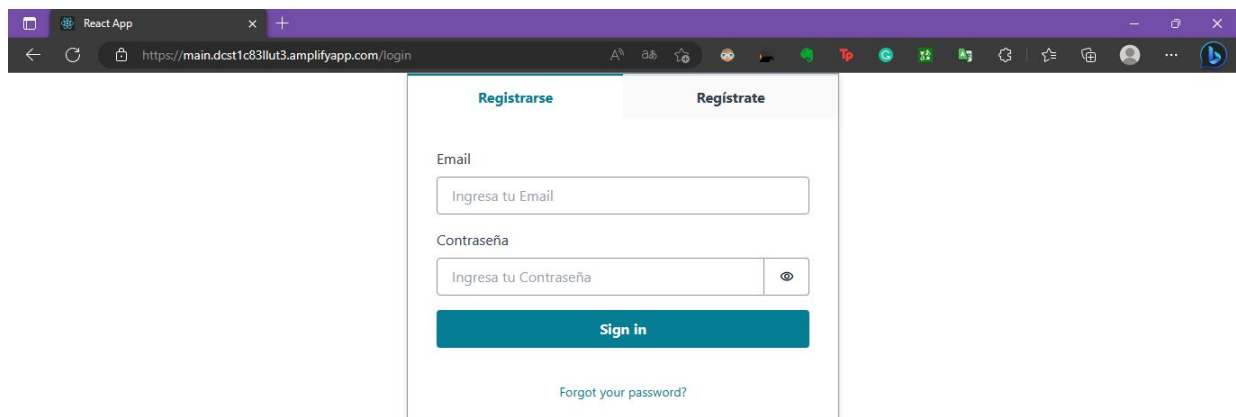


Figura 12: Acceso a la página web desde el buscador

3. Realizar la conexión de la aplicación web con el Módulo Central de Procesamiento´

3.1. Objetivo

Implementación de técnicas y configuraciones necesarias para realizar la conexión de la aplicación web

3.2. Descripción

La primer configuración que se realizó, fue la habilitación y activación de la red 4G Lte en la Jetson Nano por medio de la interfaz wwan0, que nos permite conectarnos a internet por medio de la red celular.

Posteriormente se realizó la conexión del Módulo Central de Procesamiento y la aplicación web, se utilizó la librería **Boto3**

Boto3 es una biblioteca de Python desarrollada por Amazon Web Services (AWS) que permite interactuar con los servicios de AWS de manera programática. Proporciona una interfaz de programación de aplicaciones (API) fácil de usar para acceder y administrar recursos en la nube de AWS.

Para crear una conexión a AWS Amplify utilizando Boto3, se siguieron los siguientes pasos:

En primer lugar, se instaló Boto3 utilizando el comando: `pip install boto3`. A continuación, se configuraron las credenciales de AWS, proporcionando las claves de acceso de IAM que otorgan acceso a los servicios de AWS. Estas credenciales se configuraron manualmente en el archivo de configuración. Luego, se importó el módulo Boto3 en el script de Python y se creó una conexión al servicio de AWS Amplify utilizando las credenciales configuradas anteriormente. A partir de la conexión establecida, se pudieron utilizar los métodos proporcionados por el cliente de Boto3 para interactuar con AWS Amplify.

3.3. Resultados

```

nano@nano-desktop: ~
nano@nano-desktop:~$ sudo systemctl enable simcom_wwan@wwan0.service
[sudo] password for nano:
nano@nano-desktop:~$ sudo systemctl start simcom_wwan@wwan0.service
nano@nano-desktop:~$ sudo systemctl status simcom_wwan@wwan0.service
● simcom_wwan@wwan0.service - 4G LTE network interface wwan0
   Loaded: loaded (/lib/systemd/system/simcom_wwan@.service; indirect; vendor pr
   Active: active (exited) since Wed 2023-05-17 16:52:10 CST; 5min ago
   Main PID: 7285 (code=exited, status=0/SUCCESS)
     Tasks: 1 (limit: 4181)
    CGroup: /system.slice/system-simcom_wwan.slice/simcom_wwan@wwan0.service
            └─7285 /sbin/dhclient -1 wwan0

may 17 16:51:55 nano-desktop systemd[1]: Starting 4G LTE network interface wwan0
may 17 16:52:10 nano-desktop bash[4755]: [+] wwan0 at /dev/ttyUSB2 is ready for
may 17 16:52:10 nano-desktop dhclient[7285]: DHCPDISCOVER on wwan0 to 255.255.25
may 17 16:52:10 nano-desktop dhclient[7285]: DHCPREQUEST of 10.30.139.38 on wwan
may 17 16:52:10 nano-desktop dhclient[7285]: DHCPOFFER of 10.30.139.38 from 10.3
may 17 16:52:10 nano-desktop dhclient[7285]: DHCPACK of 10.30.139.38 from 10.30.
may 17 16:52:10 nano-desktop dhclient[7285]: bound to 10.30.139.38 -- renewal in
may 17 16:52:10 nano-desktop systemd[1]: Started 4G LTE network interface wwan0.
lines 1-16/16 (END)

```

Figura 13: Red 4G LTE activa por medio de la interfaz wwan0

Conexión a AWS Amplify utilizando Boto3

```

#credenciales

dynamodb = boto3.resource('dynamodb',aws_access_key_id='AKIAVUQ3J33Z7NM07M5S',
                           aws_secret_access_key='[REDACTED]',
                           region_name='us-east-1')
table = dynamodb.Table('Incidencia-trjjwxbd3bbjrjaupwd23ijpi-dev')

location = geocoder.ip('me')
print(location.latlng)
coordenadas = json.loads(json.dumps(location.latlng), parse_float=Decimal)

nombre = 'prueba'
file = nombre + '.avi'
id = str(uuid.uuid1())
url_video = 'https://d3gh7t05x84ron.cloudfront.net/'+ id + '.mp4'
cam = cv2.VideoCapture(0)
video = VideoWriter(file, VideoWriter_fourcc(*'XVID'), 15, (640,480))

```

Figura 14: Conexión hacia Aws

4. Implementación del modelo de red neuronal para la detección de somnolencia en la Jetson Nano

4.1. Objetivo

Integrar la red neuronal entrenada en conjunto con los algoritmos de detección de rostro y ojos, y la metrica MOR.

4.2. Descripción

Se cargó el modelo de red neuronal convolucional (CNN) previamente entrenado para la detección de ojos cerrados y abiertos. Luego, se inicializó la cámara para capturar el flujo de video utilizando el método gstreamer pipeline. A continuación, se inició un bucle infinito para procesar cada cuadro de video capturado. Dentro del bucle, cada cuadro de video se convirtió a escala de grises y se utilizaron clasificadores de cascada de Haar para detectar caras y ojos en el cuadro de video. Posteriormente, se extrajo el área de interés correspondiente a cada ojo detectado. Se realizó el preprocesamiento necesario en cada ojo para que coincidiera con el formato de entrada del modelo y se llevó a cabo una predicción utilizando el modelo entrenado para determinar si los ojos estaban cerrados o abiertos. Se calculó un puntaje basado en el número de ojos cerrados detectados y se mostró este puntaje junto con un texto que indica si los ojos estaban cerrados o abiertos en el cuadro de video. Si el puntaje supera el umbral determinado, se consideraba que la persona estaba somnolienta y se activaba una alerta. El bucle continúa hasta que se presiona la tecla 'q', momento en el cual se liberaran los recursos de la cámara y se cierran las ventanas de visualización

```
face = cv2.CascadeClassifier('/usr/share/opencv4/haarcascades/haarcascade_frontalface_default.xml')
leye = cv2.CascadeClassifier('/usr/share/opencv4/haarcascades/haarcascade_lefteye_2splits.xml')
reye = cv2.CascadeClassifier('/usr/share/opencv4/haarcascades/haarcascade_righteye_2splits.xml')

eyes=cv2.CascadeClassifier('/usr/share/opencv4/haarcascades/haarcascade_eye.xml')

lbl=['Closed eyes','Open eyes']

config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.compat.v1.Session(config=config)
tf.compat.v1.keras.backend.set_session(sess)

model = tf.keras.models.load_model('/home/nano/Descargas/CNN_model.h5')
path = os.getcwd()
cap = cv2.VideoCapture(gstreamer_pipeline(), cv2.CAP_GSTREAMER) #to access the camera
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
count=0
score=0
thicc=2
rpred=[99]
lpred=[99]
```

Figura 15: Código de somnolencia parte 1


```

ret, frame = cap.read()
height,width = frame.shape[:2]

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25)) #deteccion de rostro
eye=eyes.detectMultiScale(gray) #deteccion de ojo
left_eye = leye.detectMultiScale(gray) #deteccion de ojo izquierdo
right_eye = reye.detectMultiScale(gray) # deteccion de ojo derecho

cv2.rectangle(frame, (0,height-50) , (200,height) , (0,0,0) ,thickness=cv2.FILLED)

#for (x,y,w,h) in faces:
|   #cv2.rectangle(frame, (x,y) , (x+w,y+h) , (150,150,150) , 1)

for (x,y,w,h) in eye:
|   cv2.rectangle(frame, (x,y),(x+w,y+h) ,(150,150,150) , 1)

for (x,y,w,h) in right_eye:
    r_eye=frame[y:y+h,x:x+w]
    count=count+1
    r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)
    r_eye = cv2.resize(r_eye,(100,100))
    r_eye= r_eye/255
    r_eye= r_eye.reshape(100,100,-1)
    r_eye = np.expand_dims(r_eye,axis=0)
    rpred = model.predict_classes(r_eye)

    if(rpred[0]==1):
|       lbl='Open'
    if(rpred[0]==0):
|       lbl='Closed'
    break

```

Figura 16: Código de somnolencia parte 2

```

for (x,y,w,h) in left_eye:
    l_eye=frame[y:y+h,x:x+w]
    count=count+1
    l_eye = cv2.cvtColor(l_eye,cv2.COLOR_BGR2GRAY)
    l_eye = cv2.resize(l_eye,(100,100))
    l_eye= l_eye/255
    l_eye=l_eye.reshape(100,100,-1)
    l_eye = np.expand_dims(l_eye,axis=0)
    lpred = model.predict_classes(l_eye)
    if(lpred[0]==1):
|       lbl='Open'
    if(lpred[0]==0):
|       lbl='Closed'
    break

if(rpred[0]==0 and lpred[0]==0):
    score=score+1
    cv2.putText(frame,"Cerrados",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

# if(rpred[0]==1 or lpred[0]==1):
else:
    score=score-1
    cv2.putText(frame,"Abiertos",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

if(score<0):
    score=0
cv2.putText(frame,' Puntaje:'+str(score),(100,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

```

Figura 17: Código de somnolencia parte 3


```

cv2.putText(frame,"Cerrados",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

# if(rpred[0]==1 or lpred[0]==1):
else:
    score=score-1
    cv2.putText(frame,"Abiertos",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

if(score<0):
    score=0
cv2.putText(frame,' Puntaje:'+str(score),(100,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

if(score>10):
    #person is feeling sleepy so we beep the alarm
    cv2.imwrite(os.path.join(path,'image.jpg'),frame)
    try:
        print('alerta')
    except:
        pass
    if(thicc<16):
        thicc= thicc+2
    else:
        thicc=thicc-2
        if(thicc<2):
            thicc=2
    cv2.rectangle(frame,(0,0),(width,height),(0,0,255),thicc)
#cv2.imshow('Driver drowsiness detection',frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
cap.release()
cv2.destroyAllWindows()

```

Figura 18: Código de somnolencia parte 4

El archivo respecto a la métrica MOR, se complementó con la grabación del video y el envío del video hacía la página web.

```

#subir video

def upload_video(file):

    client = boto3.client('s3',
                          aws_access_key_id='AKIAVUQ3J33Z7NMO7M5S',
                          aws_secret_access_key='[REDACTED]',
                          region_name='us-east-1')

    bucket = 'videos175126-dev'
    cur_path = os.getcwd()

    filename = os.path.join(cur_path,file)

    #abrir archivo
    data = open(filename, 'rb')

    #subir archivo
    client.upload_file(filename, bucket, file)
    table.put_item(
        Item={
            'id': id,
            'estado': None,
            'url_video': url_video,
            'ubicacion': coordenadas,
            'createdAt': datetime.now().isoformat(timespec='milliseconds') + 'Z',
            'updatedAt': datetime.now().isoformat(timespec='milliseconds') + 'Z',
            '__typename': 'Incidencia',

```

Figura 19: Grabación del video y el envío del video hacía la página web

4.3. Resultados

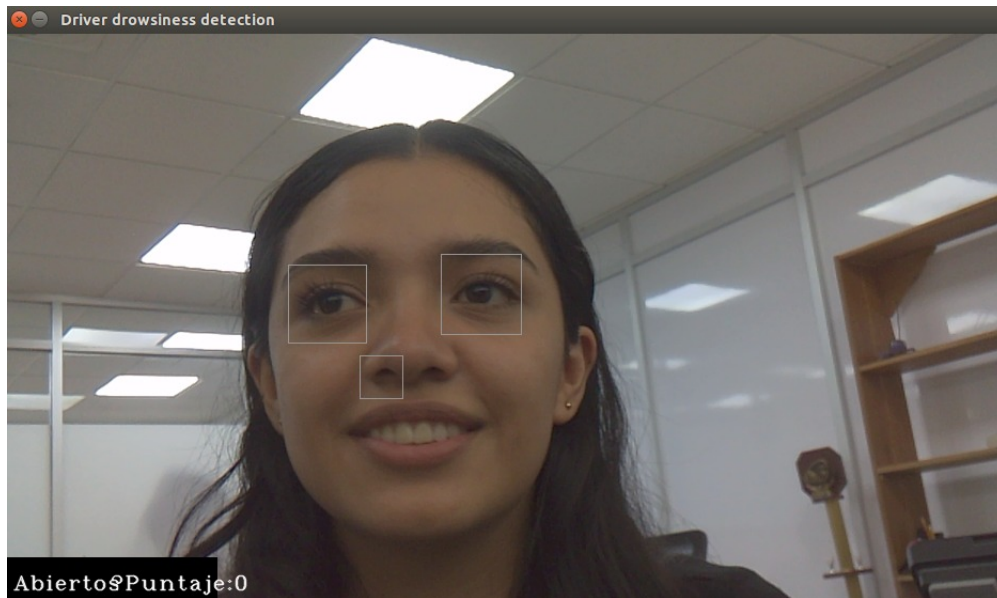


Figura 20: Prueba de somnolencia ojos abiertos

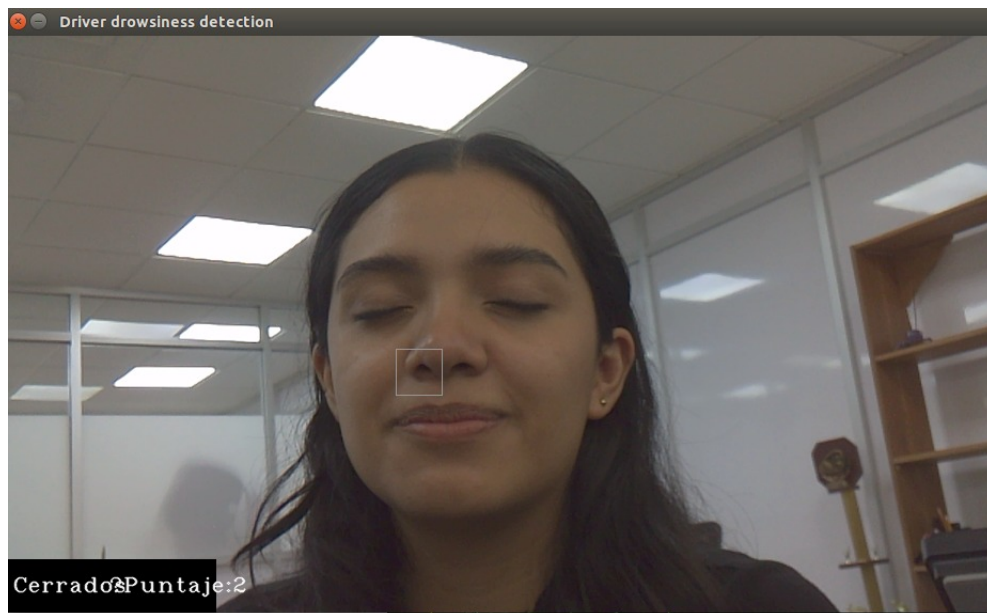


Figura 21: Prueba de somnolencia ojos cerrados

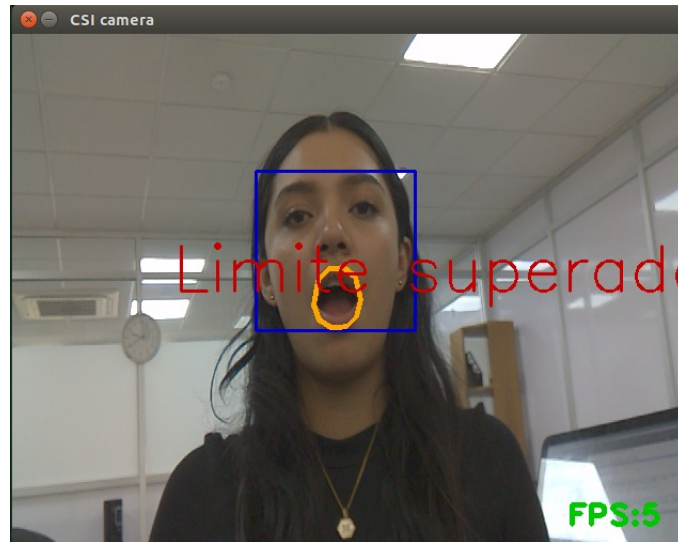


Figura 22: Prueba de la metrica MOR 1

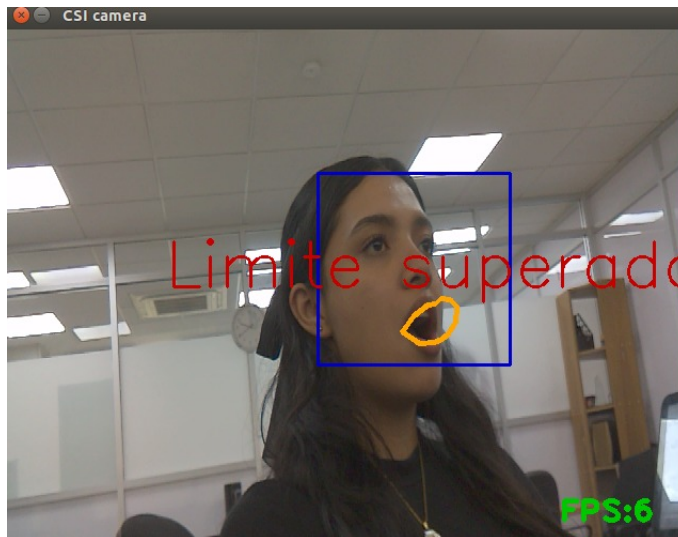


Figura 23: Prueba de la metrica MOR 2

5.3. Resultados

contenedor de la Jetson nano y los periféricos integrados:



Figura 25: Carcasa armada con la Jetson Nano y periféricos integrados

6. Conclusiones

En conclusión, en este proyecto se habilitó el GPS en la Jetson Nano a través de la terminal utilizando comandos AT y se verificó su estado mediante la lectura de la salida del puerto serial. Luego, se interactuó con el módulo SIM7600X a través de un programa en Python para obtener los datos de posición GPS. Se utilizó AWS Maps, un servicio de Amazon Web Services, para crear un mapa personalizado en la nube. Se configuraron la apariencia y la funcionalidad del mapa, incluyendo los estilos de visualización, las capas de datos y las interacciones del usuario. Se implementó una función para obtener el historial de posiciones del dispositivo, utilizando la función `client.getDevicePositionHistory` de AWS. Los resultados se mapearon en un objeto que contenía las propiedades de la posición del dispositivo. Se guardó este archivo con extensión `.jsx`. Para desplegar la aplicación web en su totalidad, se utilizó Amplify Hosting como servicio de hosting. Se seleccionó el repositorio alojado en GitHub y se configuró la conexión utilizando la biblioteca Boto3 de Python, que permite interactuar con los servicios de AWS. Además, se cargó un modelo de red neuronal convolucional (CNN) previamente entrenado para la detección de ojos cerrados y abiertos. Se utilizó la cámara de la Jetson Nano para capturar el flujo de video y se procesó cada cuadro de video para detectar caras y ojos. Se realizó una predicción utilizando el modelo entrenado para determinar si los ojos estaban cerrados o abiertos. Se mostró un puntaje y un texto indicando si los ojos estaban cerrados o abiertos, y se activó una alerta si el puntaje superaba un umbral establecido. Finalmente se integraron los periféricos junto con la Jetson nano dentro de una carcasa de metal.

Este proyecto integró el uso del GPS en la Jetson Nano, la visualización de datos geográficos en un mapa personalizado utilizando AWS Maps, la obtención de historial de posiciones del dispositivo, el despliegue de una aplicación web utilizando Amplify Hosting, la detección de ojos cerrados y abiertos utilizando un modelo de red neuronal convolucional y la integración de los periféricos con la Jetson Nano dentro de una carcasa de metal.

7. Bibliografía

Referencias

- [1] React Dev Team, *React*, React. <https://react.dev/> (accedido el 1 de abril de 2023).
- [2] Waveshare Electronics, *SIM7600G-H 4G for Jetson Nano - Waveshare Wiki*, Waveshare Electronics https://www.waveshare.com/wiki/SIM7600G-H_4G_for_Jetson_Nano_4G_connecting (accedido el 16 de abril de 2023).
- [3] Facebook Dev Team, *Introduction to GraphQL — GraphQL — A query language for your API* <https://graphql.org/learn/> (accedido el 4 de abril de 2023).
- [4] NVIDIA, "Get Started with the Jetson Nano Developer Kit", NVIDIA Developer, 2019. [Online]. Disponible: <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro>. [Accedido: Abril 02 2023].
- [5] Nvidia Developer, "Get Started with Jetson Nano Devkit," Nvidia Developer. [En línea]. Disponible: <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write>. [Accedido: 2 de abril de 2023].
- [6] Dusty, N. "Building the Repo - NVIDIA Jetson Inference," GitHub. [Online]. Disponible en: <https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md>. [Accedido en: 02-abr-2023].