



“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	4
2. Despliegue de la aplicación web	7
2.1. Objetivo	7
2.2. Descripción	7
2.3. Resultados	9
3. Realizar la conexión de la aplicación web con el Módulo Central de Procesamiento	10
3.1. Objetivo	10
3.2. Descripción	10
3.3. Resultados	12
4. Implementación del modelo de red neuronal para la detección de somnolencia en la Jetson Nano	13
4.1. Objetivo	13
4.2. Descripción	13
4.3. Resultados	19
5. Realizar la integración de los periféricos al Módulo Central de Procesamiento	20
5.1. Objetivo	20
5.2. Descripción	20
5.3. Resultados	21
6. Conclusiones	22
7. Bibliografía	23

Índice de figuras

1.	Página Principal - Layout.jsx	5
2.	Vista Reporte Incidencia Incidencia - Incidencia.jsx	5
3.	Vista Ubicacion	6
4.	Vista Conductores - Conductores.jsx	6
5.	Módulo Base-Hat SIM7600G-H	7
6.	Instalación de librerías	8
7.	Instalación de librerías	8
8.	Minicom	9
9.	Actualización de drivers	9
10.	Establecer dirección IP	9
11.	Configuración de servicio de almacenamiento S3	10
12.	Generación de Endpoint de GraphQL	10
13.	Generación de Endpoint de GraphQL	10
14.	Generación de Endpoint de GraphQL	11
15.	Generación de Endpoint de GraphQL	11
16.	Generación de Endpoint de GraphQL	11
17.	Generación de Endpoint de GraphQL	12
18.	Tablas generadas mediante los schemas definidos	12
19.	Función getConductor	13
20.	Función listConductors	14
21.	Función getIncidencia	14
22.	Función listIncidencias	15
23.	Función createConductor	15
24.	Función updateConductor	16
25.	Función deleteConductor	16
26.	Función createIncidencia	17
27.	Función oncreateIncidencia	17
28.	Consola de AWS	18
29.	Funcionamiento de Appsync	18
30.	Crear Conductor	19
31.	Resultado	19
32.	Funcionamiento de Amazon Cognito	21

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

ReactJs

Para el desarrollo del front-end del presente proyecto, se hará uso de la librería de diseño de ReactJs. ReactJs facilita la creación de componentes reutilizables e interactivos para las interfaces de usuario.

Los componentes que darán lugar a las vistas del presente proyecto son los siguientes:

- **Layout.jsx:** Este componente será la vista principal de la Aplicación Web. Se trata de un diseño tipo *dashboard* que contendrá una sección principal que contendrá etiquetas para poder ingresar a las diferentes vistas de la aplicación. Además estará compuesta también de una sección secundaria que mostrará el contenido de dichas vistas.
- **Incidencias.jsx:** Este componente se encargará de mostrar todas las incidencias registradas en la base de datos. Las incidencias serán desplegadas en forma de lista.
- **Incidencias.jsx:** Este componente de mostrar un reporte de incidencia a detalle. Contendrá una ventana que permitirá ver el video del momento de la incidencia registrada. Así como los datos de la fecha y hora. Además de botones para poder confirmar o rechazar la incidencia. Finalmente contendrá el nombre del conductor además de una opción para poder consultar la ubicación en tiempo real del conductor.
- **Ubicación.jsx:** Este componente mostrará la ubicación en tiempo real del conductor con ayuda del servicio de diseño de mapas Leaflet.
- **Conductores.jsx:** Este componente mostrará todos los conductores registrados en la base de datos en forma de lista.

1.3. Resultados

De acuerdo con los componentes explicados anteriormente, las vistas que contendrá la aplicación web son las siguientes:

- **Página Principal**

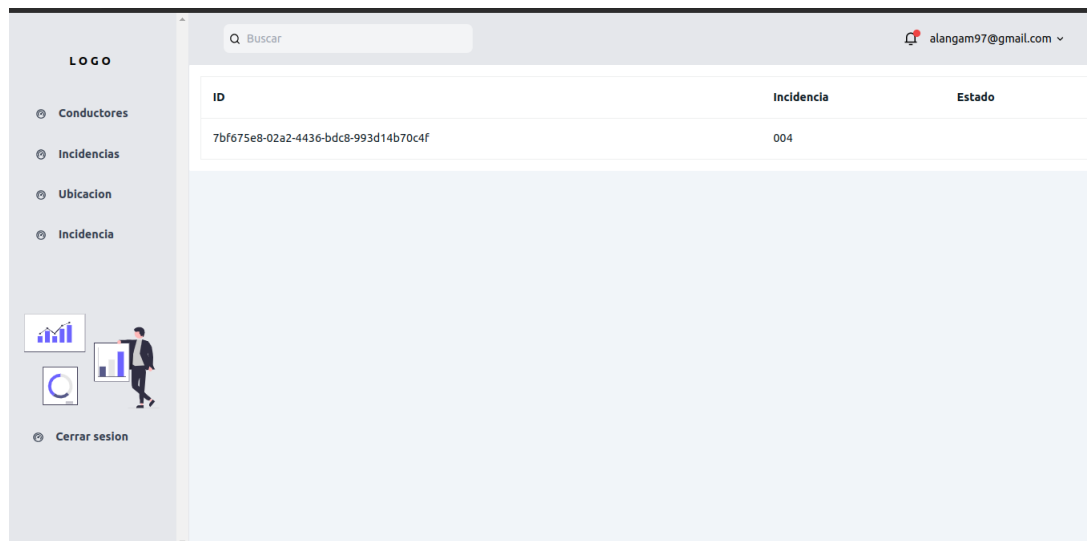


Figura 1: Página Principal - Layout.jsx

- Reporte de Incidencia

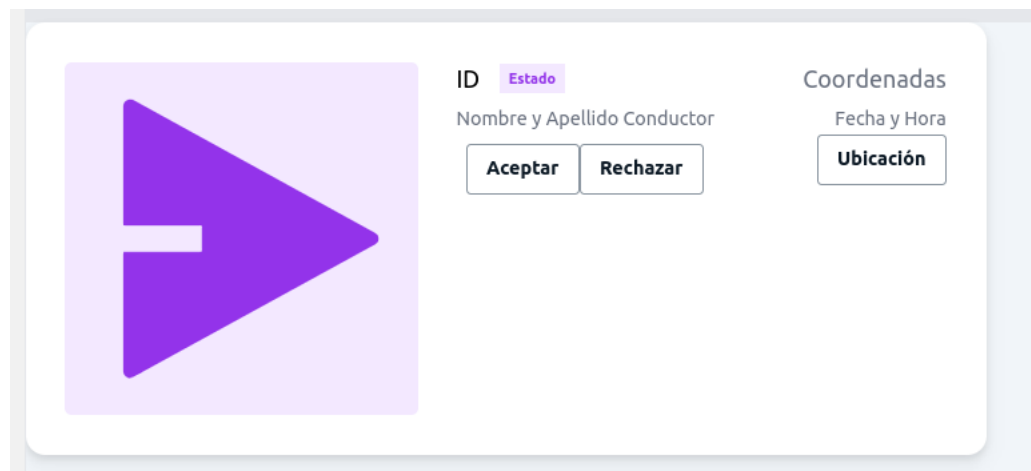


Figura 2: Vista Reporte Incidencia Incidencia - Incidencia.jsx

- Ubicación

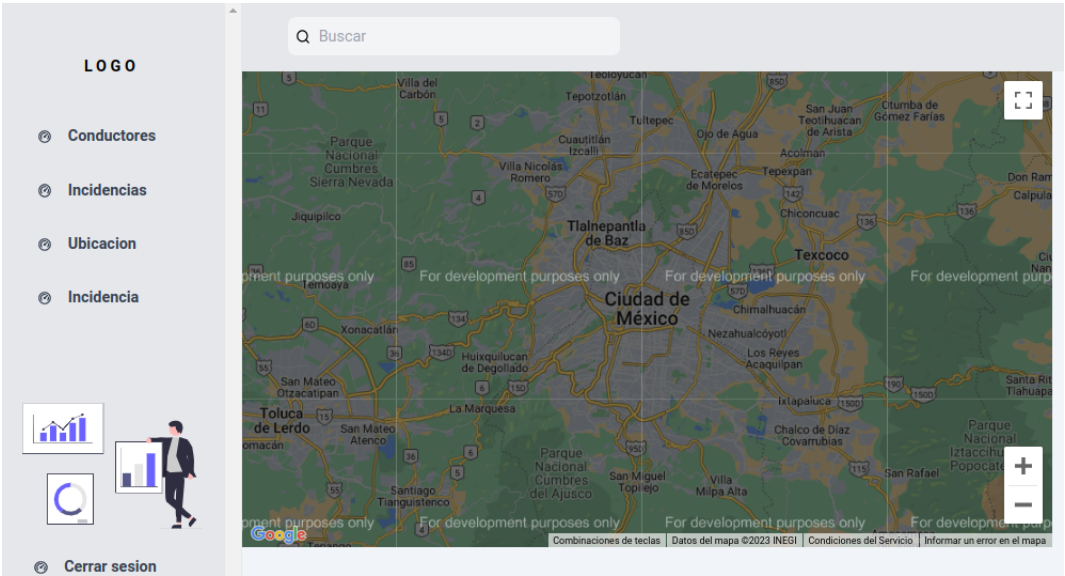


Figura 3: Vista Ubicacion

■ Conductores

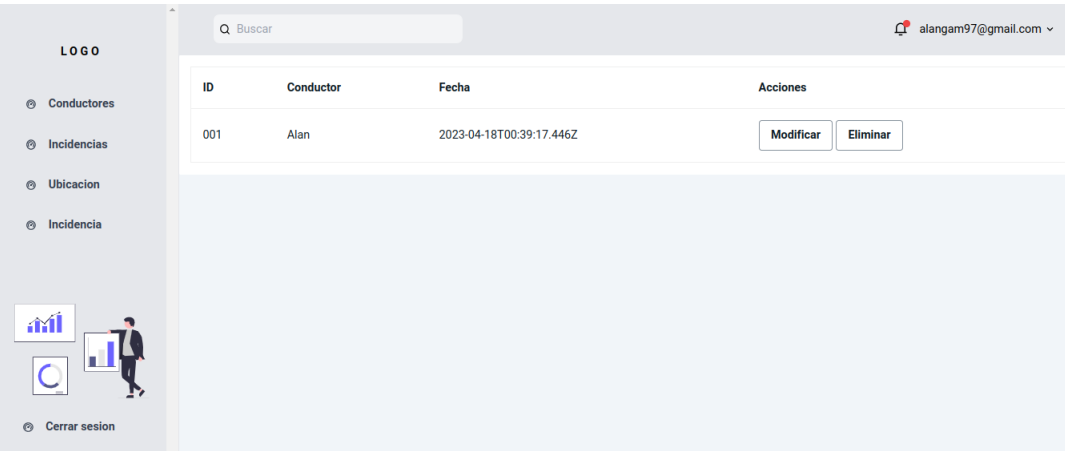


Figura 4: Vista Conductores - Conductores.jsx

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 5: Módulo Base-Hat SIM7600G-H

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.

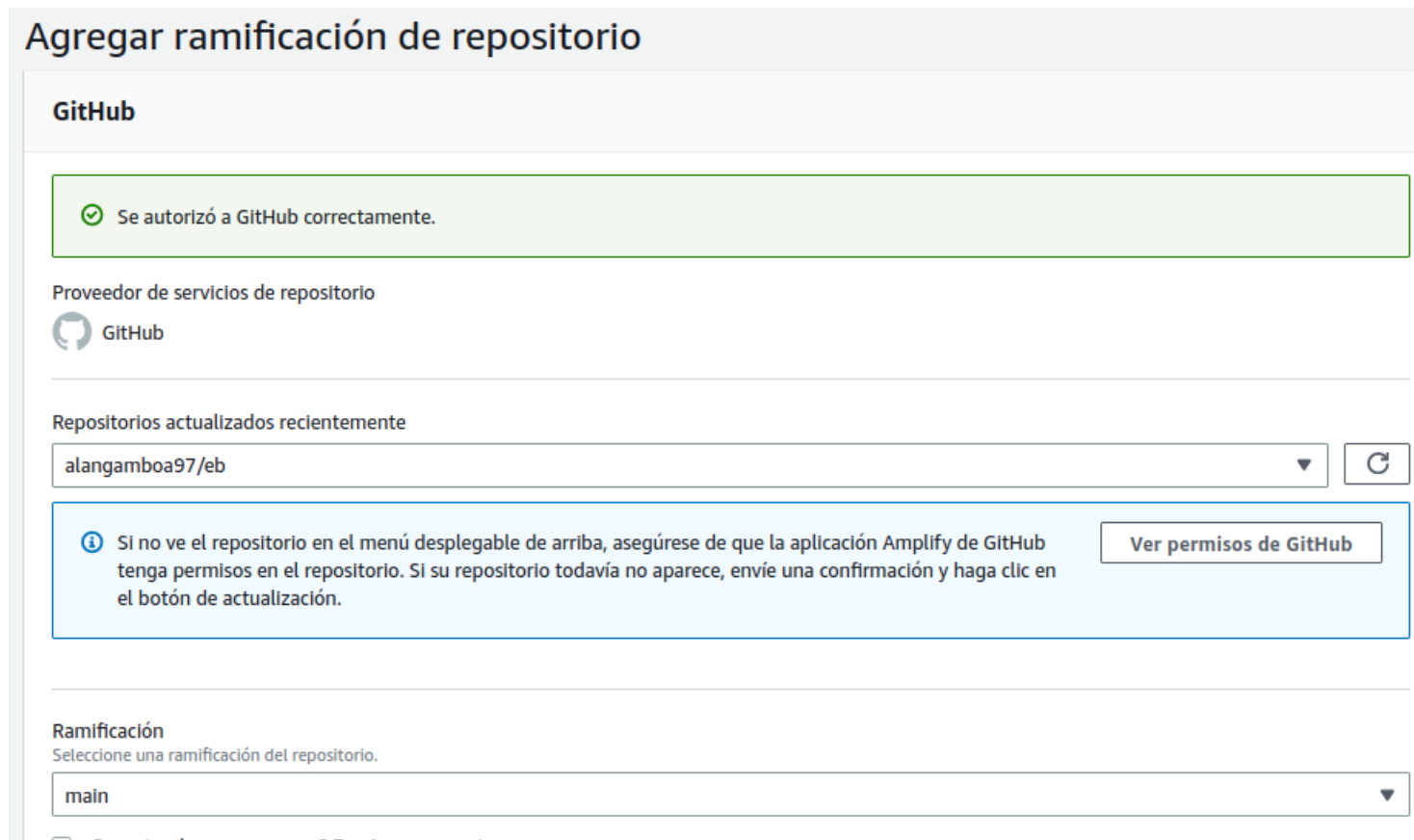


Figura 6: Instalación de librerías

Los comandos ingresados en la figura 6 se encargan de instalar todas las librerías y el software necesario para poder comenzar a utilizar la red LTE mediante el módulo SIM7600G-H. Además también se crea un directorio que contendrá la configuración de usuario así como un cuenta enlazada al módulo.

Posteriormente, probamos que el puerto GPIO de nuestra Jetson Nano esté funcionando con los siguientes comandos:

```
echo 200 > /sys/class/gpio/export
echo out > /sys/class/gpio200/direction
echo 1 > /sys/class/gpio200/value
echo 0 > /sys/class/gpio200/value
```

Figura 7: Instalación de librerías

Después de haber realizado los pasos anteriores, el pin con el nombre NET deberá parpadear constantemente, lo que significa que el módulo está listo para ser utilizado.

Para realizar la comunicación LTE, primero se ingresa a la librería minicom utilizando los siguientes comandos:

```
sudo su
killall ModemManager
minicom -D /dev/ttyUSB2
```

Figura 8: Minicom

Posteriormente, se necesita actualizar los drivers:

```
-----
cd
wget https://www.waveshare.com/w/upload/4/46/Simcom_wwan.zip
unzip Simcom_wwan.zip
cd simcom_wwan
sudo su
make
```

Figura 9: Actualización de drivers

Finalmente se establece una dirección IP con el siguiente comando:

- Allocate IP

```
apt-get install udhcpc
udhcpc -i wwan0
```

Figura 10: Establecer dirección IP

2.3. Resultados

Se realizó la investigación para conocer los pasos necesarios para establecer comunicación LTE utilizando el módulo SIM7600G-H para la NVIDIA Jetson Nano.

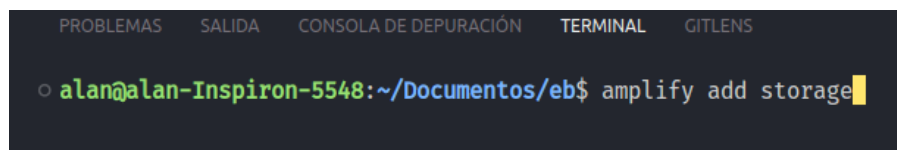
3. Realizar la conexión de la aplicación web con el Módulo Central de Procesamiento´

3.1. Objetivo

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

3.2. Descripción

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



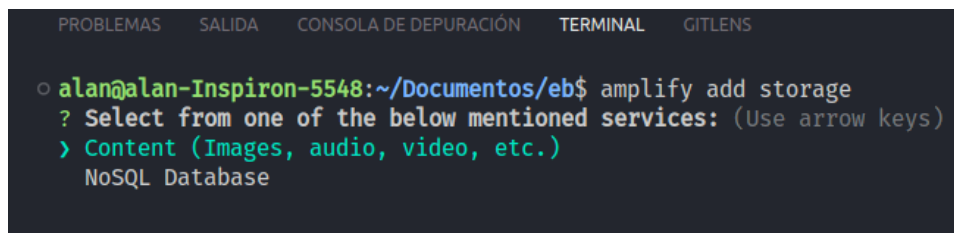
```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage

```

Figura 11: Configuración de servicio de almacenamiento S3

Posteriormente, se requiere especificar que tipo de servicio de almacenamiento se integrará a la aplicación (multimedia o base de datos NoSQL). Para el presente proyecto, se utilizará el almacenamiento de contenido multimedia, por lo tanto, se seleccionará dicha opción.



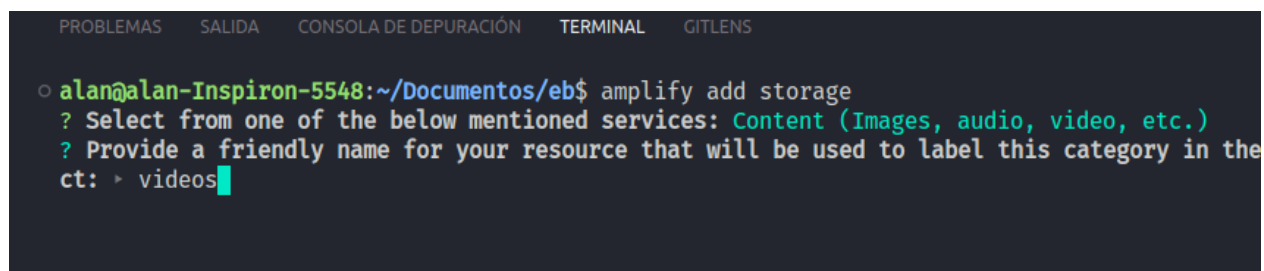
```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: (Use arrow keys)
> Content (Images, audio, video, etc.)
  NoSQL Database

```

Figura 12: Generación de Endpoint de GraphQL

Ingresamos el nombre de nuestro espacio de almacenamiento:



```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
? Provide a friendly name for your resource that will be used to label this category in the
ct: > videos

```

Figura 13: Generación de Endpoint de GraphQL

Después, se necesita establecer cuantos usuarios, así como cuales podrán acceder a dicho servicio:

```
o alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
✓ Provide a friendly name for your resource that will be used to label this category in
  ct: · videos

✓ Provide bucket name: · videos
? Who should have access: ... (Use arrow keys or type to filter)
> Auth users only
  Auth and guest users
```

Figura 14: Generación de Endpoint de GraphQL

```
o alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add storage
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
✓ Provide a friendly name for your resource that will be used to label this category in
  ct: · videos

✓ Provide bucket name: · videos
✓ Who should have access: · Auth and guest users
? What kind of access do you want for Authenticated users? ... (Use arrow keys or type to
  ● create/update
  ● read
  >● delete
  (Use <space> to select, <ctrl + a> to toggle all)
```

Figura 15: Generación de Endpoint de GraphQL

```
? Select from one of the below mentioned services: Content (Images, audio, video, etc.)
✓ Who should have access: · Auth and guest users
✓ What kind of access do you want for Authenticated users? · create/update, read, delete
? What kind of access do you want for Guest users? ... (Use arrow keys or type to filter)
>● create/update
  ● read
  o delete
  (Use <space> to select, <ctrl + a> to toggle all)
```

Figura 16: Generación de Endpoint de GraphQL

```

Edit your schema at /home/alan/Documentos/eb/amplify/backend/api/ebase/schema.graphql or place
graphql files in a directory at /home/alan/Documentos/eb/amplify/backend/api/ebase/schema
✓ Successfully pulled backend environment dev from the cloud.

Current Environment: dev

```

Category	Resource name	Operation	Provider plugin
Storage	videos	Create	awscloudformation
Auth	ebase35f92a6b	Update	awscloudformation
Function	S3Trigger6956e03e	No Change	awscloudformation
Api	ebase	No Change	awscloudformation

```

? Are you sure you want to continue? (Y/n) >

```

Figura 17: Generación de Endpoint de GraphQL

3.3. Resultados

Al ingresar a la consola de servicios de AWS, en la sección de buckets de S3, se puede observar que se encuentra el bucket recién creado llamado *videos175126-dev*.

	Nombre	Región de AWS	Acceso	Fecha de creación
<input type="radio"/>	amplify-ebase-dev-175126-deployment	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos	28 Feb 2023 5
<input type="radio"/>	ebase6c6dcb3b214e4c3fa82df5d47dce7c22175126-dev	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos	25 Mar 2023 1
<input type="radio"/>	videos175126-dev	EE. UU. Este (Norte de Virginia) us-east-1	Los objetos pueden ser públicos	17 Apr 2023 1

Figura 18: Tablas generadas mediante los schemas definidos

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

GraphQL trabaja con 3 tipos de archivos:

- *Queries*: Este archivo contiene las funciones que permitirán acceder a los datos.
- *Mutations*: En este archivo se encuentran todas las funciones que permitirán realizar el manejo de datos (actualizar, eliminar, agregar)
- *Subscriptions*: Las *subscriptions* en GraphQL son funciones de consulta especiales, que se envían a travez de un punto de conexión websocket. Permiten realizar cierta operación cada vez que se ejecuta una acción en el backend.

Para comenzar con el archivo de *Queries* se tienen las siguientes funciones:

```
export const getConductor = /* GraphQL */ `
  query GetConductor($id: ID!) {
    getConductor(id: $id) {
      id
      nombre
      apellido
      incidencias {
        items {
          id
          estado
          url_video
          ubicacion
          fecha_hora
          createdAt
          updatedAt
          conductorIncidenciasId
        }
        nextToken
      }
      num_incidencias
      createdAt
      updatedAt
    }
  }
`;
```

Figura 19: Función getConductor

La función de la figura ??, obtiene los datos de un solo Conductor.

```
export const listConductors = /* GraphQL */ `
  query ListConductors(
    $filter: ModelConductorFilterInput
    $limit: Int
    $nextToken: String
  ) {
    listConductors(filter: $filter, limit: $limit, nextToken: $nextToken) {
      items {
        id
        nombre
        apellido
        incidencias {
          nextToken
        }
        num_incidencias
        createdAt
        updatedAt
      }
      nextToken
    }
  }
`;
```

Figura 20: Función listConductors

La función de la figura 20 obtiene todos los datos de todos los conductores almacenados en la base de datos.

```
export const getIncidencia = /* GraphQL */ `
  query GetIncidencia($id: ID!) {
    getIncidencia(id: $id) {
      id
      conductor {
        id
        nombre
        apellido
        incidencias {
          nextToken
        }
        num_incidencias
        createdAt
        updatedAt
      }
      estado
      url_video
      ubicacion
      fecha_hora
      createdAt
      updatedAt
      conductorIncidenciasId
    }
  }
`;
```

Figura 21: Función getIncidencia

La función de la figura 21 obtiene los datos de una sola Incidencia.

```
port const listIncidencias = /* GraphQL */ `
query ListIncidencias(
  $filter: ModelIncidenciaFilterInput
  $limit: Int
  $nextToken: String
) {
  listIncidencias(filter: $filter, limit: $limit, nextToken: $nextToken) {
    items {
      id
      conductor {
        id
        nombre
        apellido
        num_incidencias
        createdAt
        updatedAt
      }
      estado
      url_video
      ubicacion
      fecha_hora
      createdAt
      updatedAt
      conductorIncidenciasId
    }
  }
  nextToken
}
```

Figura 22: Función listIncidencias

La función de la figura 22 obtiene los datos de todas las incidencias de almacenadas en la base de datos.

En cuanto al archivo de *Mutations* se tienen las siguientes funciones:

```
export const createConductor = /* GraphQL */ `
mutation CreateConductor(
  $input: CreateConductorInput!
  $condition: ModelConductorConditionInput
) {
  createConductor(input: $input, condition: $condition) {
    id
    nombre
    apellido
    incidencias {
      items {
        id
        estado
        url_video
        ubicacion
        fecha_hora
        createdAt
        updatedAt
        conductorIncidenciasId
      }
    }
    nextToken
  }
  num_incidencias
  createdAt
  updatedAt
}
```

Figura 23: Función createConductor

La función de la figura 23 se encarga de crear el registro de un conductor en la base de datos.

```
export const updateConductor = /* GraphQL */ `
mutation UpdateConductor(
  $input: UpdateConductorInput!
  $condition: ModelConductorConditionInput
) {
  updateConductor(input: $input, condition: $condition) {
    id
    nombre
    apellido
    incidencias {
      items {
        id
        estado
        url_video
        ubicacion
        fecha_hora
        createdAt
        updatedAt
        conductorIncidenciasId
      }
    }
    nextToken
  }
  num_incidencias
  createdAt
  updatedAt
}
```

Figura 24: Función updateConductor

La función de la figura 24 se encarga de modificar datos del registro de un conductor.

```
export const deleteConductor = /* GraphQL */ `
mutation DeleteConductor(
  $input: DeleteConductorInput!
  $condition: ModelConductorConditionInput
) {
  deleteConductor(input: $input, condition: $condition) {
    id
    nombre
    apellido
    incidencias {
      items {
        id
        estado
        url_video
        ubicacion
        fecha_hora
        createdAt
        updatedAt
        conductorIncidenciasId
      }
    }
    nextToken
  }
  num_incidencias
  createdAt
  updatedAt
}
```

Figura 25: Función deleteConductor

La función de la figura 25 se encarga de eliminar un conductor de la base de datos.

```
;
export const createIncidencia = /* GraphQL */ `
mutation CreateIncidencia(
  $input: CreateIncidenciaInput!
  $condition: ModelIncidenciaConditionInput
) {
  createIncidencia(input: $input, condition: $condition) {
    id
    conductor {
      id
      nombre
      apellido
      incidencias {
        nextToken
      }
      num_incidencias
      createdAt
      updatedAt
    }
    estado
    url_video
    ubicacion
    fecha_hora
    createdAt
    updatedAt
    conductorIncidenciasId
  }
}
`;
```

Figura 26: Función createIncidencia

La función de la figura 26 se encarga de crear una Incidencia en la base de datos. Para el archivo de *subscriptions* se tienen la siguiente función:

```
export const onCreateIncidencia = /* GraphQL */ `
subscription OnCreateIncidencia(
  $filter: ModelSubscriptionIncidenciaFilterInput
) {
  onCreateIncidencia(filter: $filter) {
    id
    conductor {
      id
      nombre
      apellido
      incidencias {
        nextToken
      }
      num_incidencias
      createdAt
      updatedAt
    }
    estado
    url_video
    ubicacion
    fecha_hora
    createdAt
    updatedAt
    conductorIncidenciasId
  }
}
`;
```

Figura 27: Función onCreateIncidencia

La función de la figura 27 se encarga de realizar una consulta cada vez que una Incidencia nueva es dada de alta en la base de datos.

Al estar trabajando con GraphQL dentro del proyecto, la manera en que se podrán realizar las operaciones *CRUD* - (*Create, Read, Update, Delete*), será mediante funciones JSON.

Para comprobar que la API permite dichas operaciones, se debe ingresar a la consola de AWS y dirigirse a la sección de AWS AppSync.



Figura 28: Consola de AWS

Posteriormente se necesita ingresar a la sección de consultas.

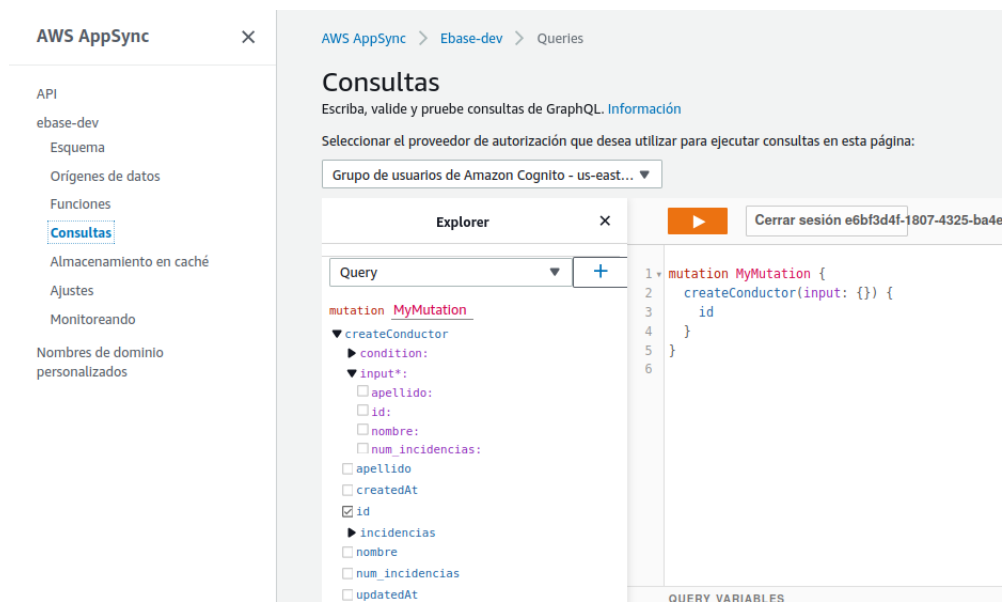


Figura 29: Funcionamiento de Appsync

AWS permite elegir si realizar un *query*, *mutation*, o *subscription* mediante código JSON

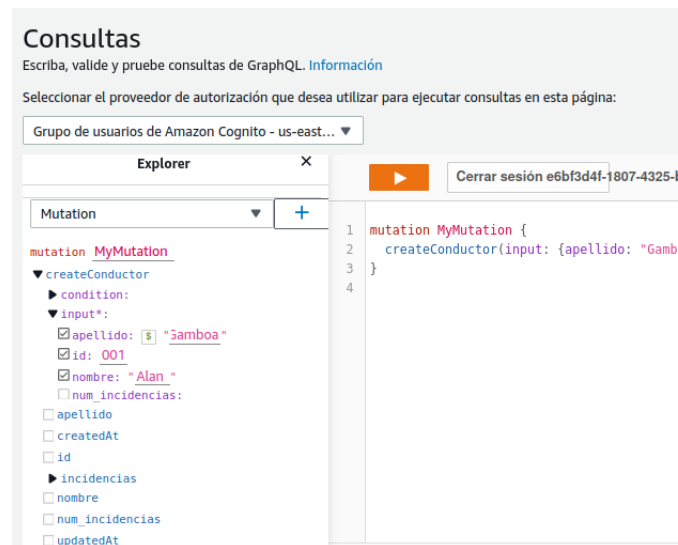


Figura 30: Crear Conductor

En la figura 31 se puede apreciar una sentencia JSON que permite utilizar las funciones previamente creadas para dar de alta un conductor.

4.3. Resultados

Como resultado tenemos un log que nos muestra la entrada creada



Figura 31: Resultado

5. Realizar la integración de los periféricos al Módulo Central de Procesamiento

5.1. Objetivo

Integrar dentro de un contenedor los periféricos junto con la Jetson Nano

5.2. Descripción

Instalación en la tarjeta microSD

El Jetson Nano Developer Kit utiliza una tarjeta microSD como dispositivo de arranque y almacenamiento principal. Por tanto fue necesario instalar un entorno de desarrollo en la propia placa, para lo cual se requirió de una tarjeta microSD de un mínimo recomendado de 32 GB de acuerdo a la documentación Nvidia. [4].

Como primer paso se descargó el *Jetson Nano Developer Kit SD Card Image* [5] y posteriormente se instaló en la tarjeta microSD desde el sistema operativo Linux, utilizando los siguientes pasos:

1. Se abrió una terminal y se insertó la tarjeta microSD.
2. Se usó el siguiente comando para mostrar que dispositivo de disco se le asignó:

```
dmesg | tail | awk '\$3 == "sd" {print}'
```

3. Se escribió la imagen de la tarjeta SD comprimida (previamente descargada) en la tarjeta microSD con el comando:

```
/usr/bin/unzip -p ~/Downloads/jetson_nano_devkit_sd_card.zip |  
sudo /bin/dd of=/dev/sda bs=1M status=progress
```

4. Finalmente se expulsó del dispositivo de disco desde la línea de comando utilizando:

```
sudo eject /dev/sda
```

Configuración y primer arranque

Jetson Nano Developer Kit permite dos formas de interactuar, la primera es por medio de otra computadora y la segunda haciendo uso de una pantalla, teclado y mouse conectados. Adicionalmente el kit de desarrollo no cuenta con una fuente de alimentación incluida por lo que se utilizó una fuente de alimentación Micro-USB(5V-2A).

Para iniciar el kit de desarrollo se conectó el mouse, la pantalla, el teclado y la fuente de alimentación, posteriormente se realizó la configuración inicial del sistema operativo el cual incluyó lo siguiente:

- Revisar y aceptar el EULA del software NVIDIA Jetson.

- Seleccionar el idioma del sistema, la distribución del teclado y la zona horaria
- Crear nombre de usuario, contraseña y nombre de la computadora.
- Seleccione el tamaño de partición de la aplicación: se recomienda utilizar el tamaño máximo sugerido.

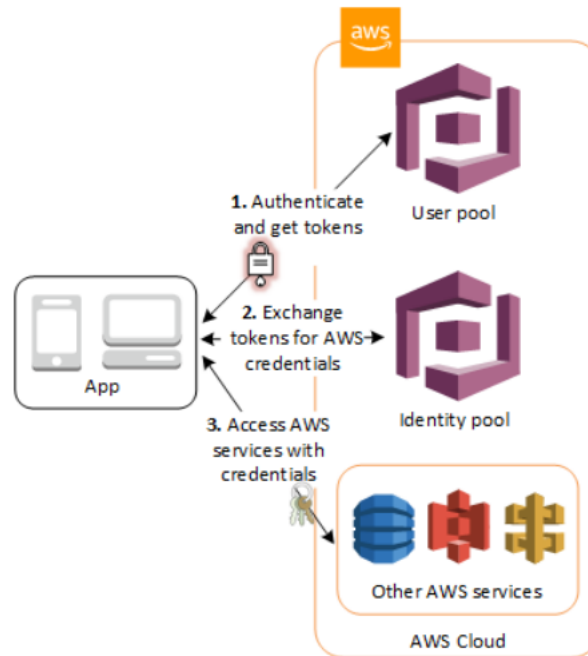


Figura 32: Funcionamiento de Amazon Cognito

5.3. Resultados

6. Conclusiones

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].