



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

Segundo Reporte Parcial

Lista de actividades

- Maquetación web
- Investigación de la documentación del módulo 3G/4G LTE-Base Hat
- Enlace de Amazon S3 con el sistema backend
- Creación de los servicios backend
- Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano
- Implementar algoritmo para la detección del rostro y ojos
- Implementar puntos faciales en el rostro y métrica MOR

Autores:

Alan Eduardo Gamboa Del
Ángel
Maite Paulette Díaz Martínez

Asesores:

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

13 de Marzo 2023

Índice

1. Maquetación web	4
1.1. Objetivo	4
1.2. Descripción	4
1.3. Resultados	7
2. Investigación de la documentación del módulo 3G/4G LTE-Base Hat	8
2.1. Objetivo	8
2.2. Descripción	8
2.3. Resultados	10
3. Enlace de Amazon S3 con el sistema backend	11
3.1. Objetivo	11
3.2. Descripción	11
3.3. Resultados	13
4. Creación de los servicios backend	14
4.1. Objetivo	14
4.2. Descripción	14
4.3. Resultados	20
5. Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano	22
5.1. Objetivo	22
5.2. Descripción	22
5.3. Resultados	23
6. Implementar algoritmo para la detección del rostro y ojos	24
6.1. Objetivo	24
6.2. Descripción	24
6.3. Resultados	24
7. Implementar puntos faciales en el rostro y la metrica MOR	25
7.1. Objetivo	25
7.2. Descripción	25
7.3. Resultados	25
8. Conclusiones	26
9. Bibliografía	27

Índice de figuras

1.	Creación proyecto de react	4
2.	Instalación React Router DOM	4
3.	Función RequireAuth	5
4.	Componente Login	5
5.	Ruta protegida del componente Home	6
6.	Directorio de rutas	6
7.	Página de Login	7
8.	Instalación Amplify CLI	8
9.	Instalación de librerías	8
10.	Instalación de librerías	9
11.	Implementación del SDK de Amplify	9
12.	Creación de aplicación de Express	9
13.	Amplify add api	10
14.	Generación de Endpoint de GraphQL	10
15.	Configuración de servicio de almacenamiento S3	11
16.	Generación de Endpoint de GraphQL	11
17.	Generación de Endpoint de GraphQL	11
18.	Generación de Endpoint de GraphQL	12
19.	Generación de Endpoint de GraphQL	12
20.	Generación de Endpoint de GraphQL	12
21.	Generación de Endpoint de GraphQL	13
22.	Tablas generadas mediante los schemas definidos	13
23.	Función getConductor	14
24.	Función listConductors	15
25.	Función getIncidencia	15
26.	Función listIncidencias	16
27.	Función createConductor	16
28.	Función updateConductor	17
29.	Función deleteConductor	17
30.	Función createIncidencia	18
31.	Función oncreateIncidencia	19
32.	Consola de AWS	19
33.	Funcionamiento de Appsync	20
34.	Crear Conductor	20
35.	Resultado	21
36.	Funcionamiento de Amazon Cognito	23

Índice de tablas

1. Maquetación web

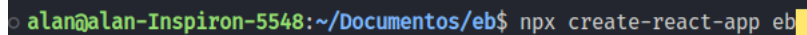
1.1. Objetivo

Definir e implementar las rutas que tendrá la aplicación, así como si serán públicas o privadas y la información que se desplegará en cada una de las mismas.

1.2. Descripción

Rutas públicas vs rutas protegidas

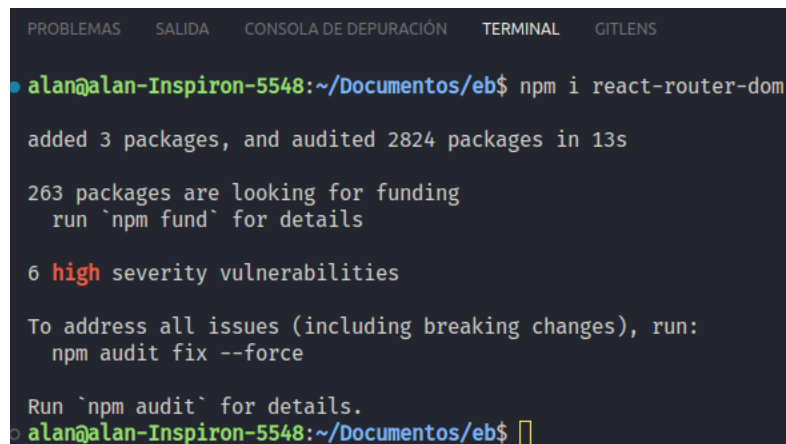
Cuando se habla de una ruta protegida en React, se refiere a programar un bloqueo en ciertas rutas a la cual se le restringe el acceso al usuario. Esto comunmente se realiza para la validación de inicio de sesión de usuarios. Sí el usuario no tiene una sesión iniciada, no podrá acceder a las rutas protegidas de la aplicación. Por otro lado, las rutas públicas son todas aquellas las cuales no requieren contar con una sesión iniciada, y pueden ser accesadas por cualquier tipo de usuario[1]. Como primer paso, se necesita crear un proyecto de React utilizando el siguiente comando:



```
alan@alan-Inspiron-5548:~/Documentos/eb$ npx create-react-app eb
```

Figura 1: Creación proyecto de react

Posteriormente, se realiza la instalación del moudelo *React Router Dom* utilizando el siguiente comando:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS

alan@alan-Inspiron-5548:~/Documentos/eb$ npm i react-router-dom

added 3 packages, and audited 2824 packages in 13s

263 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
alan@alan-Inspiron-5548:~/Documentos/eb$
```

Figura 2: Instalación React Router DOM

Utilizando la librería *useAuthenticator* ofrecida por el paquete de React Dom, crearemos un archivo de nombre *RequireAuth.js* el cuál contendrá una función la cual se encargará de validar si existe una sesión iniciada previamente.

```
src > RequireAuth.js > RequireAuth
1  import { useLocation, Navigate } from 'react-router-dom';
2  import { useAuthenticator } from '@aws-amplify/ui-react';
3
4  export function RequireAuth({ children }) {
5    const location = useLocation();
6    const { route } = useAuthenticator((context) => [context.route]);
7    if (route !== 'authenticated') {
8      return <Navigate to="/login" state={{ from: location }} replace />;
9    }
10   return children;
11 }
```

Figura 3: Función RequireAuth

Posteriormente, se necesita contar con una página de Login, la cuál permitirá validar que se encuentre una sesión iniciada por parte del usuario, para así poder acceder a las rutas protegidas.

```
src > Login.js > ...
1  import { useEffect } from "react";
2  import { Authenticator, useAuthenticator, View } from '@aws-amplify/ui-react';
3  import '@aws-amplify/ui-react/styles.css';
4  import { useNavigate, useLocation } from 'react-router';
5
6  export function Login() {
7    const { route } = useAuthenticator((context) => [context.route]);
8    const location = useLocation();
9    const navigate = useNavigate();
10   let from = location.state?.from?.pathname // '/';
11   useEffect(() => {
12     if (route === 'authenticated') {
13       navigate(from, { replace: true });
14     }
15   }, [route, navigate, from]);
16   return (
17     <View className="auth-wrapper">
18       <Authenticator></Authenticator>
19     </View>
20   );
21 }
```

Figura 4: Componente Login

Para indicar a React, que se desea implementar una ruta protegida, se necesita ir al componente de dicha ruta e ingresar el siguiente código:

```

src > Home.js > Home
1  import { useAuthenticator, Authenticator } from "@aws-amplify/ui-react";
2
3  export default function Home() {
4    const { route } = useAuthenticator((context) => [context.route]);
5    const message =
6      route === 'authenticated' ? 'FIRST PROTECTED ROUTE!' : 'Loading ...';
7    return (
8      <Authenticator>
9        ({ { signOut, user } }) => (
10          <main>
11            <h1>Bienvenido a Home</h1>
12
13            <button onClick={signOut}>Sign out</button>
14          </main>
15        )
16      </Authenticator>
17    );
18
19  }
20

```

Figura 5: Ruta protegida del componente Home

En dicho componente, se hace uso de las librerías *useAuthenticator* y *Authenticator* las cuales son ofrecidas por los servicios de Amazon Amplify. Se tendrá que hacer esto para todos los componentes que deseemos mantener como rutas protegidas.

Finalmente, dentro de nuestro componente **App.js**, crearemos una función que contendrá el directorio de rutas tanto públicas como protegidas:

```

src > App.js > MyRoutes
37
38  export function MyRoutes(){
39    return (
40      <BrowserRouter>
41        <Routes>
42          <Route path="/" element={<Layout />} />
43          <Route index element={<Home />} />
44          <Route
45            path="/conductor"
46            element={
47              <RequireAuth>
48                <Conductor />
49              </RequireAuth>
50            }
51          />
52          <Route
53            path="/incidencia"
54            element={
55              <RequireAuth>
56                <Incidencia />
57              </RequireAuth>
58            }
59          />
60
61          <Route
62            path="/ubicacion"
63            element={

```

Figura 6: Directorio de rutas

Las rutas protegidas, estarán dentro de las etiquetas `<RequireAuth>`/`</RequireAuth>`, mientras que las públicas, irán dentro de las etiquetas `<Route>`/`</Route>`.

1.3. Resultados

Como resultado de todo lo anterior, tendremos una página de Login que utilizando los servicios de AWS Amplify y Cognito, permitirá iniciar sesión así como registrar a nuevos usuarios.

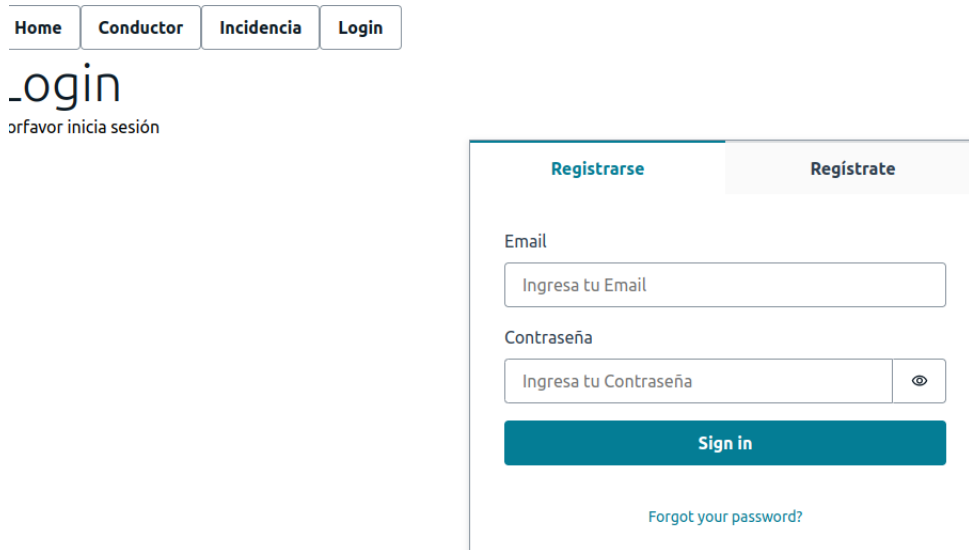


Figura 7: Página de Login

Si intentamos ingresar a las paginas de Conductores, Incidencias o Ubicacion, nos redigirá a la página de Login, debido a que estas páginas fueron definidas como rutas protegidas. Por lo tanto el usuario debe haber iniciado sesión para poder acceder a las mismas.

- **Path:** /
Descripción: En esta dirección se encontrará la el formulario para poder iniciar sesión o registrarse
- **Path:** /home
Descripción: Esta dirección será la página principal de la aplicación dónde se mostrarán las incidencias más recientes así como una lista de todos los conductores
- **Path:** /conductor/
Descripción: Esta dirección mostrará el perfil del conductor de id correspondiente
- **Path:** /detalle_incidencia
Descripción: Esta dirección mostrará cada incidencia mostrando detalles como hora, fecha, coordenadas
- **Path:** /conductor/id/ubicacion
Descripción: En esta vista se mostrará la ubicación en tiempo real de cada conductor
- **Path:** /conductor/id/incidencias
Descripción: En esta vista se mostrará todas las incidencias registradas por cada conductor

2. Investigación de la documentación del módulo 3G/4G LTE-Base Hat

2.1. Objetivo

Familiarizarse con las distintas funciones y comandos, así como el entorno de desarrollo que ofrece el dispositivo Base-Hat

2.2. Descripción

Para el presente proyecto, se hará uso del Base-Hat SIM7600G-H 4G para Jetson Nano.



Figura 8: Instalación Amplify CLI

En primer lugar, utilizando la terminal del sistema operativo Ubuntu, se ingresan los siguientes comandos:

```
sudo apt-get update
sudo apt-get install python3-pip
sudo pip3 install pyserial
mkdir -p ~/Documents/SIM7600X_4G_for_JETSON_NANO
wget -P ~/Documents/SIM7600X_4G_for_JETSON_NANO/ https://www.waveshare.com/w/upload/6/64/SIM7600X_4G_for_JETSON_NANO.tar.gz
cd ~/Documents/SIM7600X_4G_for_JETSON_NANO/
tar -xvf SIM7600X_4G_for_JETSON_NANO.tar.gz
sudo pip3 install Jetson.GPIO
sudo groupadd -f -r gpio
sudo usermod -a -G gpio your_user_name
sudo udevadm control --reload-rules && sudo udevadm trigger
sudo apt-get install minicom
```

Figura 9: Instalación de librerías

Los comandos ingresados en la figura 9 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 10: Instalación de librerías

Utilizando un editor de código, se necesita especificar que estará utilizando el *SDK* de Amazon Amplify:

```
const AWS = require('aws-sdk')
const awsServerlessExpressMiddleware = require('aws-serverless-express/middleware')
const bodyParser = require('body-parser')
const express = require('express')

AWS.config.update({ region: process.env.TABLE_REGION });
```

Figura 11: Implementación del SDK de Amplify

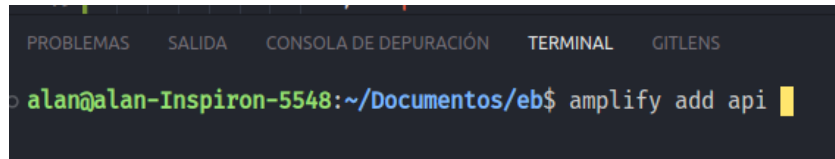
Posteriormente, declaramos una aplicación de ExpressJs la cuál nos permitirá ejecutar entre otras cosas, peticiones HTTP para la comunicación con la base de datos.

```
// declaracion de una app de express
const app = express()
app.use(bodyParser.json())
app.use(awsServerlessExpressMiddleware.eventContext())
```

Figura 12: Creación de aplicación de Express

Finalmente, se necesitan definir las rutas de las APIs que se estarán utilizando en el proyecto, las cuales serán dos, la primera se encargará de la aplicación web, y la segunda de realizar la comunicación con el Módulo Central de Procesamiento.

Para agregar una api, se necesita ejecutar el siguiente comando desde el directorio raíz del proyecto.

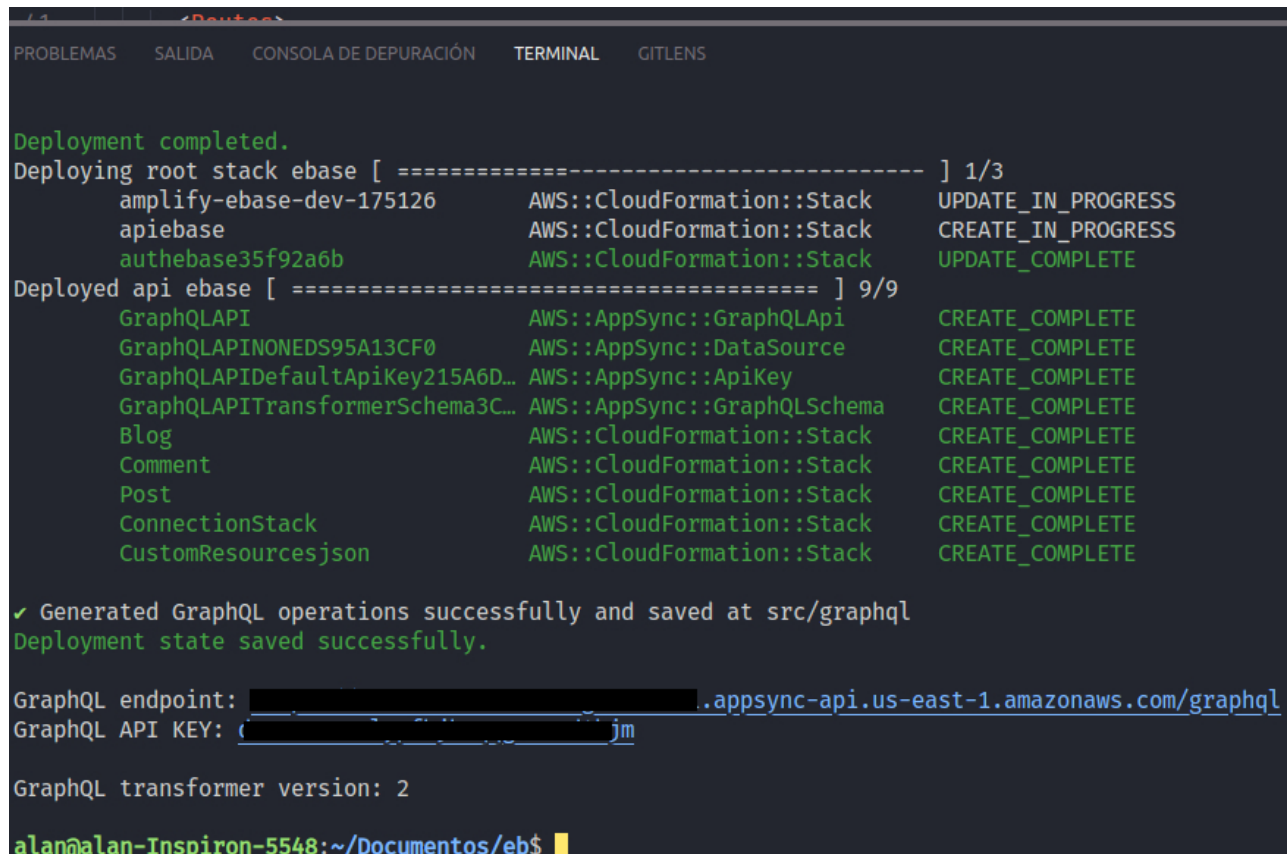


```
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add api
```

Figura 13: Amplify add api

2.3. Resultados

Como resultado, tenemos el endpoint de nuestro modelo de GraphQL y nuestra API Key generada por Amplify



```
Deployment completed.
Deploying root stack ebase [ ===== ] 1/3
  amplify-ebase-dev-175126      AWS::CloudFormation::Stack  UPDATE_IN_PROGRESS
  apiebase                     AWS::CloudFormation::Stack  CREATE_IN_PROGRESS
  authebase35f92a6b            AWS::CloudFormation::Stack  UPDATE_COMPLETE
Deployed api ebase [ ===== ] 9/9
  GraphQLAPI                   AWS::AppSync::GraphQLApi    CREATE_COMPLETE
  GraphQLAPINONEDS95A13CF0     AWS::AppSync::DataSource    CREATE_COMPLETE
  GraphQLAPIDefaultApiKey215A6D... AWS::AppSync::ApiKey        CREATE_COMPLETE
  GraphQLAPITransformerSchema3C... AWS::AppSync::GraphQLSchema CREATE_COMPLETE
  Blog                         AWS::CloudFormation::Stack  CREATE_COMPLETE
  Comment                      AWS::CloudFormation::Stack  CREATE_COMPLETE
  Post                         AWS::CloudFormation::Stack  CREATE_COMPLETE
  ConnectionStack              AWS::CloudFormation::Stack  CREATE_COMPLETE
  CustomResourcesjson          AWS::CloudFormation::Stack  CREATE_COMPLETE

✓ Generated GraphQL operations successfully and saved at src/graphql
Deployment state saved successfully.

GraphQL endpoint: [redacted].appsync-api.us-east-1.amazonaws.com/graphql
GraphQL API KEY: ([redacted])jm

GraphQL transformer version: 2

alan@alan-Inspiron-5548:~/Documentos/eb$
```

Figura 14: Generación de Endpoint de GraphQL

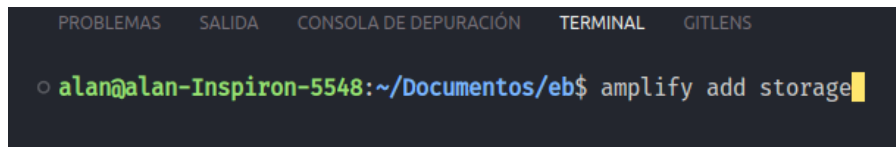
3. Enlace de Amazon S3 con el sistema backend

3.1. Objetivo

Configurar e implementar la comunicación entre el sistema de alojamiento Amazon S3 y el sistema backend

3.2. Descripción

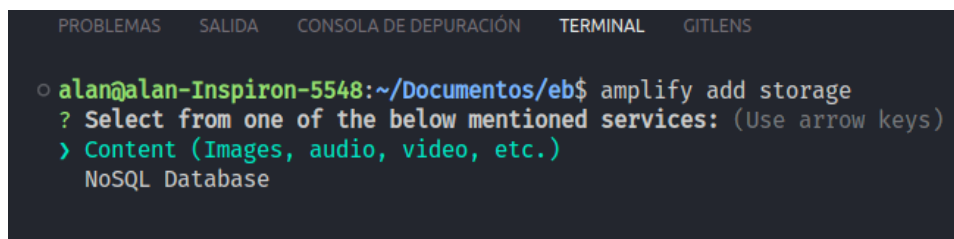
Utilizando un editor de código, y desde el directorio raíz de la aplicación, se deberá introducir el siguiente comando:



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

Figura 15: Configuración de servicio de almacenamiento S3

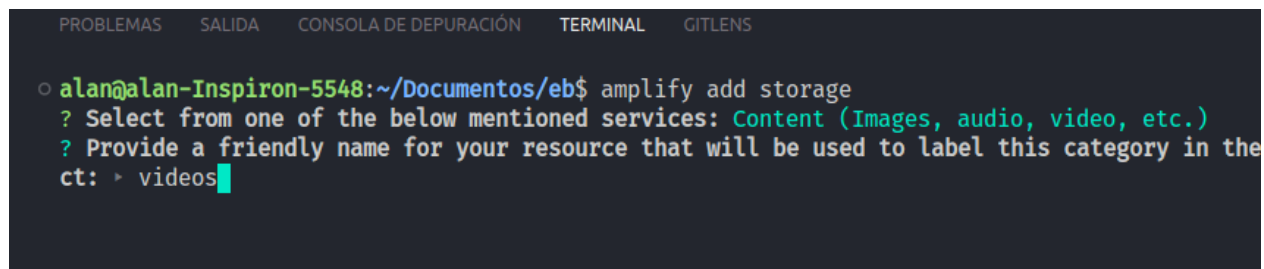
Posteriormente, se requiere especificar que tipo de servicio de almacenamiento se integrará a la aplicación (multimedia o base de datos NoSQL). Para el presente proyecto, se utilizará el almacenamiento de contenido multimedia, por lo tanto, se seleccionará dicha opción.



```
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 16: 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 17: 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 18: 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 19: 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 20: 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 21: 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*.

Buckets (3) Info				
Los buckets son contenedores de datos almacenados en S3. Más información				
	Copiar ARN	Vaciar	Eliminar	Crear bucket
<input type="text" value="Buscar buckets por nombre"/>				
	Nombre	Región de AWS	Acceso	Fecha de creac
<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 22: Tablas generadas mediante los schemas definidos

4. Creación de los servicios backend

4.1. Objetivo

Desarrollar los servicios que constituyen las funciones de la API, tales como obtención, creación y eliminación. Además, crear de los servicios que llevarán a cabo las funciones internas en la plataforma.

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 través 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 23: 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 24: Función listConductors

La función de la figura 24 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 25: Función getIncidencia

La función de la figura 25 obtiene los datos de una sola Incidencia.


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

Figura 26: Función listIncidentes

La función de la figura 26 obtiene los datos de todas las incidencias 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
        conductorIncidentesId
      }
    }
    nextToken
  }
  num_incidentes
  createdAt
  updatedAt
}
```

Figura 27: Función createConductor

La función de la figura 27 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 28: Función updateConductor

La función de la figura 28 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 29: Función deleteConductor

La función de la figura 29 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 30: Función createIncidencia

La función de la figura 30 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 31: Función onCreateIncidencia

La función de la figura 31 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 32: Consola de AWS

Posteriormente se necesita ingresar a la sección de consultas.

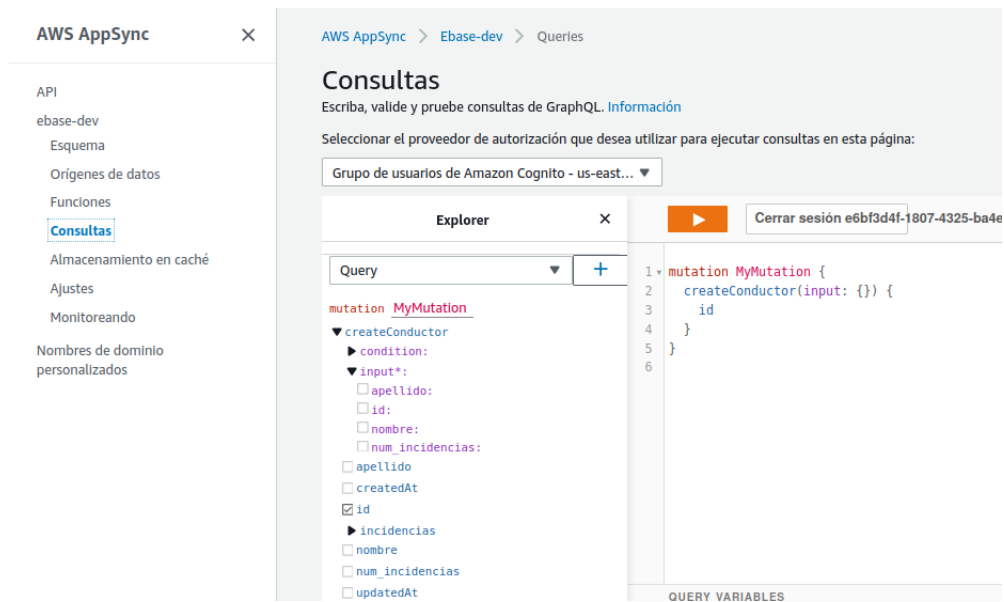


Figura 33: Funcionamiento de Appsync

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

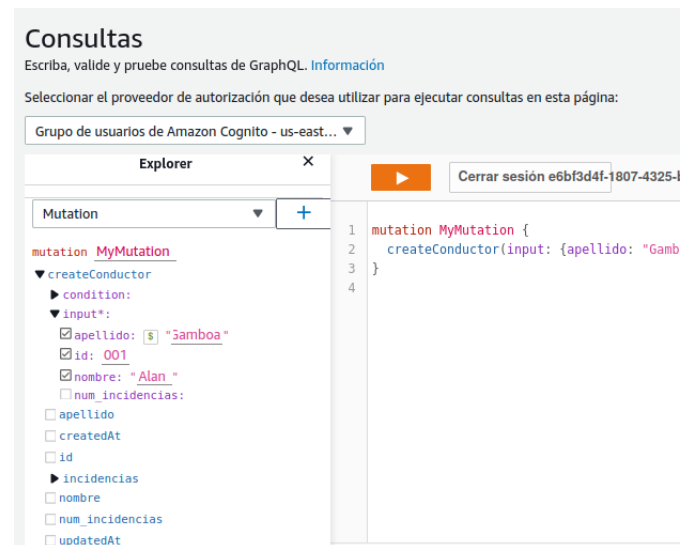


Figura 34: Crear Conductor

En la figura 35 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 35: Resultado

5. Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano

5.1. Objetivo

Investigar la documentación ofrecida por NVIDIA sobre el uso y entorno de desarrollo de 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. [5].

Como primer paso se descargó el *Jetson Nano Developer Kit SD Card Image* [6] 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.

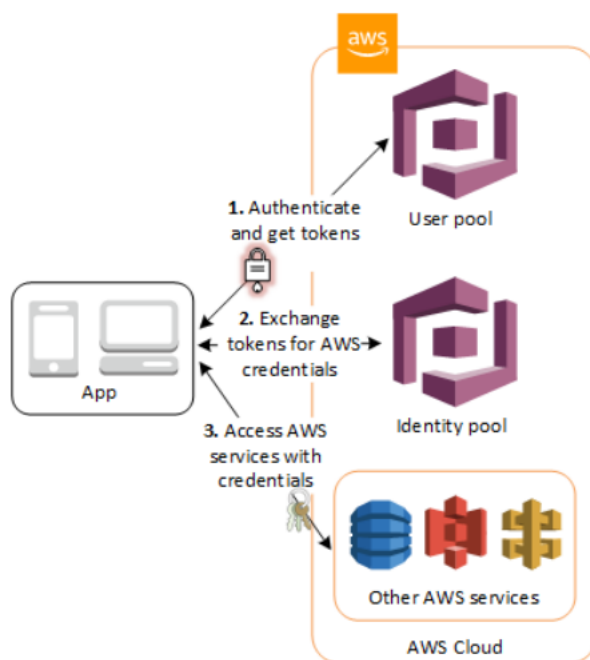


Figura 36: Funcionamiento de Amazon Cognito

5.3. Resultados

6. Implementar algoritmo para la detección del rostro y ojos

6.1. Objetivo

realizar la detección de rostro y ojos en la Jetson Nano .

6.2. Descripción

6.3. Resultados

7. Implementar puntos faciales en el rostro y la metrica MOR

7.1. Objetivo

Implementar la detección con puntos faciales en el rostro y la metrica mor para detectar si la boca se encuentra abierta o cerrada.

7.2. Descripción

7.3. Resultados

8. Conclusiones

9. Bibliografía

Referencias

- [1] F. Martinez. *Protección de rutas con React Router Dom*, DEV Community. <https://dev.to/franklin030601/proteccion-de-rutas-con-react-router-dom-144j> (accedido el 7 de marzo de 2023).
- [2] *¿Qué es Amazon DynamoDB?*, Amazon Docs. <https://docs.aws.amazon.com/es-es/amazondynamodb/latest/developerguide/Introduction.html> (accedido el 2 de marzo de 2023).
- [3] *API sin servidor de GraphQL y de publicación o suscripción – AWS AppSync – Amazon Web Services*. Amazon Web Services, Inc. <https://aws.amazon.com/es/appsync/> (accedido el 12 de marzo de 2023).
- [4] *AWS — Gestión de identidades y autenticación de usuario en la nube*, Amazon Web Services, Inc. <https://aws.amazon.com/es/cognito/> (accedido el 8 de marzo de 2023).
- [5] 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].
- [6] 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].
- [7] 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].