



“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	12
3.1. Objetivo	12
3.2. Descripción	12
3.3. Resultados	12
4. Creación de los servicios backend	13
4.1. Objetivo	13
4.2. Descripción	13
4.3. Resultados	13
5. Familiarizarse con el entorno de desarrollo de la NVIDIA Jetson Nano	15
5.1. Objetivo	15
5.2. Descripción	15
5.3. Resultados	16
6. Implementar algoritmo para la detección del rostro y ojos	17
6.1. Objetivo	17
6.2. Descripción	17
6.3. Resultados	17
7. Implementar puntos faciales en el rostro y la metrica MOR	18
7.1. Objetivo	18
7.2. Descripción	18
7.3. Resultados	18
8. Conclusiones	19
9. Bibliografía	20

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.	Configuración del proyecto	8
10.	Implementación del SDK de Amplify	8
11.	Creación de aplicación de Express	9
12.	Amplify add api	9
13.	Configuración de parámetros de GraphQL	9
14.	Definición de Schemas para el manejo de datos	10
15.	Generación de Endpoint de GraphQL	11
16.	Tablas generadas mediante los schemas definidos	12
17.	Funcionamiento de Appsync	13
18.	Generación de endpoint de GraphQL	14
19.	Funcionamiento de Amazon Cognito	16

Índice de tablas

1. Maquetación web

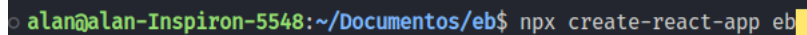
1.1. Objetivo

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

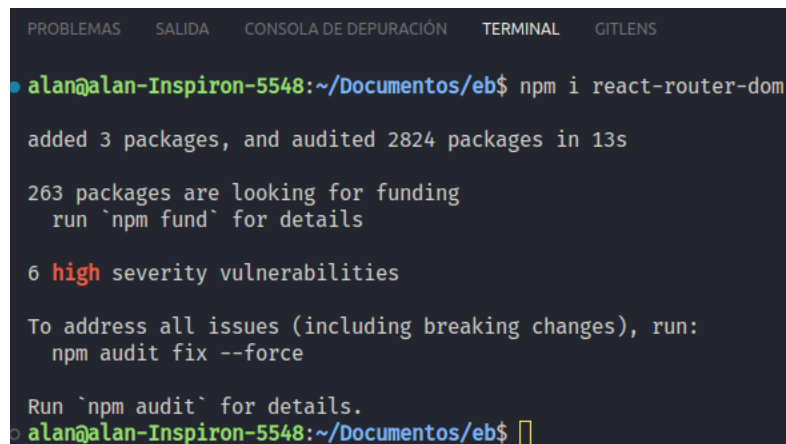
Cuando se habla de una ruta protegida en React, se refiere a programar un bloqueo en ciertas rutas a la cual se le restringe el acceso al usuario. Esto comunmente se realiza para la validación de inicio de sesión de usuarios. Sí el usuario no tiene una sesión iniciada, no podrá acceder a las rutas protegidas 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.

Home Conductor Incidencia Login

Login

o favor inicia sesión

Registrarse Regístrate

Email

Ingresa tu Email

Contraseña

Ingresa tu Contraseña

Sign in

Forgot your password?

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

Crear las rutas mediante las que el cliente realizará las peticiones y tendrá acceso a las operaciones, así como su funcionamiento en cuanto a obtención de datos y comunicación con el resto de la aplicación.

2.2. Descripción

Para poder utilizar los servicios de Amazon Amplify, necesitamos dirigirnos al directorio root de nuestro proyecto y ejecutar el siguiente comando:

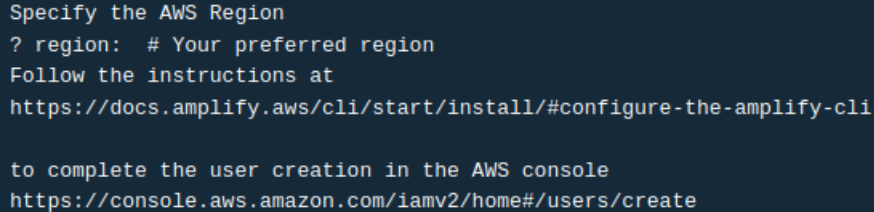


```
npm install -g @aws-amplify/cli
```

[copy](#)

Figura 8: Instalación Amplify CLI

Posteriormente, se necesita especificar la región en la cual queremos alojar nuestra aplicación web:

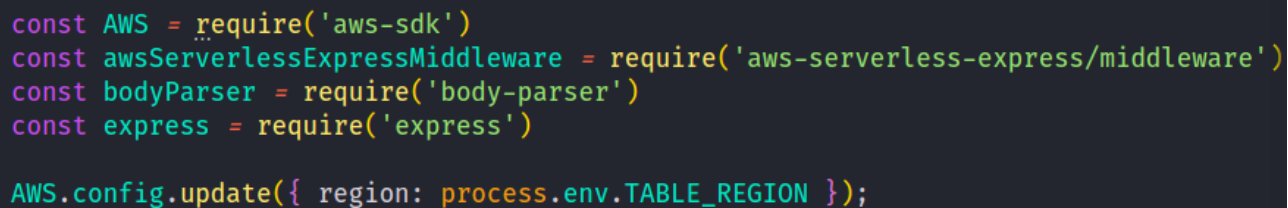


```
Specify the AWS Region
? region: # Your preferred region
Follow the instructions at
https://docs.amplify.aws/cli/start/install/#configure-the-amplify-cli

to complete the user creation in the AWS console
https://console.aws.amazon.com/iamv2/home#/users/create
```

Figura 9: Configuración del proyecto

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 10: 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 11: 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.

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

Figura 12: Amplify add api

La consola de AWS Amplify requerirá introducir parámetros con los cuales serán construida nuestra API, para el presente proyecto se estará utilizando una arquitectura mediante GraphQL, por lo cual seleccionaremos dicha opción.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  GITLENS
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify add api
? Select from one of the below mentioned services: GraphQL
? Here is the GraphQL API that we will create. Select a setting to edit or continue Continue
? Choose a schema template: One-to-many relationship (e.g., "Blogs" with "Posts" and "Comments")

⚠ WARNING: your GraphQL API currently allows public create, read, update, and delete access to all models via a
n API Key. To configure PRODUCTION-READY authorization rules, review: https://docs.amplify.aws/cli/graphql/autho
rization-rules

✔ GraphQL schema compiled successfully.

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
✔ Do you want to edit the schema now? (Y/n) - yes
```

Figura 13: Configuración de parámetros de GraphQL

Antes de poder publicar nuestra API en AWS Amplify, se necesita definir el Schema con el cuál se hará el manejo de datos. A contiución se muestran dichos schemas:

```
amplify > backend > api > ebase > ✨ schema.graphql
1  # This "input" configures a global authorization rule to enable
2  # all models in this schema. Learn more about authorization rule
3  input AMPLIFY { globalAuthRule: AuthRule = { allow: public } } #
4
5  type Conductor @model {
6    id: ID
7    nombre: String
8    apellido: String
9    incidencias: [Incidencia] @hasMany
10   num_incidencias: Int
11 }
12
13
14 type Incidencia @model {
15   id: ID
16   title: String
17   estado: Boolean
18   conductor: Conductor @belongsTo
19   detalles: [Detalles] @hasOne
20   fecha_hora: Date
21 }
22
23 type Detalles @model {
24   id: ID
25   incidencia: Incidencia @belongsTo
26   ubicacion: String
27   url_video: String
28 }
29
```

Figura 14: Definición de Schemas para el manejo de datos

Finalmente, ejecutaremos el comando *amplify push*, el cuál publicará la API hacia AWS amplify, generando el endpoint correspondiente a dicha 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])

GraphQL transformer version: 2

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

Figura 15: Generación de Endpoint de GraphQL

3. Enlace de Amazon S3 con el sistema backend

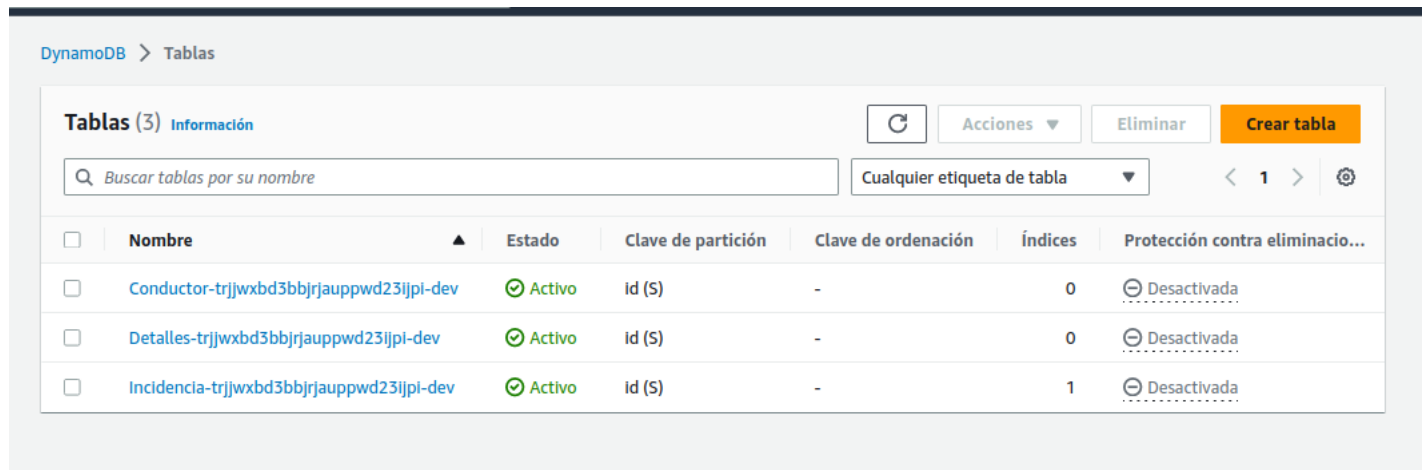
3.1. Objetivo

Crear la aplicación web a desarrollar e implementar el sistema de diseño de manera local

3.2. Descripción

3.3. Resultados

Una vez dentro, ingresamos a la sección del servicio de DynamoDB



The screenshot shows the Amazon DynamoDB console interface. At the top, there's a breadcrumb 'DynamoDB > Tablas'. Below it, a header bar contains 'Tablas (3) Información', a refresh button, an 'Acciones' dropdown, an 'Eliminar' button, and a 'Crear tabla' button. A search bar with the placeholder 'Buscar tablas por su nombre' and a dropdown for 'Cualquier etiqueta de tabla' are also present. The main table lists three tables, each with a checkbox, name, status, partition key, sort key, number of indices, and deletion protection status.

<input type="checkbox"/>	Nombre	Estado	Clave de partición	Clave de ordenación	Índices	Protección contra eliminacio...
<input type="checkbox"/>	Conductor-trjjwxbd3bbjrjauppwd23ijpi-dev	Activo	Id (S)	-	0	Desactivada
<input type="checkbox"/>	Detalles-trjjwxbd3bbjrjauppwd23ijpi-dev	Activo	Id (S)	-	0	Desactivada
<input type="checkbox"/>	Incidencia-trjjwxbd3bbjrjauppwd23ijpi-dev	Activo	Id (S)	-	1	Desactivada

Figura 16: Tablas generadas mediante los schemas definidos

Como se puede observar, gracias a los pasos realizados en la sección ??, DynamoDB crea automáticamente las tablas creadas en base a los schemas definidos previamente.

4. Creación de los servicios backend

4.1. Objetivo

Realizar la conexión de NodeJs con la base de datos MongoDB.

4.2. Descripción

Para realizar la conexión y la integración de los servicios de DynamoDB hacia nuestra API, se hará uso de AppSync. Las API de GraphQL creadas con AWS AppSync brindan a los desarrolladores frontend la capacidad de consultar varias bases de datos, microservicios y API desde un único punto de conexión de GraphQL.

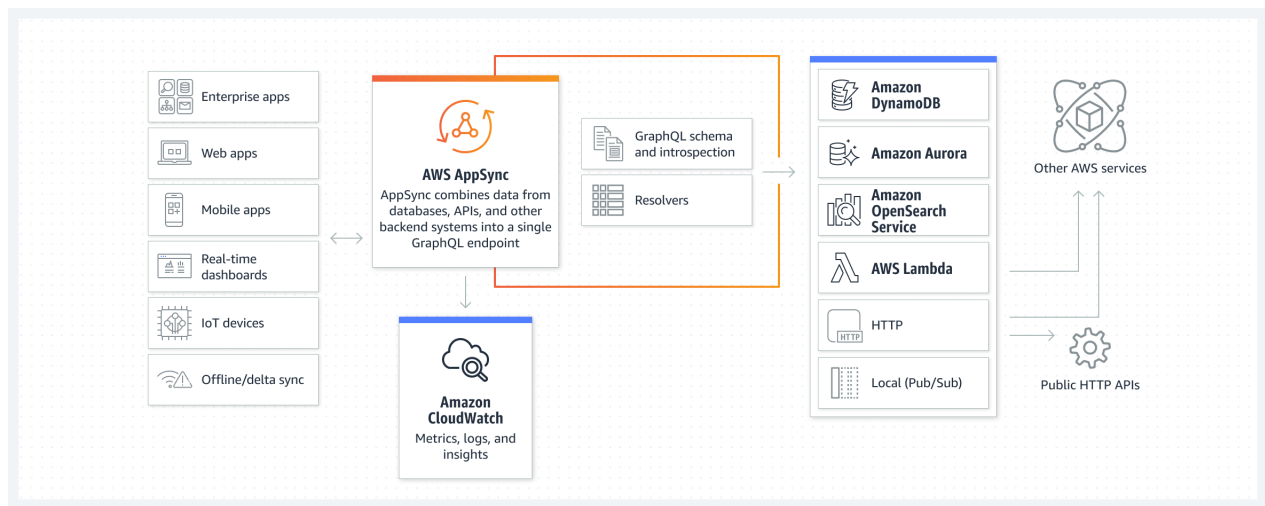


Figura 17: Funcionamiento de Appsync

AWS AppSync crea las API sin servidor de GraphQL y de publicación o suscripción que simplifican el desarrollo de aplicaciones a través de un único punto de conexión para consultar, actualizar o publicar datos.

4.3. Resultados

Después de haber realizado los pasos de la sección ??, si ejecutamos el comando *amplify status*, la consola de Amplify nos retornara el endpoint de GraphQL junto con AppSync correspondiente, el cuál se utilizará para realizar todas las operaciones con respecto al almacenamiento de datos

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

Current Environment: dev



| Category | Resource name | Operation | Provider plugin   |
|----------|---------------|-----------|-------------------|
| Auth     | ebase35f92a6b | No Change | awscloudformation |
| Api      | ebase         | No Change | awscloudformation |



GraphQL endpoint: https://ckqxuivcobc4rgh72skzudaixy.appsync-a
GraphQL transformer version: 2

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

Figura 18: Generación de endpoint de GraphQL

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* [?] 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.

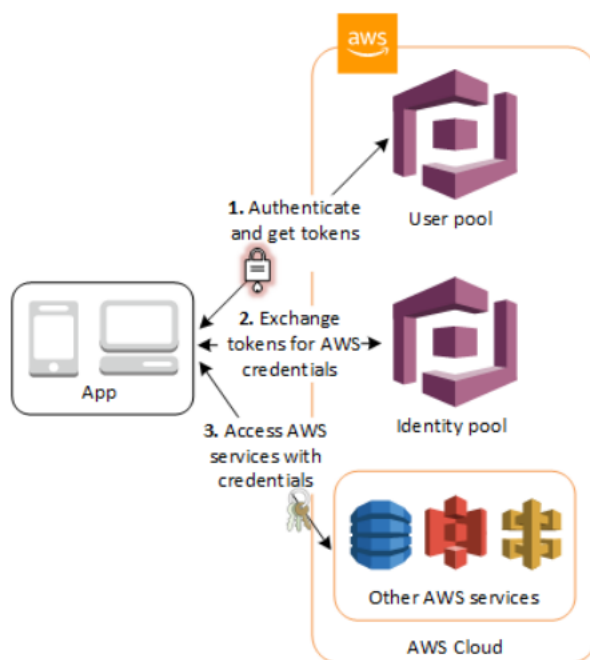


Figura 19: Funcionamiento de Amazon Cognito

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].