



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

Primer Reporte Parcial

Lista de actividades

- Definir rutas del frontend
- Diseño de rutas del backend
- Conexión Backend con Mongo DB
- Sistema de acceso con credenciales
- Creación de la base de datos no relacional
- Investigación de modelos de Redes Neuronales Convolucionales
- Diseño de una red neuronal convolucional capaz de detectar ojos cerrados y abiertos

Autores:

Alan Eduardo Gamboa Del
Ángel
Maite Paulette Díaz Martínez

Asesores:

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

Índice

1. Definir rutas del frontend	4
1.1. Objetivo	4
1.2. Descripción	4
1.3. Resultados	7
2. Definir rutas del backend	8
2.1. Objetivo	8
2.2. Descripción	8
2.3. Resultados	10
3. Conexión Backend con Mongo DB	12
3.1. Objetivo	12
3.2. Descripción	12
3.3. Resultados	12
4. Sistema de acceso con credenciales	14
4.1. Objetivo	14
4.2. Descripción	14
4.3. Resultados	16
5. Creación de la base de datos No Relacional	18
5.1. Objetivo	19
5.2. Descripción	19
5.3. Resultados	19
6. Investigación de modelos de Redes Neuronales Convolucionales	20
6.1. Objetivo	20
6.2. Descripción	20
6.3. Resultados	20
7. Diseño de una red neuronal convolucional capaz de detectar ojos cerrados y abiertos	21
7.1. Objetivo	21
7.2. Descripción	21
7.3. Resultados	21
8. Conclusiones	22
9. Bibliografía	23

Índice de figuras

1.	Creación proyecto de react	4
2.	Creación proyecto de react	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.	Página web MongoDB Atlas.	8
9.	Página web MongoDB Atlas.	8
10.	Página web MongoDB Atlas.	8
11.	Página web MongoDB Atlas.	9
12.	Página web MongoDB Atlas.	9
13.	Página web MongoDB Atlas.	9
14.	Página web MongoDB Atlas.	10
15.	Página web MongoDB Atlas.	11
16.	Página web MongoDB Atlas.	12
17.	Página web MongoDB Atlas.	13
18.	Directorio del Backend	14
19.	Directorio del Backend	15
20.	Directorio del Backend	15
21.	Directorio del Backend	16
22.	Grupos de usuario	16
23.	Grupos de usuario	17
24.	Grupos de usuario	17
25.	Directorio del Backend	18
26.	Directorio del Backend	19
27.	Directorio del Backend	19

Índice de tablas

1. Definir rutas del frontend

1.1. Objetivo

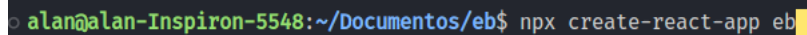
Definir e implementar las rutas que tendrá la aplicación, así como si serán públicas o privadas y la información que se desplegará en cada una de las mismas.

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.

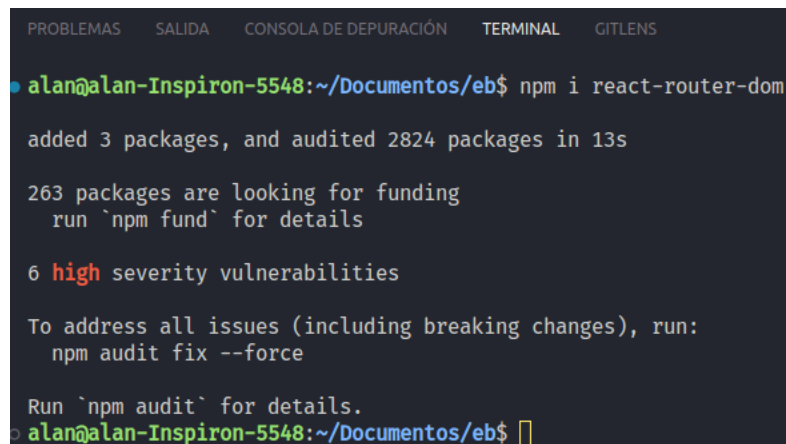
Como primer paso, se necesita crear un proyecto de React utilizando el siguiente comando:



```
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: Creación proyecto de react

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.

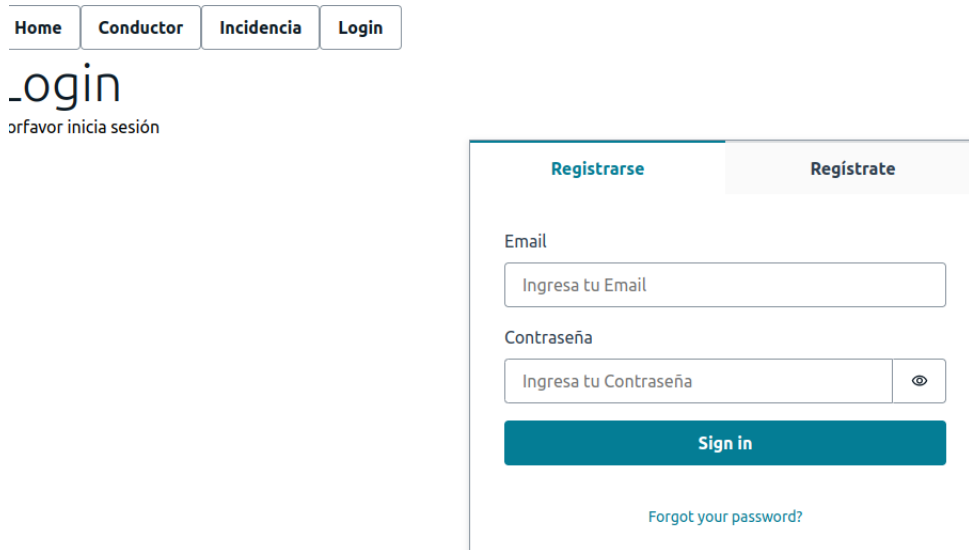


Figura 7: Página de Login

Si intentamos ingresar a las páginas de Conductores, Incidencias o Ubicación, nos redirigirá 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 el formulario para poder iniciar sesión o registrarse
- **Path:** /home
Descripción: Esta dirección será la página principal de la aplicación donde 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. Definir rutas del backend

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: Página web MongoDB Atlas.

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: Página web MongoDB Atlas.

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: Página web MongoDB Atlas.

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: Página web MongoDB Atlas.

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: Página web MongoDB Atlas.

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: Página web MongoDB Atlas.

Antes de poder publicar nuestra API en AWS Amplify, se necesita definir el Schema con el cuál se hará el manejo de datos. A contiunción 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: Página web MongoDB Atlas.

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])jm

GraphQL transformer version: 2

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

Figura 15: Página web MongoDB Atlas.

3. Conexión Backend con Mongo DB

3.1. Objetivo

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

3.2. 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 16: Página web MongoDB Atlas.

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.

3.3. Resultados

Después de haber realizado los pasos de la sección 2, 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 17: Página web MongoDB Atlas.

4. Sistema de acceso con credenciales

4.1. Objetivo

Establecer los roles de cada tipo de usuario con sus respectivos permisos de acceso a la aplicación web utilizando Amazon Cognito

4.2. Descripción

Amazon Cognito funciona utilizando *pools* de usuarios. Un pool de usuarios es un directorio almacenado en los servicios de Amazon. Los beneficios que ofrece estar registrado en un pool de usuarios de Amazon Cognito son los siguientes:

- Servicio de registro e inicio de sesión
- Gestión del directorio de usuarios
- Servicios de seguridad tales como verificación de dos pasos
- Acceso a servicios de la suite de AWS tales como S3 o Dynamodb

Funcionamiento

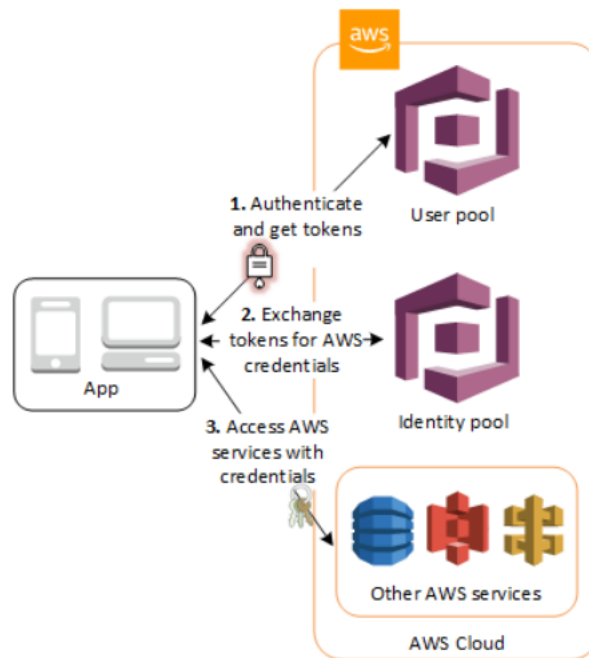


Figura 18: Directorio del Backend

- Como primer paso el usuario inicia sesión a través de un grupo de usuarios y recibe tokens del grupo de usuarios después de una autenticación exitosa.
- Posteriormente la aplicación intercambia los tokens del grupo de usuarios por las credenciales de AWS a través de un grupo de identidades.

- Finalmente, el usuario puede usar esas credenciales de AWS para acceder a otros servicios de AWS, como Amazon S3 o DynamoDB.

Implementación

Para poder hacer uso de Amazon Cognito en nuestra aplicación debemos de introducir el siguiente comando:

```
amplify add auth
```

[copy](#)

Figura 19: Directorio del Backend

Obteniendo el siguiente menú:

```
alan@alan-Inspiron-5548:~/Documentos/eb$ amplify update auth
Please note that certain attributes may not be overwritten if you choose to use defaults settings.

You have configured resources that might depend on this Cognito resource. Updating this Cognito resource could
have unintended side effects.

Using service: Cognito, provided by: awscloudformation
What do you want to do? Walkthrough all the auth configurations
Select the authentication/authorization services that you want to use: (Use arrow keys)
> User Sign-Up, Sign-In, connected with AWS IAM controls (Enables per-user Storage features for images or other
content, Analytics, and more)
  User Sign-Up & Sign-In only (Best used with a cloud API only)
  I want to learn more.
```

Figura 20: Directorio del Backend

Dejamos seleccionado la primera opción, la cual nos permitirá utilizar los servicios de autenticación ofrecidos por Amazon Cognito, además de otros servicios de la suite de AWS.

Finalmente utilizamos el comando *amplify push* para desplegar los cambios a nuestra aplicación.

4.3. Resultados



Figura 21: Directorio del Backend

Entrando a nuestra consola de AWS, en la sección de Amazon Cognito, se puede observar dos grupos de usuarios, uno de tipo Administrador, el cual puede realizar cambios a la configuración de la aplicación, y otro de tipo de Usuario, el cual sólo puede hacer uso del sistema de inicio de sesión y registro ofrecido por Cognito.

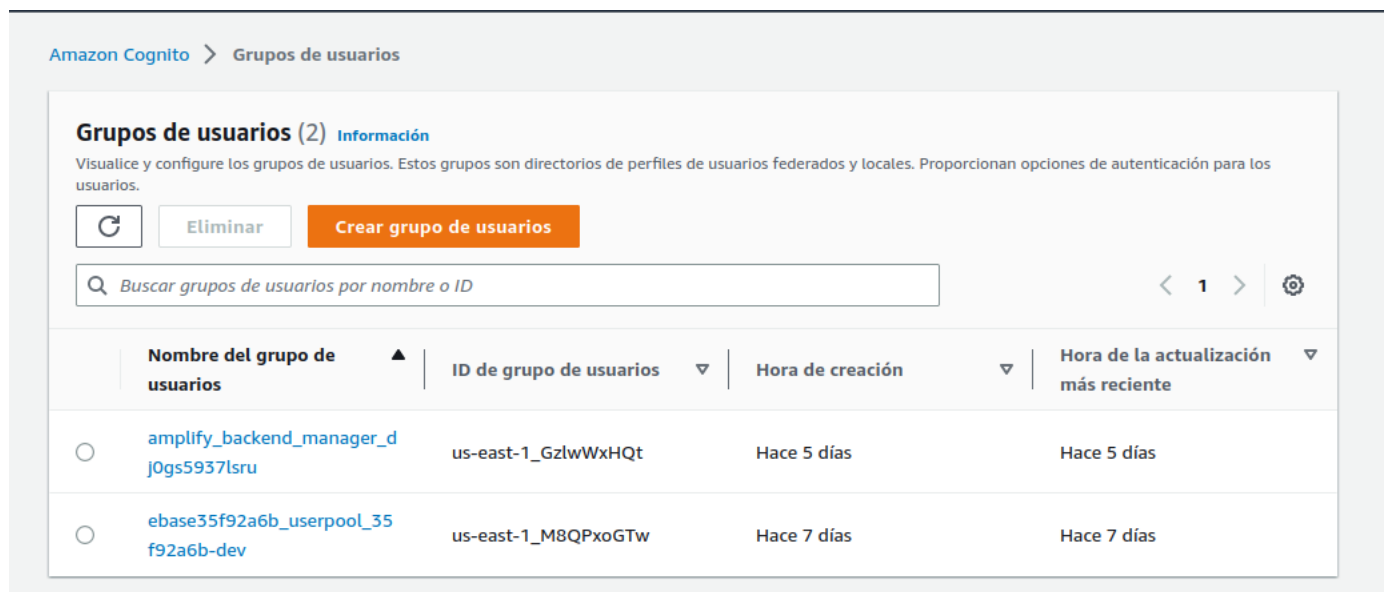


Figura 22: Grupos de usuario

Información general sobre el grupo de usuarios

Nombre del grupo de usuarios
amplify_backend_manager_dj0gs5937lsru

ID de grupo de usuarios
us-east-1_GzlwWxHQ

ARN
arn:aws:cognito-idp:us-east-1:387678068467:userpool/us-east-1_GzlwWxHQ

Número estimado de usuarios
1

Hora de creación
2 de marzo de 2023, 13:46 GMT-6

Hora de la actualización más reciente
2 de marzo de 2023, 13:46 GMT-6

Introducción

Usuarios

Grupos

Experiencia de inicio de sesión

Experiencia de inscripción

Mensajería

Integración de aplicaciones

Propiedades del grupo de usuarios

Usuarios (1) Información

Vea, edite y cree usuarios en el grupo de usuarios. Los usuarios habilitados y confirmados podrán iniciar sesión en el grupo de usuarios.

Nombre de usuario

Buscar usuarios por atributo

< 1 >

Nombre de usuario

Dirección de correo...

Correo electrónico ...

Estado de la confirmación

Estado

☐

aws-amplify-admin

-

No

Confirmado

Habilitado

Figura 23: Grupos de usuario

<

Usuarios

Grupos

Experiencia de inicio de sesión

Experiencia de inscripción

Mensajería

Integración de aplicaciones

Propied...

>

Usuarios (5) Información

Vea, edite y cree usuarios en el grupo de usuarios. Los usuarios habilitados y confirmados podrán iniciar sesión en el grupo de usuarios.

Eliminar usuario

Crear usuario

Nombre de usuario

Buscar usuarios por atributo

< 1 >

Nombre de usuario

Dirección de correo elec...

Correo electrónico verifi...

Estado de la confirmación

Estado

☐

19bd77dc-441b-4d3e-8...

maite.diazm98@gmail.com

Sí

Confirmado

Habilitado

☐

473bad4c-b959-45e4-8...

alangam97@gmail.com

No

No confirmado

Habilitado

☐

849ca9f3-6fd3-4750-a8...

mayte_dm@hotmail.com

No

No confirmado

Habilitado

☐

d1df2a48-8374-4fd2-be...

mayte_dm@hotmail.es

No

No confirmado

Habilitado

☐

e6bf3d4f-1807-4325-ba...

alangam97@gmail.com

Sí

Confirmado

Habilitado

Figura 24: Grupos de usuario

Proyecto Terminal 2

17

5. Creación de la base de datos No Relacional

Debido a problemas de integración junto con los servicios de autenticación y de despliegue de Amplify, se decidió utilizar el sistema de gestión de bases de datos DynamoDB.

DynamoDB es un servicio de base de datos NoSQL Ofrecido por Amazon Web Services. DynamoDB trabaja con tablas. Estas a su vez, contienen parámetros importantes que se mencionarán a continuación.

- **Primary Key:** Se trata de una clave primaria simple, compuesta por un solo atributo denominado clave de partición. Una clave primaria puede ser una clave de partición o una combinación de clave de partición y clave de ordenación. La clave primaria debe ser única en toda la tabla.
 - **Partition Key:** Es la llave principal por la cual se agruparán los datos, y determina cómo se particiona la información.
 - **Sort Key:** Es llave de ordenamiento de los datos.

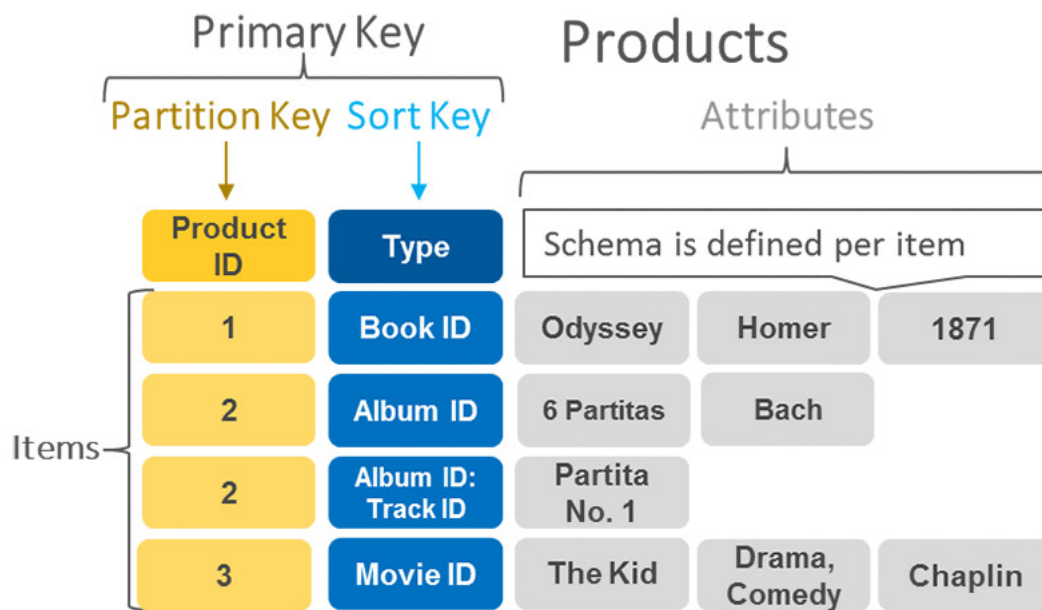


Figura 25: Directorio del Backend

DynamoDB almacena los datos como grupos de atributos, conocidos como elementos. Los elementos son similares a las filas o registros de otros sistemas de bases de datos. DynamoDB almacena y recupera cada elemento en función del valor de la clave principal, que debe ser único.

DynamoDB utiliza el valor de la clave de partición como parámetro de entrada para una función hash interna. El resultado de la función hash determina la partición en la que se almacena el elemento. La ubicación de cada elemento viene determinada por el valor hash de su clave de partición.

Todos los elementos con la misma clave de partición se almacenan juntos y, para las claves de partición compuestas, se ordenan por el valor de la clave de ordenación. DynamoDB divide las particiones por clave de ordenación si el tamaño de la colección crece más de 10 GB.

5.1. Objetivo

Crear la base de datos en MongoDB.

5.2. Descripción

Para crear nuestras tablas de DynamoDB, se debe ingresar a la consola de AWS.



Figura 26: Directorio del Backend

5.3. Resultados

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

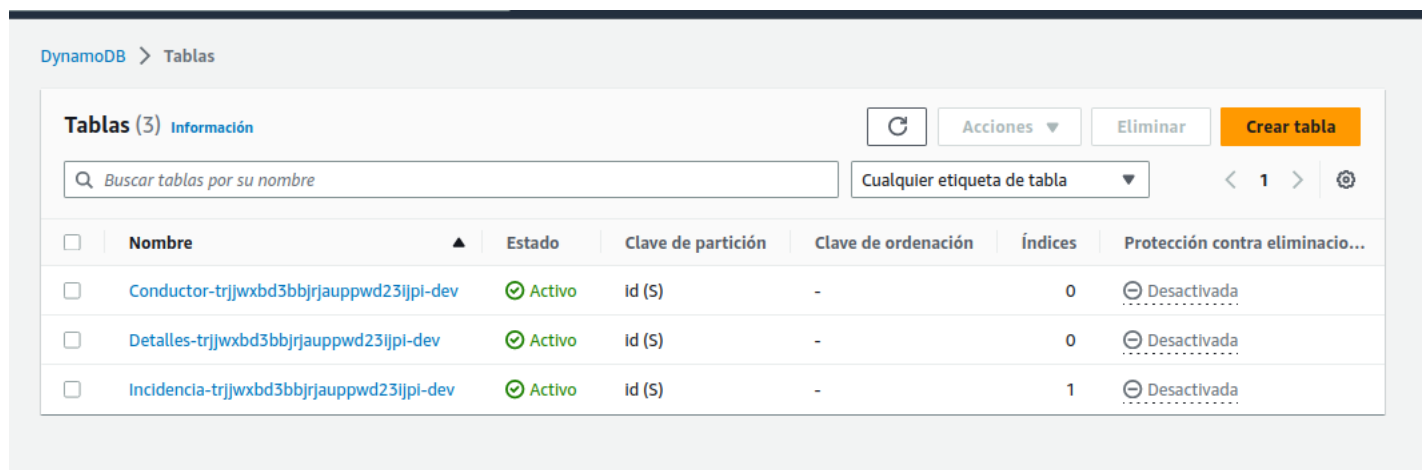


Figura 27: Directorio del Backend

Como se puede observar, gracias a los pasos realizados en la sección 2, DynamoDB crea automáticamente las tablas creadas en base a los schemas definidos previamente.

6. Investigación de modelos de Redes Neuronales Convolutionales

6.1. Objetivo

Determinar distintos modelos de redes neuronales convolucionales que ofrezcan mejor eficiencia al clasificar imágenes.

6.2. Descripción

6.3. Resultados

7. Diseño de una red neuronal convolucional capaz de detectar ojos cerrados y abiertos

7.1. Objetivo

Diseñar y realizar pruebas de los modelos de redes neuronales convolucionales previamente investigados para determinar el rendimiento y la precisión de cada uno.

7.2. Descripción

7.3. Resultados

8. Conclusiones

Para el desarrollo de la Estación base, se decidió utilizar la suite de herramientas de Amazon Amplify. Se hará uso de Amplify Hosting, el cuál tiene una integración directa con Github, esto quiere decir, que los cambios que se realicen en el repositorio se reflejarán de manera automática en la aplicación web. Además, AWS Amplify ofrece su servicio de almacenamiento en la nube S3, este será de gran ayuda para almacenar contenido multimedia, en este caso los videos de incidencia de los conductores. Para el manejo de credenciales, se utilizará Amazon Cognito, que se encargará de administrar las credenciales de acceso a la aplicación. Para el manejo de datos, se estará utilizando MongoDB, un manejador NoSQL que trabaja con documentos. Para el desarrollo de la aplicación, se decidió utilizar el lenguaje de programación Javascript, junto con NodeJs que nos ayudará a manejar varias peticiones al mismo tiempo. Finalmente el análisis del sistema de comunicaciones, en un principio sólo se había contemplado el análisis de telemetría pero al ir realizando las actividades del tercer reporte e ir profundizando en algunos temas de comunicaciones, se decidió que también que se necesitaba el análisis de la cobertura y de los datos, es decir, de la transmisión de los fotogramas por lo que se incluyeron también en este reporte.

9. Bibliografia

Referencias