PRÁCTICA NO GUIADA: ART WITH DRONES

SISTEMAS DISTRIBUIDOS 2023-2024

Participantes

Judit Serrano Espinosa , 74379872B

Grupo 7

INTRODUCCIÓN

El objetivo principal de esta práctica es desarrollar una simulación de creación de figuras mediante drones. Todo esto se llevará a cabo en un entorno distribuido, donde los drones de manera autónoma se sincronizarán a tiempo real.

Para poder dar solución a este problema, se implementan cinco programas. AD_Engine el motor que se encarga de coordinar todo el espectáculo. AD_Registry un servidor que se encarga de registrar, dar de baja y editar los drones en la base de datos, devolviendo así un token a cada uno para poder identificarse con el Engine. AD_Drone el cliente que puede registrar el dron y unirse al espectáculo, además de calcular el movimiento de manera automática. API_Engine que se encarga de consultar el estado de los drones y obtener el mapa, además de consultar la auditoría. (Esta api se utilizará para el front). Por último, un front-end que permitirá a los usuarios ver el mapa y el movimiento de los drones.

AD_Weather es un servidor que informa al Engine de que la temperatura es adecuada para que se realice el espectáculo, en caso contrario, se cancelará. No obstante, se ha prescindido de él y se ha utilizado una api para obtener la temperatura. Dentro del json api_key se especifica la clave para acceder a la api y la ciudad donde quiere consultar la temperatura.

IMPLEMENTACIÓN

Todo lo anteriormente mencionado se ha implementado en Nodejs. Además, como base de datos se ha usado un fichero json para guardar los datos de los drones (drones_DB), otro para guardar las auditorias llamado (auditorias.json) y un fichero de texto para consultar la temperatura. Ya que, según su funcionamiento no se ha requerido de algo más complejo.

Sumando a eso kafka que consta de tres topics (new_position, mapa y posiciones).

IMPLEMENTACIÓN DE LA PRÁCTICA 1

Mi compañero y yo decidimos hacer la práctica por nuestra cuenta individualmente, no obstante, se ha cogido código que realizamos juntos. Por lo que,

Iván ha realizó la implementación de:

1. Registry-Dron

El programa se inicia con el proceso de registro de los drones destinados para el espectáculo a través del despliegue de los módulos AD_Drone y AD_Registry mediante una comunicación por sockets. La selección de la acción que desea realizar se lleva a cabo a través de un menú de opciones, que incluye el registro, la baja o la actualización de los datos de un dron.

En el proceso de **registro**, el módulo Registry recibe el id y el alias del dron desde el módulo Drone, realizando una serie de validaciones, como verificar la validez de los datos y asegurarse de que no haya campos vacíos. Si el dron no se encuentra previamente registrado en la base de datos, se genera un token aleatorio para dicho dron y se almacena su información en la base de datos. En caso de que el dron ya esté registrado, se muestra un mensaje de error y se retorna al menú de selección.

En el caso de dar de **baja** a un dron, el módulo Registry solicita al módulo Drone el ID del dron que se desea eliminar. Si el dron está registrado en la base de datos, se eliminará su información. En caso contrario, se muestra un mensaje de error y se regresa al menú principal.

Para la **actualización** de datos, el módulo Registry recibe el id del dron que se desea actualizar. Si el dron se encuentra en la base de datos, se solicita al usuario ingresar el nuevo id y el nuevo alias, y estos cambios se reflejan en la base de datos. Si el dron no está registrado, se muestra un mensaje de error y se retorna al menú principal.

Una vez realizada cualquiera de las anteriores acciones termina la conexión del Drone con el Registry.

2. Parte de Dron-kafka
3. Parte de Engine-Dron

Yo realicé la implementación de:

1. Weather-Engine
    a. En primer lugar, se ejecuta AD_Weather indicando el puerto de escucha. Este se mantendrá escuchando hasta que reciba una petición.

```
server.listen(process.argv[2],()=>{
    console.log(`Servidor de clima escuchando desde el puerto ${process.argv[2]}...`
})
```

Una vez reciba conexión, AD_Weather esperará a que el cliente le envíe el evento 'ciudad'. Cuando esto ocurra, el servidor consultará el fichero ciudades.txt, que contiene un número con la temperatura que hace, y se la devolverá al cliente.

```
io.on('connection',(socket)=>{
    console.log('Usuario conectado');
    socket.on('cuidad',()=>{
        fs.readFile('ciudades.txt','utf-8',(err,data)=>{
            if(err){
                console.log('error: ',err);
                return;
            }else{
                socket.emit('temperatura',data)
            }
        })
    })
})
```

    b. El AD_Engine al ejecutarse establece conexión AD_Weather. Este consultará cada 5 segundos la temperatura y no cerrará la conexión a no ser que las temperaturas sean adversas.

```
socket.on('connect',()=>{
        console.log('Conectado con servidor clima');
        socket.emit('cuidad');
})


const idInterval = setInterval(()=>{
        socket.emit('cuidad');
    },5000);
```

Es decir, que la temperatura sea 0 o menor.

```
socket.on('temperatura',(temp)=>{
    if(temp<=0){
        clearInterval(idInterval);
        console.log('CONDICIONES CLIMATICAS ADVERSAS. ESPECTACULO FINALIZADO')
        estado = "CONDICIONES CLIMATICAS ADVERSAS. ESPECTACULO FINALIZADO"
        socket.disconnect();
    }
})
```

En caso de que esto ocurra informará a los drones enviando mediante el topic mapa de kafka que las condiciones no son adecuadas para proseguir con el espectáculo.

```
if(estado!="CONDICIONES CLIMATICAS ADVERSAS. ESPECTACULO FINALIZADO"){
    await producer.send({
        topic:'mapa',
        messages:[
            {value:estado}
        ]
    })
    producer.disconnect()
}
```

2. Engine-kafka

   La implementación de kafka consta de dos partes. Una que lee la información de AwD_figuras.json donde guarda cada figura y le envía la posición destino a cada dron registrado. Otra lee las posiciones que le envían los drones, los mueve en el tablero y se lo envía al dron.

En primer lugar, definimos kafka.

```
kafka
const kafka = new Kafka({
    clientId: 'ad_engine',
    brokers: [process.argv[4]+':'+process.argv[5]]
})
```

Y los productores y los consumidores.

```
const producer = kafka.producer();
const consumer = kafka.consumer({groupId: 'ad_engine'});
```

a. Para enviarle las posiciones a las que deben moverse los drones definimos new_position.

```
const new_position = async () => {
    await producer.connect();
    await leerfigura();
```

Este llama a leerfigura() donde lo primero que hace es comprobar si el tablero está a cero, si lo está no hará nada. Si no lo está, lo primero que hará será comprobar si la figura está completa, comprobando si algún dron del tablero tiene arrived a falso.

Si está completa leerá AwD_figuras.json y quitará la primera figura.

A continuación, volverá a leer AwD_figuras.json y sacará la primera figura.

```
const figuras = await fs.promises.readFile('AwD_figuras.json','utf-8');
const figurasJSON = JSON.parse(figuras);
const figura = figurasJSON.figuras[0];
```

iterará el tablero leyendo todos los drones registrados y si el dron se encuentra en la figura le enviará la posición en forma de un string cuyo primer número es el id y los dos siguientes son las coordenadas(mediante el topic new_position). Además de añadir la posición actual que será 1,1 y la posición destino al tablero.

```
for(let i=0;i<tablero.drones.length;i++){
    const dron = figura.Drones.find(dron => dron.ID == tablero.drones[i].id)
    if(dron){
        if("posicion_destino" in tablero.drones[i]){
            if(tablero.drones[i].posicion_destino.join(',') != dron.POS){
                tablero.drones[i].posicion_destino = dron.POS.split(',')
            }
            else{
                return;
            }
        }else{
            tablero.drones[i].posicion_destino = dron.POS.split(',')
            tablero.drones[i].posicion_actual = [1,1]
        }
        if(llegado(tablero.drones[i].posicion_actual, tablero.drones[i].posicion_destino)){
            tablero.drones[i].arrived = true
        }
        const dronI = tablero.drones[i]
        await producer.send({
            topic: 'new_position',
            messages: [
                {value: ''+ dronI.id.toString() +' ' + dronI.posicion_destino.join(',') + ''}
            ]})
```

Además de eso, guardará la posición actual y la destino en la base de datos a la que llamamos drones_DB.json.

```
const drones = await fs.promises.readFile('drones_DB.json','utf-8');
const dronesJSON = JSON.parse(drones);
const dron_to_edit = dronesJSON.drones.find(dron => dron.ID == dronI.id)
dron_to_edit.posicion_actual = dronI.posicion_actual
dron_to_edit.posicion_destino = dronI.posicion_destino
const added = await fs.promises.writeFile('drones_DB.json',JSON.stringify(dronesJSON,null,2));
```

b. Para recibir las posiciones de los diferentes drones y devolverle el mapa, definimos la función movimiento.

```
const movimiento = async () =>{
    await consumer.connect();
    await consumer.subscribe({topic:"posiciones"});
    await consumer.run({
```

está recibirá las posiciones puesta en el topic posiciones y si las condiciones son adecuadas y se le ha mandado al engine parar el espectáculo, este se lo comunicará a los drones mediante el topic mapa.

```
eachMessage: async({topic,partition,message})=>{
    if(estado=="CONDICIONES CLIMATICAS ADVERSAS. ESPECTACULO FINALIZADO" || estado=="STOP"){
        await producer.send({
            topic:'mapa',
            messages:[
                {value:estado}
            ]
        })
        producer.disconnect()
    }
```

En condiciones normales el engine cogerá la posición y buscará al dron. Si no lo encuentra no hará nada.

```
const position = JSON.parse(message.value)
console.log(position)
const dron_to_move = tablero.drones.find(dron => dron.id == position.id)
if(dron_to_move){
```

si lo encuentra comprobará que la posición es correcta. Si es así cambiará la posición actual del dron, después comprobará si ha llegado a su posición y si ha llegado cambiará el valor arrived a true. Después enviará el tablero entero para ser procesado por el dron.

```
    if(posicionConsecutiva(position.posicion,dron_to_move.posicion_actual)){
        dron_to_move.posicion_actual = position.posicion

        if(llegado(position.posicion,dron_to_move.posicion_actual)){
            dron_to_move.arrived = true;
        }
        await producer.send({
            topic: 'mapa',
            messages: [
                {value:JSON.stringify(tablero,null,2)}
            ]
        })
```

seguidamente guarda los cambios del dron en la base de datos.

```
const drones = await fs.promises.readFile('drones_DB.json','utf-8');
const dronesJSON = JSON.parse(drones);
const dron_to_edit = dronesJSON.drones.find(dron => dron.ID == dronI.id)
dron_to_edit.posicion_actual = dronI.posicion_actual
const added = await fs.promises.writeFile('drones_DB.json',JSON.stringify(dronesJSON,null,2));
```

3. Parte de Engine-Dron

La parte que he realizado del Engine-dron es el engine que se mantiene escuchando a que le lleguen peticiones.

a. Si se intenta conectar, pero el número de drones que hay en el tablero ya es el número máximo, avisa al dron de que no hay espacio para más drones y desconecta la conexión.

```
console.log('Usuario conectado');
if(tablero.drones.length == process.argv[3]){
    client.emit('mensaje','No hay sitio para más drones');
    client.disconnect();
}
```

b. En caso contrario esperará a que le pases el id y comprobará que es id existe en la base de datos.

```
else{
    client.emit('mensaje','Id: ');
    client.on('id',(id)=>{
        fs.readFile('dron.json','utf8',(err,data)=>{
            if(err){
                console.error('Error al leer dron.json:',err)
                return;
            }
            try{
                const dronesData = JSON.parse(data);
                const dron_to_seach = dronesData.drones.find(dron => dron.ID == id);
                if(dron_to_seach){
```

A continuación, esperará a que le envie el token y comprueba que es correcto.

```
if(dron_to_seach){
    client.emit('mensaje','Token: ');
    client.on('token',(token)=>{
        if(dron_to_seach.TOKEN === token){
            client.emit('right','Correcto')
```

Si no es correcto. Envía client.emit('right','Incorrecto')

Si es correcto. Envía client.emit('right','Correcto') y añade al tablero el dron.

```
if("posicion_actual" in dron_to_seach){
    tablero.drones.push({
        id:id,
        posicion_actual:dron_to_seach.posicion_actual,
        arrived:false
    })
}else{
    tablero.drones.push({id:id,arrived:false})
}
```

PRACTICA 2

REGISTRY

Api rest

Se ha realizado una implementación una conexión mediante api para realizar las mismas funcionalidades que se hacían por sockets (dar de baja, dar de alta, actualizar). Para la implementación de la api se ha utilizado el framework de nodejs express.

Para dar de alta se ha utilizado un método POST, se obtiene el id y el alias en el cuerpo. Y a partir de eso llama a la función registerDrone que también se utilizaba para los sockets. Si todo va bien, envía el dron creado con el token que ha generado, acompañado de un status 200, si el dron ya existe enviará un error 400 y si hubo problemas con la base de datos envía un error 500

```
app.post("/register",function(req,res){
  const drone = req.body;
  registerDrone(drone, (response)=>{
    if(response.error){
      if(response.error === 'Ya existe un dron con el id: ' + drone.ID){
        res.status(400)
        console.log(response);
        res.send(response)
      }
      else{
        res.status(500)
        console.log(response)
        res.send(response)
      }
    }
    else{
      res.status(200);
      console.log(response);
      res.send(response);
    }
```

Para dar de baja se utiliza un método DELETE, donde a partir del id pasado por parámetro en la ruta, obtiene el dron y lo elimina desde la función deregisterDrone, si va todo bien se envia el dron eliminado y un estado 200. Si no lo encuentra envia un error 404 y si hay problemas con la base de datos un error 500.

```
app.delete("/:id",function(req,res){
  var id = parseInt(req.params.id)
  deregisterDrone(id,(response)=>{
    if(response.error){
      if(response.error === 'Drone not found'){
        res.status(404)
        res.send(response)
      }
      else{
        res.status(500)
        res.send(response);
      }
    }else{
      res.status(200)
      res.send(response);
    }
  })
})
```

Para actualizar se obtiene el dron de la ruta y los datos a cambiar (Id, alias) del cuerpo. Se actualiza llamando a la función verificarUpdate y si todo va bien envia un 200 con el dron actualizado y si los errores funcionan igual que en el dar de baja.

```
app.put("/:id", function(req,res){
  const id = parseInt(req.params.id)
  const drone = req.body;
  verificarUpdate(id, drone.ID,drone.ALIAS,(response)=>{
    if(response.error){
      if(response.error === 'Drone not found'){
        res.status(404);
        res.send(response)
      }
      else{
        res.status(500);
        res.send(response)
      }
    }else{
      res.status(200);
      res.send(response);
    }
  });
})
```

Seguridad

En los sockets se ha utilizado el framework crypto para realizar un cifrado asimétrico RSA con el drone. Tanto el dron como registry generan claves públicas y privadas.

```
const {publicKey, privateKey} = crypto.generateKeyPairSync('rsa',{
  modulusLength: 2048,
  publicKeyEncoding:{
    type: 'spki',
    format: 'pem'
  },
  privateKeyEncoding: {
    type: 'pkcs8',
    format: 'pem'
  }
});
```

Cuando el dron establece conexión con registry, el registry le envía la clave pública que ha generado y el dron le envía su clave pública cuando indica que quiere realizar una operación.

```
console.log('[Cliente conectado]');
socket.emit('publicKey',publicKey);
```

Registry

```
socket.on('publicKey',(serverpublicKey)=>{
  clavePublica = serverpublicKey
  options(socket);
})
```

Dron

Por ejemplo, en la siguiente imagen cuando quiere registrar un dron le envía el id y el alias cifrado y la clave pública para que el registry pueda enviarle cosas cifradas.

```
case '1':
  rl.question('ID: ', (id) => {
    rl.question('Alias: ', (alias) => {

      const drone = { ID: id, ALIAS: alias };
      const encryptedDron = crypto.publicEncrypt({
        key: clavePublica,
        padding: crypto.constants.RSA_PKCS1_PADDING
      }, Buffer.from(JSON.stringify(drone)));

      //socket.emit('register', JSON.stringify(drone));
      socket.emit('register',{clave:publicKey, datos:encryptedDron});
```

Después registry recibe la información cifrada, la decodifica con la clave privada que tiene, hace las operaciones necesarias para cumplir con la petición del dron y obtener un resultado lo cifra con la clave pública que le ha pasado el dron.

```
socket.on('register', (datos) => {
  let client_publicKey = datos.clave
  const descrypterDron = crypto.privateDecrypt({
    key:privateKey,
    padding: crypto.constants.RSA_PKCS1_PADDING
  }, datos.datos)

  const data = JSON.parse(descrypterDron.toString());

  registerDrone(data, (response) => {
    console.log(response)
    const encryptedToken = crypto.publicEncrypt({
      key: client_publicKey,
      padding: crypto.constants.RSA_PKCS1_PADDING
    }, Buffer.from(JSON.stringify(response)));

    //socket.emit('responseRegister', JSON.stringify(response));
    socket.emit('responseRegister', encryptedToken);
  });
});
```

Finalmente, el dron recibe la información cifrada y la decodifica con su clave privada. Lo muestra por pantalla.

Open weather

Se ha sustituido el servidor weather por una api que se llama desde el engine repetidas veces. De este modo, comprueba si la temperatura en la ciudad que hay apuntada en el fichero llamado api_key.json tiene la temperatura correcta para seguir/empezar el espectáculo.

Este fichero se llama así porque, aparte de contener la ciudad, también contiene la clave de api para hacer una petición al open weather.

En este caso usamos axios para realizar la petición, mientras que en dron se utiliza fetch. Esto es debido a que la gestión de errores 404 y 400 daba menos problemas con fetch.

```
let idIntervalo1;
async function getTemperature() {
    const data = await fs.promises.readFile('api_key.json', 'utf-8');
    const datos = JSON.parse(data);
    const apiKey = datos.api_key;
    const city = datos.city;
    const apiUrl = `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}`;
    const response = await axios.get(apiUrl);
    const temperature = Math.round(response.data.main.temp - 273.15);
    return temperature;

}
```

Token

Ahora como el token caduca se ha utilizado jwt-simple para generarlo. Se compone del alias y la fecha de caducidad que se calcula usando moment. Esto lo que hace es coger el tiempo actual y le añade 20 segundos.

```
function generateRandomToken(alias) {
  const payload = {
    alias:alias,
    exp: moment().add(20, 'seconds').valueOf()
  };
  const TOKEN = jwt.encode(payload, secret);
  return TOKEN;
}
```

Secret es una contraseña que ayuda a codificar y se encuentra definida al principio

```
const secret = 'claveSecretaParaToken';
```

Todo esto ocurre en el registry.

En el engine cuando se realiza la operación de autenticación, lo que hace es decodificar el token que se encuentra en la base de datos y comparar el tiempo que hay guardado en el tiempo con la fecha/hora en este momento. Si ahora es un número mayor al tiempo de caducidad quiere decir que el token ha caducado, por lo que, deniega el acceso y el dron debe solicitar otro nuevo token a registry.

```
(dron_to_seach.TOKEN === token) {
 const this_token = jwt.decode(token, secret);
 const actual_time = moment().valueOf();
 if (actual_time < this_token.exp) {
     apuntarEvento(clientIp,"autentificacion","El dron actentificado",[id,token])
     client.emit('right', 'Correcto')
     let aux = [1, 1];
     if ("posicion_actual" in dron_to_seach) {
         aux = dron_to_seach.posicion_actual
     }
```

Cifrado del canal

Se planteó la idea de implementar un sistema simétrico como se hizo en registry, no obstante el intercambio de clave pública tanto por parte de dron como por parte de engine, resultaba tedioso. Porque desde la autenticación se pensó que el engine podría generar una clave pública y privada y le pasaría la pública al dron. Y los drones tendrían que generar sus claves públicas y privadas y pasarle la clave pública al engine. No obstante, eso serían muchas claves con las que tratar a la hora de gestionar kafka.

Otra opción podría ser que el engine creará dos claves públicas y dos privadas y que pasara una de cada para poder cifrar de manera asimétrica. ¿Pero qué sentido tendría una clave privada si la pasas por el canal?

Por lo que se decidió crear una clave aleatoria y utilizar un sistema simétrico.

```
const claveSimetrica= jwt.encode(secret,Math.floor(Math.random()*process.argv[2]).toString())
```

Se pasa desde la autenticación y se codifica/decodifica usando jwt.

```
id = jwt.decode(Esteid,claveSimetrica);
```

siendo la clave la generada aleatoriamente y el primer parámetro el mensaje a enviar.

A la hora de comunicarse con kafka se utiliza la misma clave codificando lo que se envía por el producto.

```
async function enviarNuevaPosicion(id, posicion) {
    const mensaje = JSON.stringify({ id, posicion });
    const encrypted =  jwt.encode(mensaje, claveSimetrica);
    producer.send([{ topic: 'newPosition', messages: encrypted }], function (err, data) {
        if (err) console.error('Error al enviar nueva posición:', err);
        else console.log('Nueva posición enviada:', data);
    });
}
```

y decodificando al consumidor.

```
const movimiento = () => {
    consumer.on('message', function (message) {
        const decrypted = jwt.decode(message.value, claveSimetrica);
        const position = JSON.parse(decrypted);
        const dron_to_move = tablero.drones.find(dron => dron.id === decrypted.id);
        if (dron_to_move) {
            if (posicionConsecutiva(position.posicion, dron_to_move.posicion_actual)) {
                dron_to_move.posicion_actual = position.posicion;
                if (llegado(position.posicion, dron_to_move.posicion_destino)) {
```

Auditoría

Por último, la auditoría es una función que llama el engine cuando ocurre un evento. Ya sea un fallo al conectarse con kafka

```
consumer.on('error', function (error) {
    apuntarEvento(mi_ip,"kafka","error en el productor de kafka",[]);
    console.error('Error en el consumidor de Kafka');
```

o realizar con éxito una autenticación.

```
if (actual_time < this_token.exp) {
    apuntarEvento(clientIp,"autentificacion","El dron actentificado",[id,token])
```

La funcion apuntarEvento guarda la ip del implicado, la fecha y hora, la accion, que ocurre y los parámetros si existen.

```
async function apuntarEvento(ip='',operacion, evento, parametros=[]){
    fs.readFile('auditorias.json', 'utf-8',(err,datos)=>{
        if (err) {
            console.error('Error al leer el archivo:', err);
            return;
        }

        let auditoriasJSON;
        try {
            // Parsear el contenido del archivo JSON
            auditoriasJSON = JSON.parse(datos);
        } catch (err) {
            console.error('Error al parsear el JSON:', err);
            return;
        }

        // Obtener la fecha y hora actual
        let fechaHoraActual = new Date().toISOString();

        // Crear el objeto de auditoría
        let auditoria = {
            "fechaHora": fechaHoraActual,
            "ip": ip,
            "Donde": operacion,
            "evento": evento,
            "parametros": parametros
        };

        // Añadir la auditoría al array
        auditoriasJSON.auditorias.push(auditoria);
```

y esto se guarda en auditorías.json.

DRONE

Dron se ha editado para codificar el canal entre registry y dron y para codificar el canal entre engine, kafka y dron.

Además como el token ahora caduca cada 20 segundos, se ha implementado una función tanto para socket como para api para que el registry devuelva un nuevo token. Esta función también se encuentra codificada.

Para socket:

```
socket.on('responseToken', (data) => {
  const descrypterResp = crypto.privateDecrypt({
    key:privateKey,
    padding: crypto.constants.RSA_PKCS1_PADDING
  }, data)
const response = JSON.parse(descrypterResp.toString());
if (response.success) {
  console.log('Nuevo token');
  droneToken = response.token;
  console.log(droneToken)
} else {
  console.log('Respuesta del servidor:', response);
}
socket.disconnect();
mainMenu();
);
```

Para api:

```
async function solicitarNuevoToken(){
  if(ID !=-1){
    try{
          const resp = await fetch(URL_BASE + ID + '/token');
          resultado = await resp.json();
            if(resp.status == 200){
              droneToken = resultado.token;
              console.log(color.green(resultado.sucess));
            }
            if(resultado.error){
              console.log(color.red(`Error ${resp.status}: ${resultado.error}`));
            }
            mainMenu();

    }catch(e){
      console.error('Error al obtener un token', e)
    }
  }else{
    console.log('No estas registrado')
    mainMenu()
  }
}
```

API ENGINE

Se ha hecho una api donde consulta la base de datos tanto drones_DB como auditorias. Tiene dos peticiones get, una para devolver el mapa.

En esta fábrica un mapa a base de arrays de arrays de strings donde si está vacío contiene un espacio (‘ ’) y si hay un dron, comprueba que la posición actual es diferente a la destino. Entonces se guarda un string con el id del dron y la inicial color que debe mostrarse si está en su posicion o no (‘r’ si no esta que viene de ‘rojo’) y (‘v’ si está que viene de ‘verde).

```
drones.forEach(dron => {
    if(dron.posicion_actual){
        const key = dron.posicion_actual.toString();
        if(dron.posicion_actual.toString()=== dron.posicion_destino.toString()){
            posicion_a_id[key] = [dron.ID, 'v'];
        } else {
            posicion_a_id[key] = [dron.ID, 'r'];
            completado = false

        }
```

Por parte de auditorías pasa un listado de todas las auditorías.

FRONT

Para el front se ha utilizado vue creando dos componentes, uno para el mapa llamado MapaComponent que se redirecciona automáticamente con la ruta por defecto y AuditoriasComponent donde se muestran todas las auditorias guardadas por el engine. Tanto el mapa como las auditorías son proporcionadas por la api engine. Podemos hacer peticiones a esa api desde los servicios creados en stores llamados AuditoriasApi y MapaApi.

DESPLIEGUE

La práctica se desplegará en tres ordenadores. Uno contendrá kafka y el servidor de clima (AD_Weather.js); otro contendrá AD_Registry-js y AD_Engine.js, y por último el ordenador o ordenadores que contiene los drones (AD_Drone.js).

Para comenzar, empezaremos desplegando el primer ordenador.

En primer lugar, desplegamos kafka. Esto consta de una serie de paso:

0. Instalar kafka sí no se encuentra ya en dicho ordenador.

1. Levantamos zookeeper con el comando

   .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

2. A continuación, Levantamos el servidor con el comando

   .\bin\windows\kafka-server-start.bat .\config\server.properties

3. Por último creamos tres topics (mapa, new_position, posiciones)
   a. .\bin\windows\kafka-topics.bat --create --topic mapa --bootstrap-server localhost:9092
   b. .\bin\windows\kafka-topics.bat --create --topic new_position --bootstrap-server localhost:9092
   c. .\bin\windows\kafka-topics.bat --create --topic posiciones --bootstrap-server localhost:9092

Ya no es necesario inicializar el servidor de clima por lo que,

Continuamos con el segundo ordenador donde se ejecuta AD_Registry.js y AD_Engine.js

1. Ejecutamos “npm i” para que se instalen los paquetes que utilizan
2. Ejecutamos AD_Registry.js primero. Aunque el orden en que se ejecutan cualquiera de los dos no es relevante. “node AD_Registry.js ip_de_este_ordenador y el puerto_escucha”
   a. ejemplo: node AD_Registry.js localhost 8080
3. Ejecutamos AD_Engine.js con “node AD_Engine.js puerto_escucha máx_drones ip_kafka puerto_kafka ip_Weather puerto_Weather”
   a. Ejemplo: node AD_Engine.js 4000 2 172.20.43.199 9092 172.20.43.201 5000
4. Ejecutamos API_Engine no hace falta indicar el puerto que vamos a utilizar siempre utiliza el puerto 8001.

Por último, el tercer ordenador con los drones y el front.

1. Ejecutamos “npm i” para que se instalen los paquetes que utilizan
2. Ejecutamos AD_Drone.js con “node AD_Drone.js localhost 4000 localhost 9092 localhost 8080”

3. Entramos en la carpeta llamada front-end
4. Ejecutamos “npm i” para que se instalen los paquetes que utilizan
5. Ejecutamos “npm run dev” para inicializar el front del servidor

RESULTADOS

Cuando ejecutas el AD_Weather.js:

```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\Practica 2\SD_Practicas>node AD_Weather.js 5000
Servidor de clima escuchando desde el puerto 5000...
```

cuando se conecta con el engine sale que hay un usuario conectado:

```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\Practica 2\SD_Practicas>node AD_Weather.js 5000
Servidor de clima escuchando desde el puerto 5000...
Usuario conectado
```

Cuando ejecutamos AD_Engine.js:

```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\Practica 2\SD_Practicas>node AD_Engine.js 4000 2 localhos
t 9092 localhost 5000
{"level":"WARN","timestamp":"2023-11-05T21:07:07.102Z","logger":"kafkajs","message":"KafkaJS v2.0.0 switched default par
titioner. To retain the same partitioning behavior as in previous versions, create the producer with the option \"create
Partitioner: Partitioners.LegacyPartitioner\". See the migration guide at https://kafka.js.org/docs/migration-guide-v2.0
.0#producer-new-default-partitioner for details. Silence this warning by setting the environment variable \"KAFKAJS_NO_P
ARTITIONER_WARNING=1\""}
Para server escribiendo STOP...Servidor de autentificación escuchando en el puerto 4000...
{"level":"INFO","timestamp":"2023-11-05T21:07:07.144Z","logger":"kafkajs","message":"[Consumer] Starting","groupId":"ad_
engine"}
Conectado con servidor clima

Para server escribiendo STOP...
```

Sale un warning de kafka, que se ha conectado con clima y la sugerencia de parar el espectáculo con STOP.

Cuando ejecutamos el AD_Registry.js:

```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\Practica 2\SD_Practicas\drone-registry>node AD_Registry.j
s localhost 3000
Servidor creado y a la espera en localhost 3000
```

cuando ejecutas AD_Drone.js:

```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\Practica 2\SD_Practicas\drone-registry>node AD_Drone1.js
 localhost 4000 localhost 9092 localhost 5000
localhost 9092
{"level":"WARN","timestamp":"2023-11-05T22:52:31.957Z","logger":"kafkajs","message":"KafkaJS v2.0.0 switched default par
titioner. To retain the same partitioning behavior as in previous versions, create the producer with the option \"create
Partitioner: Partitioners.LegacyPartitioner\". See the migration guide at https://kafka.js.org/docs/migration-guide-v2.0
.0#producer-new-default-partitioner for details. Silence this warning by setting the environment variable \"KAFKAJS_NO_P
ARTITIONER_WARNING=1\""}
Conectado al servidor de AD_Registry
Elige una opción:
1. Registro
2. Unirse al espectaculo
3. Salir:
```

le das a 1

```
1
Elige una opción:
1. Registrar dron
2. Dar de baja dron
3. Actualizar dron
4. Volver al menú principal:
```

Cuando envía una posición al engine el dron:

```
{"level":"INFO","timestamp":"2023-11-05T22:21:49.514Z","logger":"kafkajs","message":"[Consumer] Starting","groupId":"map
a-group"}
{"level":"INFO","timestamp":"2023-11-05T22:21:49.604Z","logger":"kafkajs","message":"[ConsumerGroup] Consumer has joined
 the group","groupId":"mapa-group","memberId":"ejemplo-d9f70911-8ada-47f0-bfcb-f3a2c9f0ffe3","leaderId":"ejemplo-d9f7091
1-8ada-47f0-bfcb-f3a2c9f0ffe3","isLeader":true,"memberAssignment":{"mapa":[0]},"groupProtocol":"RoundRobinAssigner","dur
ation":88}
     01   02   03   04   05   06   07   08   09   10   11   12   13   14   15   16   17   18   19   20
01 [ 3] [ 1] [ 2] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
02 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
03 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
04 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
05 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
06 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
07 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
08 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
09 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
10 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
11 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
12 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
13 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
14 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
15 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
16 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
17 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
18 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
19 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
20 [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
```

cuando ejecutas el api engine:

```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\SD_2023-24_recu>node API_Engine.js
Servidor api escuchando en el puerto 8001
```

cuando ejecutas el front-end:

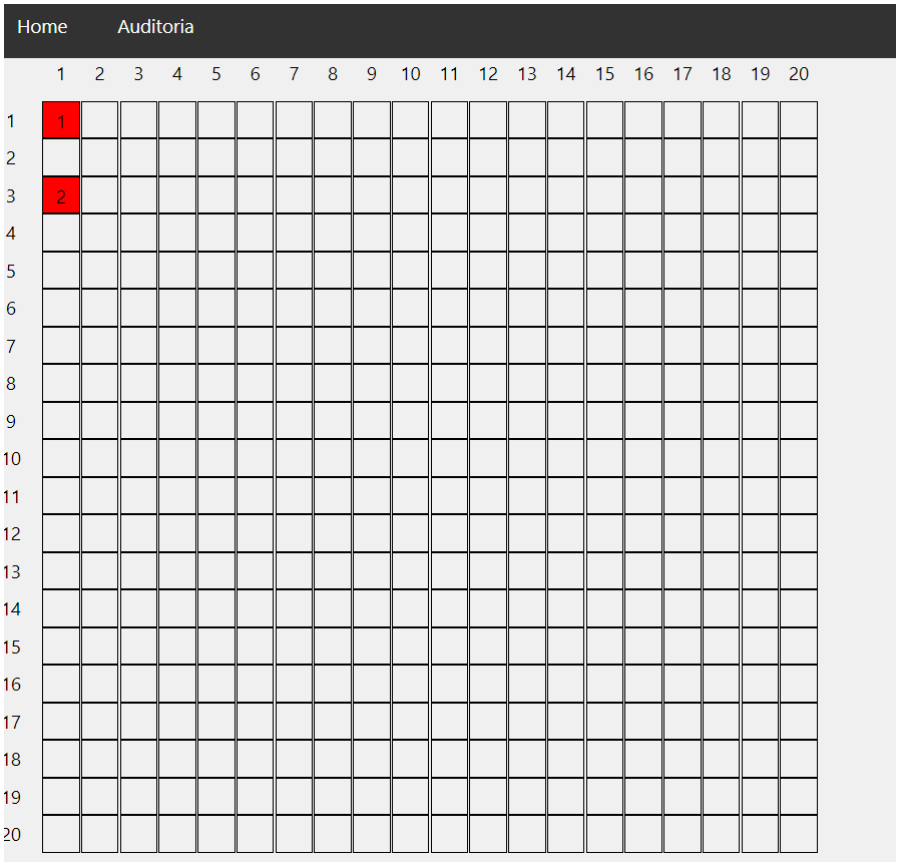
```
C:\Users\bot13\Desktop\Universidad\Cuarto_año\Primer_cuatri\SD\SD_2023-24_recu\front-end>npm run dev

> front-end@0.0.0 dev
> vite


  VITE v5.3.2  ready in 340 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: http://192.168.56.1:5173/
  ➜  Network: http://192.168.18.10:5173/
  ➜  Network: http://172.28.80.1:5173/
  ➜  press h + enter to show help
```

Abre el navegador con alguna de las url que indica



y auditoría:

Home Auditoria

.istado de Auditorías

Fecha y Hora	IP	Ubicación	Evento	Parámetros
2024-06-30T20:02:54.486Z	::1	autentificacion	El dron actentificado	• 1 • eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGlhcyI6ImHDsWHDsWFuliwiZXhwIjoxNzE5Nzc3NzkyMDkxfQ.1SZdJBTXFmK7JhUtYMafXBIbiqnhj2ZlkbwHgt4IA
2024-06-30T20:12:44.287Z	::1	autentificacion	El dron actentificado	• 1 • eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGlhcyI6ImFsYW4iLCJleHAiOjE3MTk3NzgzNzk2NDd9.viH5GGtJlbmZTOZmnSzFfdme7DhPMgdYEW3C7k
2024-06-30T20:12:48.423Z	::1	autentificacion	Token Expirado	• 2 • eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGlhcyI6InBldGVyliwiZXhwIjoxNzE5Nzc3MDM3MjI3fQ.O0Zr09fdbTiKZqtnOVEygoHM1a970GLAIYdtG5t4
2024-06-30T20:12:58.288Z	::1	autentificacion	El dron actentificado	• 2 • eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGlhcyI6Im51ZXZvNDAiLCJleHAiOjE3MTk3NzgzOTYzOTJ9.evBxFhqSeTJZY0-ckweu0Nbqx2DeGz5N01ZqrO1EBZU
2024-06-30T20:13:22.412Z	::1	autentificacion	El dron actentificado	• 2 • eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhbGlhcyI6InBlcmUiLCJleHAiOjE3MTk3Nzg0MjA3NDJ9.nTDh6pFxVEMXg58jHiYW-oGpvpgwc3luZ240pLBCI