Índice de figuras

Índice de figuras

Índice de tablas

Índice de tablas

Capítulo 3

Diseño e implementación

En este capítulo se presentan los detalles técnicos de diseño e implementación de la solución IoT que se tuvieron en cuenta durante el desarrollo del trabajo.

3.1. Arquitectura de software del sistema

El sistema cuenta con una arquitectura robusta y flexible en la que se integra el dispositivo robótico de exploración ambiental [25] desarrollado en el marco de la Carrera de Especialización en Sistemas Embebidos, con un sistema *back-end* desplegado en la nube pública [26], y una red Blockchain [19] a fin de poder asegurar la inmutabilidad y transparencia de las lecturas ambientales.

3.2. Hardware e infraestructura del sistema

3.3. Integración de los módulos y subsistemas

3.3.1. Capas de percepción y red

El desarrollo e integración de los componentes de estas capa consistió en la publicación de un *topic* MQTT desde el *backend* y la adaptación del firmware desplegado en el robot explorador para extender sus funcionalidades y enviar las lecturas ambientales al *topic*. Para esto se configuró el servicio AWS IoT Core, donde se creó una nueva instancia de un dispositivo remoto con el nombre ceit-robot-explorador y se configuró el nombre del *topic* como /ceit/robot/readings.

Se configuró la redirección de los mensajes MQTT recibidos a un *bucket* AWS S3 para poder ser almacenados y se descargó el empaquetado de certificados de seguridad que deben ser desplegados en el robot explorador para poder conectarse al *topic*.

3.3.2. Capas de procesamiento y almacenamiento cloud

3.4. Plataforma de desarrollo y despliegue

17

Capítulo 3

Diseño e implementación

En este capítulo se presentan los detalles técnicos de diseño e implementación de la solución IoT que se tuvieron en cuenta durante el desarrollo del trabajo.

3.1. Arquitectura de software del sistema

El sistema cuenta con una arquitectura robusta y flexible en la que se integra el dispositivo robótico de exploración ambiental [25] desarrollado en el marco de la Carrera de Especialización en Sistemas Embebidos, con un sistema *back-end* desplegado en la nube pública [26], y una red Blockchain [19] a fin de poder asegurar la inmutabilidad y transparencia de las lecturas ambientales.

3.2. Hardware e infraestructura del sistema

<A desarrollar>

3.3. Integración de los módulos y subsistemas

3.3.1. Capa de percepción

El desarrollo e integración de los componentes de esta capa consistió en la adaptación del firmware desplegado en el robot explorador para extender sus funcionalidades y enviar las lecturas de parámetros ambientales al *topic* MQTT. Dentro de las funcionalidades que se le agregaron al robot explorador se encontraron:

- Capturar fecha y hora local del sistema.
- Generación de coordenadas geograficas (con datos *mock*).
- Conexión segura con tópico MQTT y envío de los datos generados.

La configuración de la fecha y hora se realizó por medio del uso del servicio SNTP [57] que permite la sincronización del hardware de una red con la fecha y hora provista por servicios externos en estandar en una zona horaria. Esta configuración se realizó incluyendo el encabezado **esp_sntp.h** en el código del robot. Una vez realizado esto fue posible obtener la fecha y hora local invocando a la función *localtime*.

La generación de las coordenadas geograficas con datos *mock* se realizo mediante la generación de las ecuaciones 3.1 y 3.2 a continuación.

$$MockLat = \left(\frac{rand()}{RAND\_MAX}\right)(LAT\_MAX - LAT\_MIN) + LAT\_MIN \quad (3.1)$$

$$MockLong = \left(\frac{rand()}{RAND\_MAX}\right)(LONG\_MAX - LONG\_MIN) + LONG\_MIN \quad (3.2)$$

Finalmente, los datos capturados fueron enviados en formato JSON al tópico MQTT con la estructura de la tabla:

TABLA 3.1. Tabla de objetos AWS

Nombre del campo	Tipo del campo	Descripción
deviceId	string	Id del dispositivo
type	string	Tipo de lectura
value	string	Valor de la lectura
geoLat	string	Latitud geográfica
geoLong	string	Longitud geográfica
date	string	Fecha
time	string	Hora

A continuación podemos apreciar un valor de ejemplo del objeto JSON enviado por el robot:

```
{
        "deviceId": "12ad-dao23-ux23",
        "type": "Temperature",
        "value": "0.00",
        "geoLat": "-26.056772",
        "geoLong": "-64.014824",
        "date": "2025-04-1",
        "time": "11:23:59"
}
```

3.3.2. Capa de red

El desarrollo de los componentes de esta capa consistió en la publicación de un *topic* MQTT desde el servicio AWS IoT Core y la configuración de la lógica de redirección y almacenamiento de los mensajes recibidos en AWS S3.

Para la conexión segura con el tópico MQTT se configuró el servicio AWS IoT Core, donde se creó una nueva instancia de un dispositivo remoto con el nombre ESP32. Una vez creado este dispositivo se descargaron e instalaron en el código del robot los certificados listados a continuación:

- AmazonRootCA1.pem, renombrado a brokerCA.crt: Es la autoridad certificadora que AWS usa para firmar certificados de sus servidores. El dispositivo lo necesita para verificar la identidad del servidor AWS IoT al conectarse.

Capítulo 4

Ensayos y resultados

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

- dev-certificate.pem.crt, renombrado a client.crt: Contiene la clave pública correspondiente a la clave privada (.key) y está firmado por AWS (o por una CA en la que AWS confía) para verificar la identidad del dispositivo.
- dev-private.pem.key, renombrado a client.key: Usada por el dispositivo para firmar su identidad durante la conexión TLS. Nunca se comparte ni se sube a AWS. Tu dispositivo la usa para autenticar su certificado (.crt).

Una vez realizada la configuración del servicio AWS IoT core e integrado el robot con el tópico, se probó la recepción de lecturas con el cliente de prueba provisto por AWS suscrito al tópico *readings* como puede apreciarse en la figura 3.1.

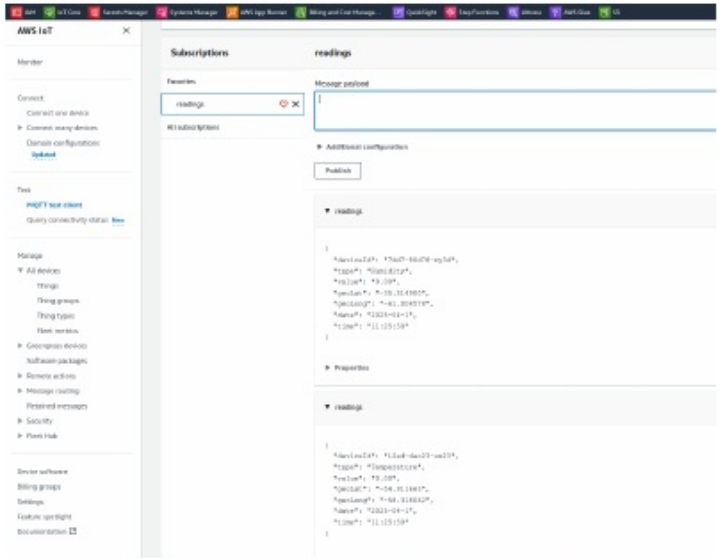

FIGURA 3.1. Prueba de recepción de mensajes MQTT.

Para el almacenamiento en AWS S3 de los mensajes recibidos, se configuro una *routing rule* o regla de redirección en AWS IoT Core, indicando mediante una consulta con sintaxis SQL, que todos los mensajes recibidos en el tópico *readings* deben almacernarce en el bucket S3 ceiot-exploratory-robot. En la figura 3.2 puede apreciarse esta configuración.

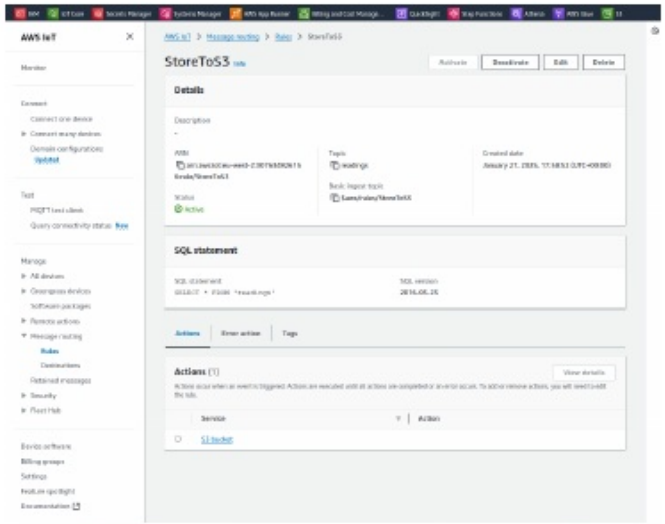

FIGURA 3.2. Configuración de redirección de mensajes MQTT.

Como resultado de la configuración realizada, los mensajes recibidos en MQTT fueron redirigidos y almacenados en AWS S3 como puede apreciarse en la figura 3.3.

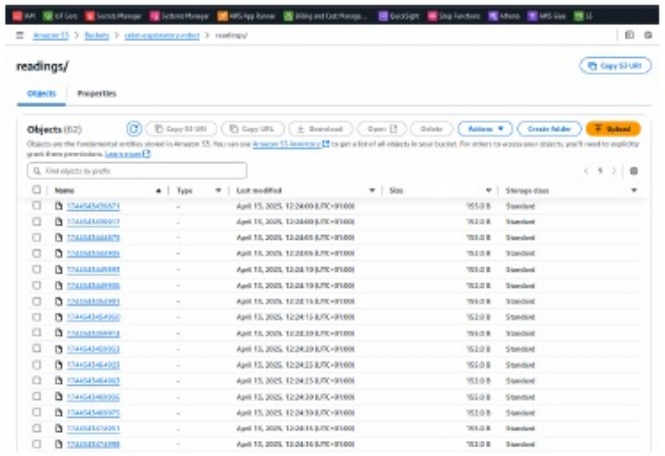

FIGURA 3.3. Almacenamiento de mensajes JSON en AWS S3.

Capítulo 5

Conclusiones

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

En esta sección no se deben incluir ni tablas ni gráficos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

3.3.3. Capas de procesamiento y almacenamiento - cloud

Una vez realizadas las configuraciones de ingesta de datos en *streaming* se realizaron las configuraciones para poder administrarlos y procesarlos. Para ello se crearon una base de datos y una tabla en AWS Glue para representar el esquema de datos almacenados en AWS S3 en formato JSON, como se puede apreciar en la figura 3.4.

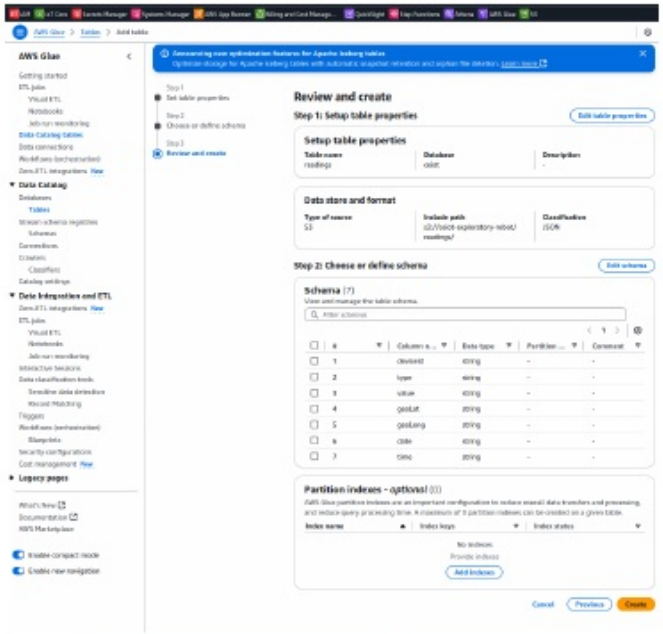

FIGURA 3.4. Creación de base datos, tabla y esquema AWS Glue.

Con el esquema de datos definido en el catálogo de AWS Glue, fue posible realizar consultas SQL sobre los datos almacenados en AWS S3 desde AWS Athena, como se puede apreciar en la figura

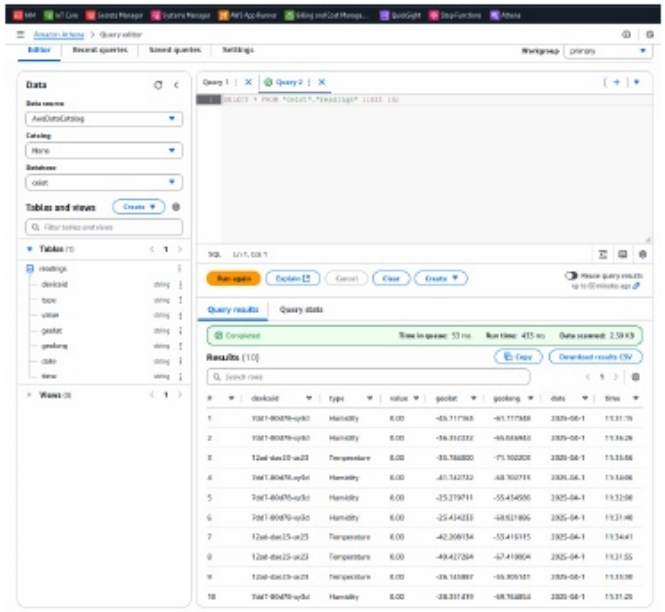

FIGURA 3.5. Consulta de datos SQL desde AWS Athena.

3.3.4. Capas de procesamiento y almacenamiento - blockchain

Uno de los primeros pasos para poder comenzar a desarrollar los componentes blockchain fue la obtención de tokens para poder realizar despliegues y ejecutar la aplicación en las redes de prueba de Ethereum sin utilizar fondos reales. Para Para poder realizar esto, primero fue necesario crear un *wallet* o billetera digital, para lo que se utilizó el servicio Metamask. Luego, para la obtención de créditos se utilizaron los *faucets* de Google [58]. Como se puede apreciar en las figura 3.6 y tras seleccionar la dirección del *wallet* y 3.7, tras realizar las transacciones en el *faucet* se reciben los fondos en la billeta digital.

Bibliografía

[1] Wikipedia. *Wi-Fi*. URL: https://es.wikipedia.org/wiki/Wifi.

[2] Wikipedia. *5G*. URL: https://es.wikipedia.org/wiki/5G.

[3] Wikipedia. *LoraWAN*. URL: https://es.wikipedia.org/wiki/LoRaWAN.

[4] Wikipedia. *Bluetooth*. URL: https://en.wikipedia.org/wiki/Bluetooth.

[5] Wikipedia. *Zigbee*. URL: https://en.wikipedia.org/wiki/Zigbee.

[6] Wikipedia. *Narrowband IoT*. URL: https://en.wikipedia.org/wiki/Narrowband_IoT.

[7] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/iot-core/.

[8] Azure. *Azure IoT Hub*. URL: https://azure.microsoft.com/en-gb/products/iot-hub.

[9] As.com | Marc Fontrodona. *Una estación de esquí china despliega una patrulla de perros robot*. URL: https://as.com/deportes_accion/snow/una-estacion-de-esqui-china-despliega-una-patrulla-de-perros-robot-n.

[10] Cadenaser.com | Radio Bilbao. *Bilbao inspecciona sus redes de saneamiento con drones y robots para mejorar la eficiencia y la seguridad*. URL: https://cadenaser.com/euskadi/2024/12/18/bilbao-inspecciona-sus-redes-de-saneamiento-con-drones-y-robots-para-mejorar-la-eficiencia-y-la-seguridad-radio-bilbao/.

[11] Los40.com | Dani Cabezas. *Así se gestiona un "tecnobosque"*. URL: https://los40.com/2024/12/10/asi-se-gestiona-un-tecnobosque/.

[12] Boston Dynamics. *Spot*. URL: https://www.bostondynamics.com/products/spot.

[13] Waygate Technologies. *BIKE - An advanced crawler robot for remote visual inspection*. URL: https://www.bakerhughes.com/waygate-technologies/robotic-inspection/bike.

[14] Latam Mining. *Robots y minería: Gobierno argentino quiere implementarlos*. URL: https://www.latam-mining.com/robots-y-mineria-gobierno-argentino-quiere-implementarlos/.

[15] Diario de Cuyo. *Gobierno pone la mira en el desarrollo de robots para la actividad minera*. URL: https://www.diariodecuyo.com.ar/politica/Gobierno-pone-la-mira-en-el-desarrollo-de-robots-para-la-actividad-minera-20200202-0052.html.

[16] Universidad Nacional de San Juan. *Robots en la minería*. URL: http://www.unsj.edu.ar/home/noticias_detalles/4810/1.

[17] Ing. Nelson Dario García Hurtado e Ing. Melvin Andrés González Pino. *Robot de exploración terrestre Geobot*. URL: https://www.unipamplona.edu.co/unipamplona/portalIG/home_40/recursos/01_general/revista_1/09102011/v01_09.pdf.

[18] Ing. Hernán L. Helguero Velásquez1 e Ing. Rubén Medinaceli Tórrez. *Robot Minero: Sistema Detector de Gases utilizando Sensores en Tiempo Real MIN – SIS 1.0 SDG-STR*. URL: http://www.scielo.org.bo/scielo.php?script=sci_arttext&pid=S2519-53522020000100003.

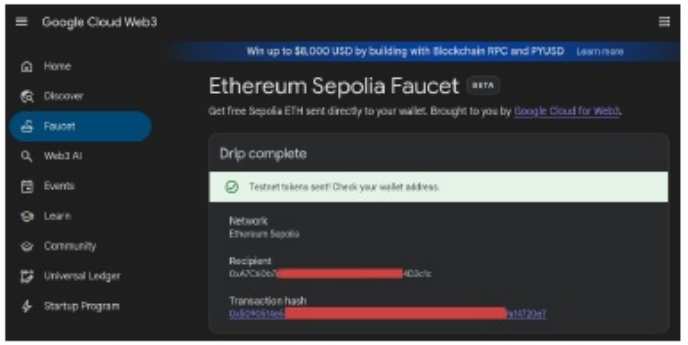

FIGURA 3.6. Obtención de créditos mediante Google Web3.

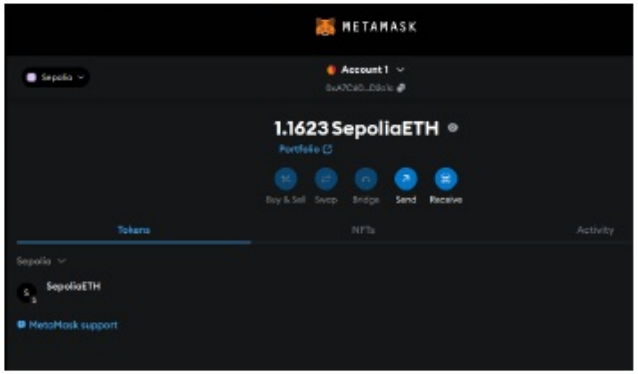

FIGURA 3.7. Saldo en Metamask.

Posteriormente como podemos apreciar en la figura, en el servicio Etherscan las transacciones realizadas para generar fondos en la dirección de la billetera quedan publicadas en la red.

[19] Wikipedia. *Blockchain*. URL: https://en.wikipedia.org/wiki/Blockchain.

[20] Wikipedia. *Smart Contracts*. URL: https://en.wikipedia.org/wiki/Smart_contract.

[21] Wikipedia. *DApp - Decentralized Application*. URL: https://en.wikipedia.org/wiki/Decentralized_application.

[22] Walmart | Global Tech. *Blockchain in the food supply chain - What does the future look like?* URL: https://tech.walmart.com/content/walmart-global-tech/en_us/blog/post/blockchain-in-the-food-supply-chain.html?utm_source=chatgpt.com.

[23] ledgerinsights | Nicky Morris. *ScanTrust's anti-counterfeit solution isn't just about blockchain*. URL: https://www.ledgerinsights.com/scantrust-anti-counterfeit-blockchain/.

[24] Paula Eiroa Interempresas | Julio Lema. *Mejorar la eficiencia energética con IoT y blockchain*. URL: https://www.interempresas.net/Energia/Articulos/446423-Mejorar-la-eficiencia-energetica-con-IoT-y-blockchain.html.

[25] Esp. Ing. Gonzalo Carreño. *LSE-FIUBA - Trabajo Final CESE- Robot de exploración ambiental*. URL: https://lse-posgrados-files.fi.uba.ar/tesis/LSE-FIUBA-Trabajo-Final-CESE-Gonzalo-Carreno-2024.pdf.

[26] Amazon Web Services. *¿Qué es una nube pública?* URL: https://aws.amazon.com/es/what-is/public-cloud/.

[27] OASIS. *MQTT Protocol Specification*. URL: https://mqtt.org/mqtt-specification/.

[28] Espressif. *ESP-IDF Programming Guide | Get Started*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/.

[29] Espressif. *ESP32*. URL: https://www.espressif.com/en/products/socs/esp32.

[30] FreeRTOS. *FreeRTOS | Real-time operating system for microcontrollers and small microprocessors*. URL: https://www.freertos.org/.

[31] Amazon Web Service. *Amazon Web Service*. URL: https://aws.amazon.com/.

[32] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/apprunner/.

[33] Docker. *Docker*. URL: https://docker.com/.

[34] Amazon Web Service. *AWS Glue*. URL: https://aws.amazon.com/glue/.

[35] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/s3/.

[36] Amazon Web Service. *AWS Athena*. URL: https://aws.amazon.com/athena/.

[37] Node.js. *Node.js*. URL: https://nodejs.org/en.

[38] Ethereum.org. *Welcome to Ethereum*. URL: https://ethereum.org/.

[39] Wthereum.org. *Ethereum Virtual Machine (EVM)*. URL: https://ethereum.org/en/developers/docs/evm/.

[40] Truffle Suite. *Truffle | What is Truffle?* URL: https://etherscan.io/.

[41] Truffle Suite. *Truffle | What is Truffle?* URL: https://sepolia.etherscan.io/.

[42] Truffle Suite. *Truffle | What is Truffle?* URL: https://holesky.etherscan.io/.

[43] Soliditylang.org. *Contract ABI Specification*. URL: https://docs.soliditylang.org/en/latest/abi-spec.html.

[44] Web3.js. *Web3.js - Ethereum JavaScript API*. URL: https://web3js.readthedocs.io/en/v1.10.0/.

[45] Wikipedia. *Proof of Stake*. URL: https://en.wikipedia.org/wiki/Proof_of_stake.

[46] soliditylang.org. *Solidity programming language*. URL: https://soliditylang.org/.

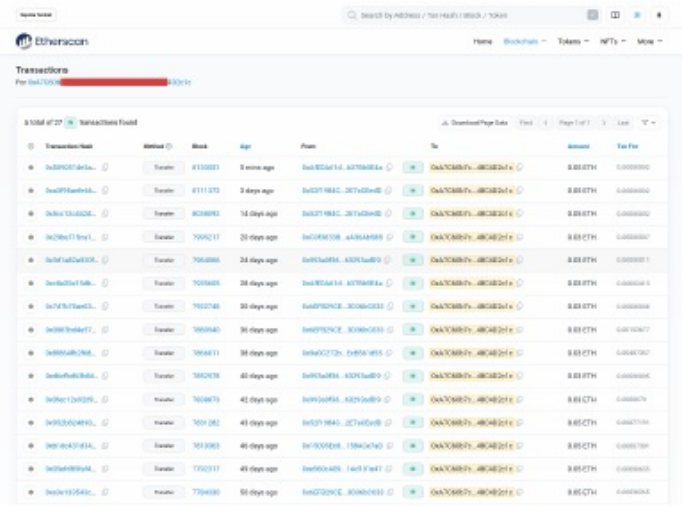

FIGURA 3.8. Transacciones de generación de fondos de prueba.

Una vez obtenidos los fondos de prueba en la billetera se procedió con el desarrollo de los componentes blockchain. Para el desarrollo de los *smart contracts* se utilizó Solidity como lenguaje de programanción y Truffle como herramienta de gestión de configuración, compilación, empaquetado y despliegue. Truffle utiliza una configuración basada en archivos Javascript para la descripción de las tareas, y para realizar el despliegue a diferentes redes, como por ejemplo de forma local a Ganache o de forma remota a redes como Sepolia, Holesky y Mainnet. En la los archivos de configuración de Truffle se agregaron entradas para poder desplegar a Ganache y a Sepolia como se puede apreciar en la figura 3.9.

```
networks: {

  development: {
    host: "ganache", // Dirección de Ganache en el host local
    port: 7545,        // Puerto donde Ganache está corriendo
    network_id: "*",   // Conecta a cualquier red
  },


  sepolia: {
    provider: () => new HDWalletProvider(
      MNEMONIC, // Frase secreta de MetaMask
      `${BLOCKCHAIN_URL}` // URL RPC de la Red a la cual se despliega
    ),
    gas: 3000000, // Límite de gas
    gasPrice: 5000000000, // 5 Gwei
    network_id: 11155111, // ID de la red Sepolia
    skipDryRun: true, // Evita correr una simulación en dry-run antes del despliegue

    // confirmations: 2, // Nº de confirmaciones antes de considerar la transacción válida
    // timeoutBlocks: 200, // Nº de bloques de espera antes de falla

    pollingInterval: 1000000,
    disableConfirmationListener: true,
  },
```

FIGURA 3.9. Configuración de redes de despliegue en Truffle.

Desde el punto de vista del *backend* blockchain, se desarrolló la dApp utilizando

[47] Truffle Suite. *Ganache | One click blockchain*. URL: https://archive.trufflesuite.com/ganache/.
[48] Truffle Suite. *Truffle | What is Truffle?* URL: https://archive.trufflesuite.com/docs/truffle/.
[49] Alchemy. *Alchemy | The most reliable way to build web3 apps*. URL: https://www.alchemy.com/.
[50] Etherscan. *Etherscan*. URL: https://etherscan.io/.
[51] Metamask. *Metamask | Your home in Web3*. URL: https://metamask.io/.
[52] Github. *Github*. URL: https://github.com/.
[53] Amazon Web Services. *AWS CodePipeline*. URL: https://aws.amazon.com/codepipeline/.
[54] Amazon Web Services. *AWS Elastic Container Registry*. URL: https://aws.amazon.com/ecr/.
[55] Visualstudio. *Visualstudio Code*. URL: https://code.visualstudio.com/.
[56] Ubuntu. *Ubuntu*. URL: https://ubuntu.com/.

Node.js y la biblioteca Javascript Web3.js para la comunicación con los *smart contracts*. Para la dApp, se desarrollaron varios *endpoints* listados a continuacion en la siguiente tabla:

Como se puede apreciar en la figura 3.10, durante el proceso de despliegue se pueden observar cierta información por pantalla:

- La red a la cual se esta realizando el despliegue.
- Los *smart contracts* incluidos en el despliegue y la dirección que toman una vez desplegados.
- El *hash* o identificador de la transacción.
- El número de bloque en el que se encuentra la transacción de despliegue.
- La dirección de la billetera digital y el balance disponible previo a la transacción.
- La cantidad de gas utilizado en el despliegue, el costo unitario y el costo total de la transacción.

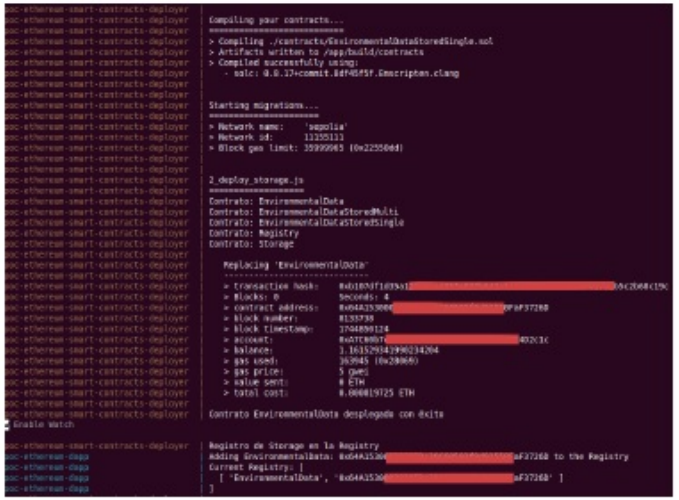

FIGURA 3.10. Salida por pantalla durante el proceso de despliegue de los componentes blockchain.

3.4. Plataforma de desarrollo y despliegue

3.5. Tabla de todos los objetos AWS creados