

Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología

Maestría en Ingeniería de Software y Sistemas Informáticos
Implementación de tecnología Serverless

**Computing en módulo de gestión de
adopciones para centros de bienestar
animal en México**

Trabajo fin de estudio presentado por:	<ul style="list-style-type: none">• Gerardo Alberto Cataño Cañizales• Luis Diego Fierros Zamarripa• Jessica Herrera Gutiérrez• Betania Estebania Jimenez Jimenez
Tipo de trabajo:	Proyecto de Aplicación
Modalidad:	En línea
Director/a:	Dr. José Eduardo Ferrer Cruz
Fecha:	12 de agosto de 2024

Resumen

El presente estudio aborda la implementación de la tecnología Serverless Computing aplicada durante el desarrollo de un módulo de Software para la gestión de adopciones, y cuya puesta en funcionamiento es aplicable dentro de centros, fundaciones u organizaciones de bienestar animal en México.

A través de una metodología ágil y mediante el uso de servicios e infraestructura Serverless en AWS, se desarrolló una solución de Software en la nube basada en una colección de REST APIs, que permiten gestionar de manera eficiente el registro, seguimiento y la promoción de adopciones de animales en situación de calle.

Dicha solución pretende obtener una mejora significativa en la eficiencia operativa de los centros de adopción mediante la automatización de las actividades de registro manual que conllevan los procesos de adopción, así como una mayor transparencia de la información recopilada durante los mismos.

Palabras clave: Adopción de mascotas, API REST, AWS, Gestión de bienestar animal, Serverless Computing.

Abstract

This study addresses the implementation of Serverless Computing technology applied during the development of a software module for adoption management, and whose implementation is applicable within centers, foundations or animal welfare organizations in Mexico.

Through an agile methodology and by using Serverless services and infrastructure in AWS, a cloud software solution was developed based on a collection of REST APIs, which allow efficient management of the registration, monitoring and promotion of adoptions of homeless animals.

This solution aims to achieve a significant improvement in the operational efficiency of adoption centers by automating the manual registration activities involved in adoption processes, as well as greater transparency of the information collected during them.

Keywords: Animal welfare management, AWS, Pet adoption, REST API, Serverless Computing.

Índice de contenidos

Resumen	2
Abstract	3
1. Introducción	9
1.1. Justificación	10
1.2. Planteamiento del problema	11
1.3. Estructura del trabajo	12
2. Contexto y estado del arte	14
2.1. Desarrollo Front-End	14
2.2. Desarrollo Back-End	14
2.3. Arquitectura REST	15
2.4. Bases de Datos NoSQL	15
2.5. Cloud Computing	18
2.6. Serverless Computing	21
3. Objetivos concretos y metodología de trabajo	23
3.1. Objetivo general	23
3.2. Objetivos específicos	23
3.3. Metodología del trabajo	24
4. Desarrollo específico de la contribución	30
4.1 Requerimientos	30
4.2 Modelo de Datos	41
4.3 Arquitectura	46
4.4 Desarrollo de interfaces de usuario	49
4.5 Pruebas unitarias	53
4.6 Pruebas de integración	57
4.7 Pruebas de usabilidad	62
4.8 Análisis de ciberseguridad	63

5. Conclusiones y trabajo futuro	68
5.1. Conclusiones	68
5.2. Trabajo futuro	70
6. Referencias bibliográficas	72
6.1. Referencias	72
6.2. Bibliografía	74
Anexo A. Artículo	77
Anexo 1. Puesta en marcha	81
Anexo 2. Manual básico de usuario	95
Anexo 3. Guía básica de mantenimiento	99
Anexo 4. Código Python para función Lambda	102

Índice de figuras

Figura 2-1: Ejemplo de Key Value Stores	16
Figura 2-2: Ejemplo de Web Column Stores	17
Figura 2-3: Tendencia de participación de mercado de proveedores de nube	20
Figura 3-1: Kanban: Metodología para aumentar la eficiencia de procesos	24
Figura 3-2: Principios Kanban	25
Figura 3-3: Kanban: Diagrama de flujo de trabajo	27
Figura 3-4: Diagrama de componentes	28
Figura 3-5: Diagrama de actividades	29
Figura 4-1: Diagrama de casos de uso todos los perfiles	38
Figura 4-2: Diagrama de casos de uso perfil Solicitante	39
Figura 4-3: Diagrama de casos de uso perfil Agente	40
Figura 4-4: Diagrama de casos de uso perfil Admin	41
Figura 4-5: Diagrama de objetos del proceso de adopción de mascotas	43
Figura 4-6: Diagrama de clases del proceso de adopción de mascotas	45
Figura 4-7: Diagrama de paquetes del módulo de gestión de adopciones	46
Figura 4-8: Arquitectura Back-End del módulo de gestión de adopciones	47
Figura 4-9: Diagrama de secuencia del módulo de gestión de adopciones	48
Figura 4-10: Página principal del proyecto	49
Figura 4-11: Captura de una solicitud de adopción y aviso de envío de datos de manera correcta	50
Figura 4-12: Visor de solicitudes de adopción	51
Figura 4-13: Edición de una solicitud de adopción y aviso de edición de datos de manera correcta	51
Figura 4-14: Visor con edición de una solicitud de adopción	52
Figura 4-15: Configuración del ambiente de pruebas unitarias en el proyecto	53
Figura 4-16: Elementos para la ejecución de pruebas unitarias UpdatePage.vue	54
Figura 4-17: Generación automática de pruebas en Postman	55
Figura 4-18: Configuración de parámetros para pruebas en Postman	56
Figura 4-19: Resultados de la ejecución de pruebas en Postman	57
Figura 4-20: Configuración de límites de API Throttling dentro de servicio AWS API Gateway	64
Figura 4-21: Ejemplo de comando oha para realización de prueba de estrés	65
Figura 4-22: Ejemplo de resultados de prueba de estrés	66
Figura 4-23: Ejemplo de resultados de pruebas de análisis en SonarQube	67

Figura A1-1: Creación de una tabla en DynamoDB	82
Figura A1-2: Tablas necesarias en DynamoDB	83
Figura A1-3: Creación de función Lambda	84
Figura A1-4: Despliegue de función Lambda	85
Figura A1-5: Rol por defecto de la función Lambda dentro de servicio IAM	86
Figura A1-6: Añadir permisos de DynamoDB al rol por defecto de la función Lambda dentro de servicio IAM	87
Figura A1-7: Integración de función Lambda en API Gateway	88
Figura A1-8: Configuración de rutas en API Gateway	89
Figura A1-9: Configuración de Stage en API Gateway	90
Figura A1-10: Configuración de Triggers aplicados a la función Lambda	91
Figura A1-11: Opción para monitoreo de logs en CloudWatch desde función Lambda	92
Figura A1-12: Ejecución de API para alta de usuario desde Postman	93
Figura A1-13: Registro de logs para API de alta de usuario en CloudWatch	94
Figura A2-1: Sitio web - Centro de adopción PG+, página de bienvenida	95
Figura A2-2: Formulario para la creación de un usuario	95
Figura A2-3: Ejemplo de creación de nuevo usuario en la plataforma	96
Figura A2-4: Ventana que se visualiza una vez creado su perfil en el portal	96
Figura A2-5: Realización de una solicitud de adopción, Captura y envío de solicitud	97
Figura A2-6: Ventana con la opción de visualizar solicitudes creadas	98
Figura A2-7: Ejemplo de visualización de captura de solicitud de adopción	98
Figura A3-1: Nivel gratuito en servicios de AWS	100
Figura A3-2: Planes de soporte en AWS	101

Índice de tablas

Tabla 4-1: Casos de Uso - Requisitos	35
Tabla 4-2: Funciones por perfil	37
Tabla A1-1: Configuración de tablas y partition keys en DynamoDB	82
Tabla A1-2: Configuración de métodos, rutas e integraciones en API Gateway	88

1. Introducción

El sistema de gestión y seguimiento de adopciones está dirigido a centros de bienestar animal enfocados en la adopción de perros y gatos callejeros con el objetivo de ayudar disminuir la cantidad de animales abandonados expuestos a los peligros de las calles, tales como enfermedades como la rabia, la micosis, leptospirosis, parásitos entre otras, además de disminuir el riesgo que representan para la salud humana tomando en cuenta que estos animales pueden ser transmisores de estas enfermedades.

La idea de crear este proyecto surge ante la necesidad de crear un sistema de adopción que permita combatir el abandono y brindar un hogar a los animales que lo requieren, esto tomando en cuenta las consecuencias que trae consigo el abandono de animales, proporcionando un software que facilite su registro, promocionar la adopción por medio de canales digitales, además de permitir que las personas interesadas en adoptar puedan buscar a sus futuras mascotas por determinadas características y condiciones, permitiendo una mayor claridad en el proceso de adopciones, de manera que se reduzca el riesgo de abandono.

Este proyecto busca ser parte de la solución a esta problemática por medio de la construcción de una aplicación web que permita la gestión y seguimiento de adopción de animales en situación de calle, utilizando tecnologías de última generación, de manera que las personas en todo el país de México puedan conectar con la causa.

Las tecnologías por utilizar en este proyecto se centran en la computación Serverless utilizando los servicios de AWS, ya que este ofrece una amplia gama de funciones en la nube que se adaptan al objetivo del sistema desarrollado. Además de esto, el sistema desarrollado incluirá una API de tipo REST, lo que permitirá la integración de este sistema con aplicaciones existentes por medio del consumo de los métodos que conforma el sistema, ayudando así con la reutilización en distintos formatos de este.

1.1. Justificación

México se encuentra en una situación preocupante; según Celaya (2022), México ocupa el tercer lugar en todo Latinoamérica de maltrato animal y el primero en abandono.

El término **animales de la calle** hace referencia a todos los animales, no solo perros, que se encuentran en situación de abandono y vagan por las calles sin un hogar o dueño definido. Esto puede incluir perros, gatos, y en algunos casos animales como cerdos, caballos y aves.

En México, la problemática de animales en situación de calle es un tema complejo y multifacético. Existen varias razones de las cuales la que más contribuyen a esta situación son:

- El abandono continuo de animales por parte de sus dueños sigue siendo la causa principal de la población de animales en situación de calle.
- La reproducción descontrolada y la falta de esterilización de animales contribuye a la proliferación de la población de animales en situación de calle.

El impacto en el bienestar animal y la salud pública sobre los animales en situación de calle exponen a diversos peligros, incluida la desnutrición, las enfermedades, los accidentes de tráfico y el maltrato. Además, su presencia en las calles puede plantear riesgos para la salud pública, como la transmisión de enfermedades zoonóticas (Gobierno de la CDMX, s.f.).

La colaboración entre el gobierno, las organizaciones de protección animal y la sociedad en general es fundamental para abordar de manera efectiva el problema de animales en situación de calle y mejorar el sistema de adopción en México.

Los **sistemas de adopción de animales en México** enfrentan desafíos significativos, estos pueden ser operados por organizaciones gubernamentales, como los departamentos de control animal de los municipios, así como por organizaciones no gubernamentales, refugios de animales y grupos de rescate.

Algunas de estas organizaciones que se dedican a facilitar la adopción de animales cuentan con plataformas que suelen listar perfiles de animales disponibles para adopción, brindar

información sobre los refugios y organizaciones de adopción, y proporcionar recursos para ayudar a los posibles adoptantes a encontrar el animal adecuado para ellos como:

- Adopta un Amigo: Es una plataforma en línea formada por voluntarios y personas con empatía en el tema de la adopción, ánimo de prevenir el abandono de mascotas y contra el maltrato.
- AdoptaCDMX: Es un sitio web de la Procuraduría Ambiental y del Ordenamiento Territorial del Gobierno de la Ciudad de México para promover la adopción responsable de perros y gatos.
- Proyecto Salvavidas: Es una plataforma 100% digital y una red de colaboradores y voluntarios dedicados al rescate, rehabilitación y promoción de adopciones para perros y gatos de las calles, y que radica en la zona metropolitana de la ciudad de Monterrey, Nuevo León.

La idea principal del proyecto es desarrollar un sistema web que facilite la adopción de animales en situación de calle. Aunque existen servicios web de adopción de animales, nuestro objetivo es proporcionar un servicio accesible para aquellas organizaciones que carecen de una plataforma digital para el proceso de adopción. Además, aspiramos a actuar como un intermediario entre estas organizaciones y aquellas que ya disponen de un sistema, pero que pueden ser desconocidas para el usuario y así poder facilitar la adopción de animales en situación de calle.

1.2. Planteamiento del problema

México se enfrenta a una crisis de abandono animal de proporciones alarmantes. Millones de perros y gatos viven en las calles, sufriendo de hambre, enfermedades, exposición a los elementos y violencia. Este problema no solo causa un inmenso sufrimiento animal, sino que también representa una amenaza para la salud pública y el medio ambiente.

Se estima que en México hay 27 millones de perros y gatos sin hogar, de los cuales 18.8 millones son perros y 9.1 millones son gatos (Sandoval, 2023). Cada año, se abandonan alrededor de 500,000 perros y gatos en México (Lozano, 2022).

Este problema requiere soluciones objetivas y a largo plazo, las cuales pueden incluir medidas más estrictas por la violación de las leyes existentes, programas de educación y orientación orientadas a la población, además del desarrollo de herramientas tecnológicas que permitan gestionar el proceso de adopción y darle seguimiento de manera efectiva.

Por tal motivo, este documento presenta una solución tecnológica que permita facilitar todo el proceso que conlleva encontrar el hogar idóneo para los animales en situación de calle o en abandono. Este sistema pretende cubrir los siguientes objetivos:

- Crear una base de datos de animales disponibles para adopción en la cual se describan sus características.
- Evaluar el perfil de las personas que pretenden adoptar con el objetivo de minimizar el riesgo de que los animales adoptados recaigan en el abandono.
- Permitir que las personas interesadas en adoptar puedan visualizar las características de las futuras mascotas filtrando su información por raza, edad, sexo, necesidades especiales entre otros.
- Dar seguimiento al proceso de adopción por medio de estados durante el mismo y posterior a este.

1.3. Estructura del trabajo

En los próximos apartados, se describe el contexto en el cual se desarrollará la aplicación de gestión de adopción, además de los objetivos (generales y específicos) del proyecto, se especificará de forma detallada la metodología utilizada en el desarrollo del mismo, así como la justificación del por qué elegir dicha metodología de trabajo.

Posteriormente, se mostrarán los resultados del trabajo de forma clara, por medio de imágenes que permitan visualizar el sistema desarrollado y el proceso de su creación.

Después, se describirán los beneficios de este trabajo a diferencia de proyectos similares, y para concluir, se expondrán los hallazgos del trabajo realizado, conclusiones y el trabajo a futuro por realizar.

2. Contexto y estado del arte

En esta sección se abordarán de manera general cada una de las áreas y componentes que conformarán las bases técnicas de la solución propuesta.

2.1. Desarrollo Front-End

En el ámbito del desarrollo de aplicaciones web/escritorio se ahondará en la que interactúa directamente el usuario, donde el Front-End es responsable de mostrar la información de una manera atractiva para los usuarios.

Algunos de los lenguajes de comunicación más conocidos para el desarrollo en Front-End son JavaScript, BackboneJS, Angular, React y Vue.js.

2.2. Desarrollo Back-End

En contraste con el Front-End, donde la información es presentada de manera agradable para el usuario, la principal función del Back-End es la de manipular los datos del Software que no cualquier usuario accede, estos datos son almacenados en las bases de datos.

El Back-End además de manipular los datos, también se encarga de controlar el flujo o el acceso de los servidores donde se alojan las aplicaciones. Algunos de los lenguajes comúnmente usados son PHP, Java, Ruby, .NET, Python, entre otros.

2.3. Arquitectura REST

Representation State Transfer o por su acrónimo REST, es una arquitectura para la implementación de aplicaciones y que a su vez funciona bajo otros protocolos de comunicación para su implementación como podrían ser HTTP o XML. Dicho estándar cuenta con cuatro recursos principales para su manipulación: GET (Consultar), POST (Crear), PUT (Editar) y DELETE (Eliminar).

Al estar basado en el estándar HTTP se tiene un bajo acoplamiento, donde el cliente solo conoce la URL y al seleccionar acciones en el servidor se obtiene escalabilidad. Además de que es independiente de la plataforma del servidor y del cliente, es muy utilizado por múltiples y diversos lenguajes y estándares para la comunicación.

Las respuestas que esta arquitectura maneja son regularmente JSON sin importar el tipo de lenguaje utilizado para el desarrollo.

2.4. Bases de Datos NoSQL

Las bases de datos NoSQL (Not Only SQL) surgen a partir de las necesidades de gestión de volúmenes de información masivos que no podrían ser cumplidas bajo un esquema relacional, llamados comúnmente “Sistemas de gestión de bases de datos relacionales (RDBMS)”. NoSQL es una categoría amplia para un grupo de datos que no siguen el modelo de datos relacional. Varios autores debaten si es que existen ventajas y beneficios en su uso o si solo son ideales para ciertas situaciones bajo un esquema controlado en algún laboratorio.

Las bases de datos NoSQL se clasifican en cuatro categorías generales:

- Orientadas a clave-valor (Key-Value stores)
- Orientadas a columnas (Wide Column stores)
- Orientadas a documentos (Document stores)
- Orientadas a grafos (Graph databases)

Key-Value Stores: Sistema comprometido con tener un rendimiento mayor y simplista para volúmenes de datos en gran cantidad. Dentro de cada base de datos de este tipo, existen tablas formadas por filas y columnas, y también cuenta con contenedores en donde cada uno puede contener X número de parejas de Key-Value como sea posible. Gracias a los sistemas basados en Cloud Computing, las bases de datos “Key-Value” han aumentado su popularidad.

Figura 2.1

Ejemplo de Key Values Stores

[users.cab]	[users_data.cab]
jonh=myp4ssw0rd	jonh_name=Jonh Clax
sony=4f5h0r8vn0	jonh_email=jonh@yahoo.com
	jonh_country=Canada
	jonh_birthdate=04/05/1960
	sony_name=Sony Sand
	sony_email=sony@yahoo.com
	sony_country=England
	sony_birthdate=04/05/1948

Adaptado de *Bases de Datos NoSQL*, por Gracia, H. y Yanes, O., 2012, Revista Telem@tica, 11(3), 21-33. (<https://revistatelematica.cujae.edu.cu/index.php/tele/article/view/74/74>) CC

BY-NC 4.0

Wide Column Stores: Lo que diferencia las bases de datos orientadas a columnas a los demás sistemas de bases de datos, es el almacenamiento de datos en secciones de columnas en lugar de su almacenamiento en filas de datos, además de optimizar sus consultas y sus tiempos de respuesta.

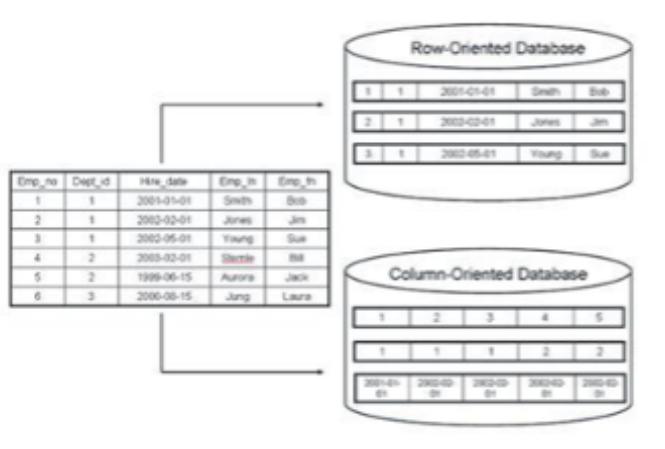
En este tipo de base de datos son mencionados los términos “Familia de columnas” donde se agrupan una o más super columnas, “Súper Columna” como la agrupación de varias columnas y “Columna”, como el almacenamiento de nombre y valor.

Algunos ejemplos de sistemas que utilizan este tipo de bases de datos son:

- Cassandra
- Big Table
- Accumulo
- Amazon SimpleDB

Figura 2-2

Ejemplo de Web Column Stores



Adaptado de *Bases de Datos NoSQL*, por Gracia, H. y Yanes, O., 2012, Revista Telem@tica, 11(3), 21-33. (<https://revistatelematica.cujae.edu.cu/index.php/tele/article/view/74/74>) CC BY-NC 4.0

Document Stores: Las bases de datos orientadas a documentos comprenden una estructura de datos más compleja que encapsula pares clave-valor en documentos, además de su indexación por árboles. Cuando se habla de almacenar y recuperar todos los datos relacionados en una sola unidad esto trae consigo una gran ventaja en el rendimiento y la escalabilidad. Algunas bases de datos de este tipo se manejan a través de HTTP y almacenan datos como documentos bajo notación JSON. Entre las más utilizadas se encuentran:

- MongoDB (mongodb.org)
- CouchDB (couchdb.apache.org)
- RavenDB (ravendb.net)

Graph DataBases: Son bases de datos utilizadas en estructuras de grafo con nodos, aristas y propiedades para representar y almacenar. Están diseñadas para datos cuya representación es en forma de grafo, donde los datos son elementos interconectados con un número no determinado de relaciones entre sí.

Los algoritmos de grafos son más aplicados en la teoría, ya que incluyen cálculos de camino más corto, rutas geodésicas, HITS y muchos más, pero la aplicación de estos algoritmos ha sido mayormente con fines de investigación, lo cual indica que es poco común que se aplique en la práctica por su gran rendimiento en la implementación. Esto ha ido cambiado con el tiempo y ya existen algunos proyectos que aplican este sistema de gestión de datos, como:

- Neo4j
- Infinite Graph
- InfoGrid
- HyperGraphDB
- DEX
- GraphBase
- Trinity

2.5. Cloud Computing

El Cloud Computing, o computación en la nube, es un modelo de servicio donde el proveedor suministra servicios informáticos a través de Internet (la nube) a sus clientes, y estos pagan únicamente por los recursos que han sido utilizados.

Dependiendo del proveedor de servicios en la nube (CSP), los servicios informáticos ofrecidos pueden ser de procesamiento, almacenamiento, redes, base de datos, Software u otros servicios más especializados, como los que involucran Inteligencia Artificial (AI), Internet de las Cosas (IoT), Big Data, Blockchain, Computación sin Servidor (Serverless Computing), entre otros.

Entre las principales características que el Cloud Computing brinda, tenemos que estos servicios son proporcionados bajo demanda y están disponibles desde prácticamente cualquier región geográfica (lo que se traduce en una baja latencia). Además, estos también pueden ajustarse rápidamente de manera automática o manual, dependiendo de las necesidades de demanda del consumidor.

El Cloud Computing ofrece 3 tipos de modelos de servicio:

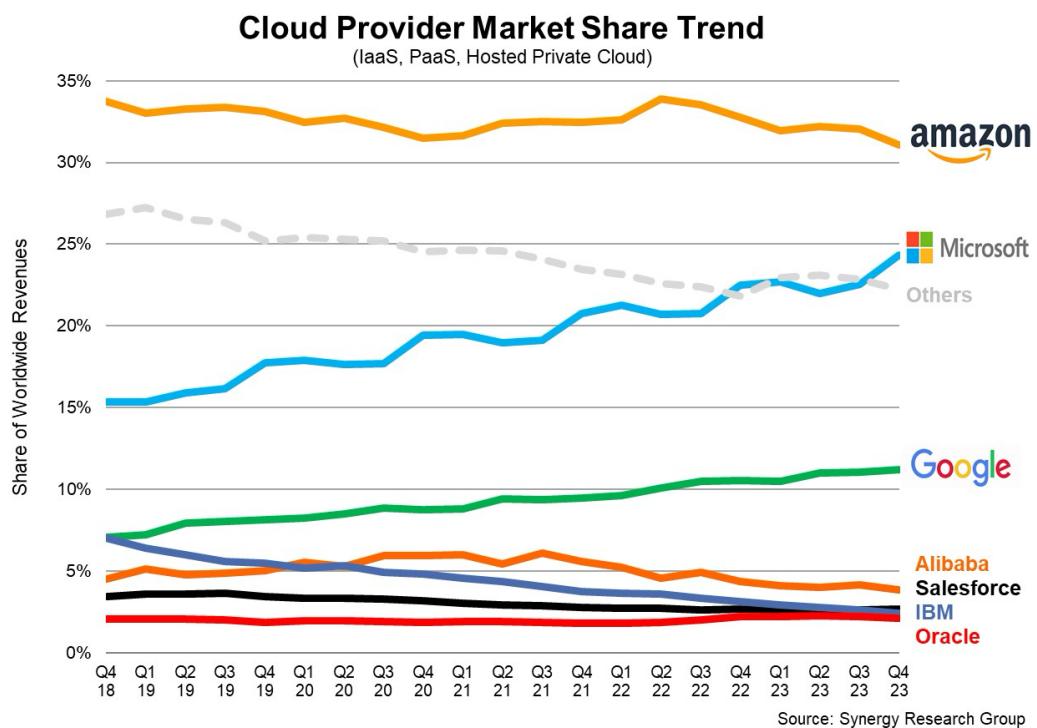
- Software as a Service (SaaS): El modelo de Software como Servicio implica que el consumidor sea capaz de utilizar las aplicaciones que se ejecutan sobre la infraestructura de nube, a través de un navegador web o un cliente ligero. Algunos ejemplos comunes de SaaS son la suite ofimática Microsoft Office 365, la aplicación de correo electrónico Gmail o herramientas de comunicación como Zoom y Slack.
- Platform as a Service (PaaS): En el modelo de Plataforma como Servicio, el consumidor utiliza los ambientes preconfigurados o servidores de aplicaciones del proveedor de nube para desplegar sus propias soluciones de Software. Ejemplos de ello son AWS Elastic Beanstalk, Google Cloud App Engine y Heroku.
- Infrastructure as a Service (IaaS): En el modelo de Infraestructura como Servicio, el proveedor de servicios proporciona los recursos fundamentales de cómputo como procesamiento, almacenamiento o redes directamente hacia el consumidor, siendo éste capaz de gestionarlos acorde a sus necesidades. Ejemplos comunes de esta

categoría son Amazon Web Services, Microsoft Azure, Google Cloud Platform y Oracle Cloud Infrastructure.

Finalmente, al primer trimestre del año 2024 y según el último estudio realizado por Synergy Research Group (2024), Amazon Web Services tiene una ligera tendencia a la baja, pero aún lidera el mercado global de proveedores de nube con un porcentaje de 31%, en tanto que Microsoft Azure y Google Cloud Platform continúan su tendencia al alza, con porcentajes de 24% y 11%, respectivamente.

Figura 2-3

Tendencia de participación de mercado de proveedores de nube



Adaptado de *Cloud Market Gets its Mojo Back; AI Helps Push Q4 Increase in Cloud Spending to New Highs* [Web], por Synergy Research Group, 2024, SRGResearch (<https://www.srgresearch.com/articles/cloud-market-gets-its-mojo-back-q4-increase-in-cloud-spending-reaches-new-highs>). Todos los derechos reservados 2024 por Synergy Research Group.

2.6. Serverless Computing

El Serverless Computing (computación sin servidor) es un modelo de servicio de tipo PaaS, donde el desarrollador publica su solución de código en forma de funciones (entrada y salida de datos) y el proveedor de nube se encarga de proporcionar la infraestructura y servicios de gestión necesarios para su correcto funcionamiento.

Aunque este modelo implica el uso de recursos computacionales como cualquier servicio de nube, el término Serverless hace alusión a que el desarrollador no tendrá que interactuar de ninguna forma con servidores, o siquiera ser capaz de conocer los detalles de donde se ejecuta su solución.

La operación del Serverless Computing está basada en la ejecución de funciones de código, las cuales pueden ser desencadenadas por diferentes tipos de eventos, como una solicitud HTTP, la subida de un archivo hacia un determinado repositorio o la actualización de un registro en una base de datos.

Una vez que se recibe un evento, el proveedor de servicios en la nube (CSP) se encargará de inicializar la infraestructura necesaria para ejecutar este código, dejándola disponible por un breve periodo de tiempo (establecido por el CSP) para poder ejecutar peticiones del mismo tipo. Una vez concluido este periodo, el CSP se encargará de remover los recursos utilizados, y el ciclo se repetirá para las peticiones subsecuentes.

Entre las principales características del Serverless Computing se encuentran:

- Sin gestión por parte del consumidor: el proveedor de nube se encarga de las tareas de gestión de la infraestructura, como la actualización del sistema operativo, instalación de parches de seguridad, etc.

- Alta disponibilidad (HA) y mecanismo de recuperación de desastres (DR): características que son responsabilidad del proveedor de nube, ofrecidas sin ningún costo adicional.
- Escalabilidad automática: característica que permite la asignación automática de mayores o menores recursos dependiendo de la demanda de peticiones en un determinado tiempo.
- Costo por utilización: el consumidor únicamente paga por el tiempo real de la ejecución del código y no por el tiempo en que permanece inactivo, como es el caso de los servidores convencionales.

Para concluir, los principales proveedores de servicios en la nube de hoy día ofrecen sus respectivas plataformas Serverless, como AWS Lambda en Amazon Web Services, Azure Functions en Microsoft Azure y Google Cloud Functions en Google Cloud Platform.

3. Objetivos concretos y metodología de trabajo

En este apartado se describirá el objetivo general de este proyecto y se detallará, a través de los objetivos específicos, lo que se desea lograr al finalizarlo.

3.1. Objetivo general

Implementar tecnología Serverless Computing en módulo de gestión de adopciones para centros de bienestar animal en México que permita consultar los datos de los animales disponibles para adopción y de los candidatos a que desean adoptar, mejorando la calidad y la claridad en el proceso utilizando la tecnología que nos ofrece la computación en la nube por medio del desarrollo de un API REST.

3.2. Objetivos específicos

- Desarrollar una aplicación fácil de integrar con sistemas existentes para gestionar el proceso de adopción de animales en situación de calle.
- Establecer un canal que permita compartir la información de los animales disponibles para adoptar de forma clara y detallada.
- Clasificar los animales a ser dados en adopción por sus características permitiendo así que los interesados puedan filtrar sus datos.
- Explorar el uso de la tecnología en soluciones a problemáticas como lo son el abandono de animales en México.

- Integrar un sistema de seguimiento post adopción que permita identificar el estado de la mascota y validar que esta esté en buenas manos, minimizando con esto el riesgo de abandono.

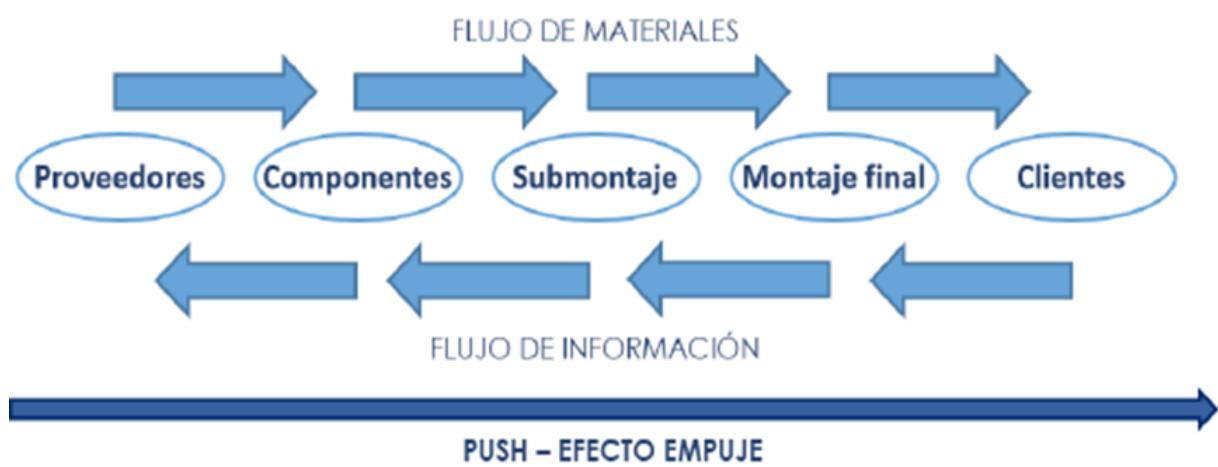
3.3. Metodología del trabajo

La metodología que será aplicada para este proyecto es la metodología Kanban, la cual se abordará en este apartado, exhibiendo su descripción, sus módulos y las fases que serán aplicadas en este.

Para hablar de la presente metodología, es necesario hablar sobre el término “Just in time” el cual se traduce como producir lo que se requiere en ese preciso momento, con la calidad que se cuenta y sin perder recursos. La principal misión de la metodología Kanban es el manejo y la gestión de los recursos/materiales fluyendo conforme a las etapas y sin tiempos de retraso.

Figura 3-1

Kanban: Metodología para aumentar la eficiencia de procesos



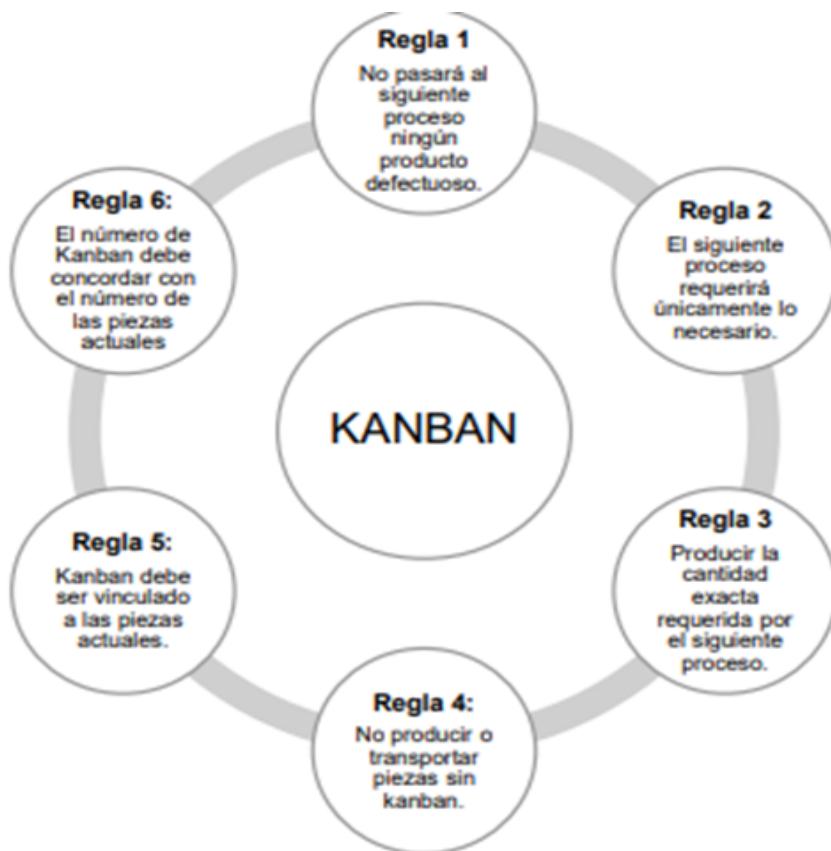
Adaptado de *Kanban. Metodología para aumentar la eficiencia de los procesos*, por Castellano, L., 2019, 3C Tecnología. Glosas de innovación aplicadas a la pyme, 8(1), 30-41.

(<http://dx.doi.org/10.17993/3ctecno/2019.v8n1e29/30-41>). CC BY-NC 4.0

Kanban también cuenta con sus principios para su deber ser, los cuales son mencionados a continuación. La Calidad, intentando hacer todo lo que se pretende realizar y optimizar en lo posible. Minimización, enfocarse en las actividades primarias (principio YAGNI). Mejora continua, ir mejorando los procesos con base al objetivo que se desea llegar. Flexibilidad, priorizar las tareas según lo necesitado. Por último, construcción y mantenimiento, que consiste en manejar un plan a largo plazo con el proveedor.

Figura 3-2

Principios Kanban



Adaptado de *Aplicación de la metodología Kanban en el desarrollo del software para generación, validación y actualización de reactivos, integrado al sistema informático de control académico UNACH* [Tesis de Bachiller], por Yépez, E. y Armijos, K., 2020, Universidad Nacional de Chimborazo. (<http://dspace.unach.edu.ec/handle/51000/6457>). CC BY 4.0

Kanban, aunque está enfocada en la mejora y la gestión de líneas de producción, pero esto no ha sido una limitante para que esta metodología ágil sea implementada en otros ambientes sociales, en este caso en el desarrollo de aplicación.

En un estudio realizado por la Universidad Nacional del Chimborazo donde se aplicó esta metodología, demostró ser superior a los métodos tradicionales, siendo adaptable a cualquier lenguaje de programación, en este caso C# con ASP.NET CORE, por lo antes mencionado se obtienen resultados satisfactorios en la generación, validación y actualización de reactivos.

Kanban ha demostrado ser eficiente y adaptable en más casos de estudio, siempre que se cuente con un equipo coordinado y cohesionado. La flexibilidad y adaptabilidad de Kanban se ha demostrado en proyectos de desarrollo de software, aplicaciones móviles, sistemas web.

Ahora que han mencionado algunas de las ventajas de Kanban, a continuación, en base a su diagrama de flujo, se mostrarán las actividades que se desarrollarán para este proyecto en específico, así como algunos de los diagramas generales que se irán detallando en secciones posteriores.

Figura 3-3

Kanban: Diagrama de flujo de trabajo

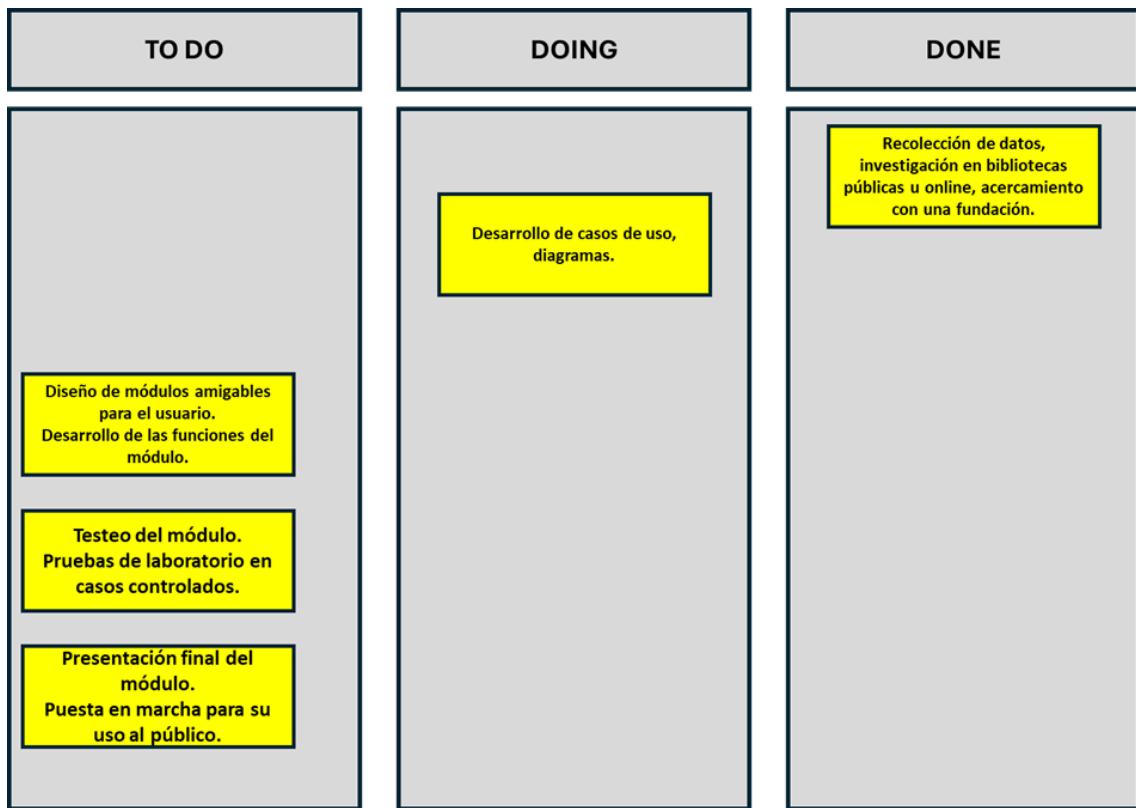


Figura 3-4

Diagrama de componentes

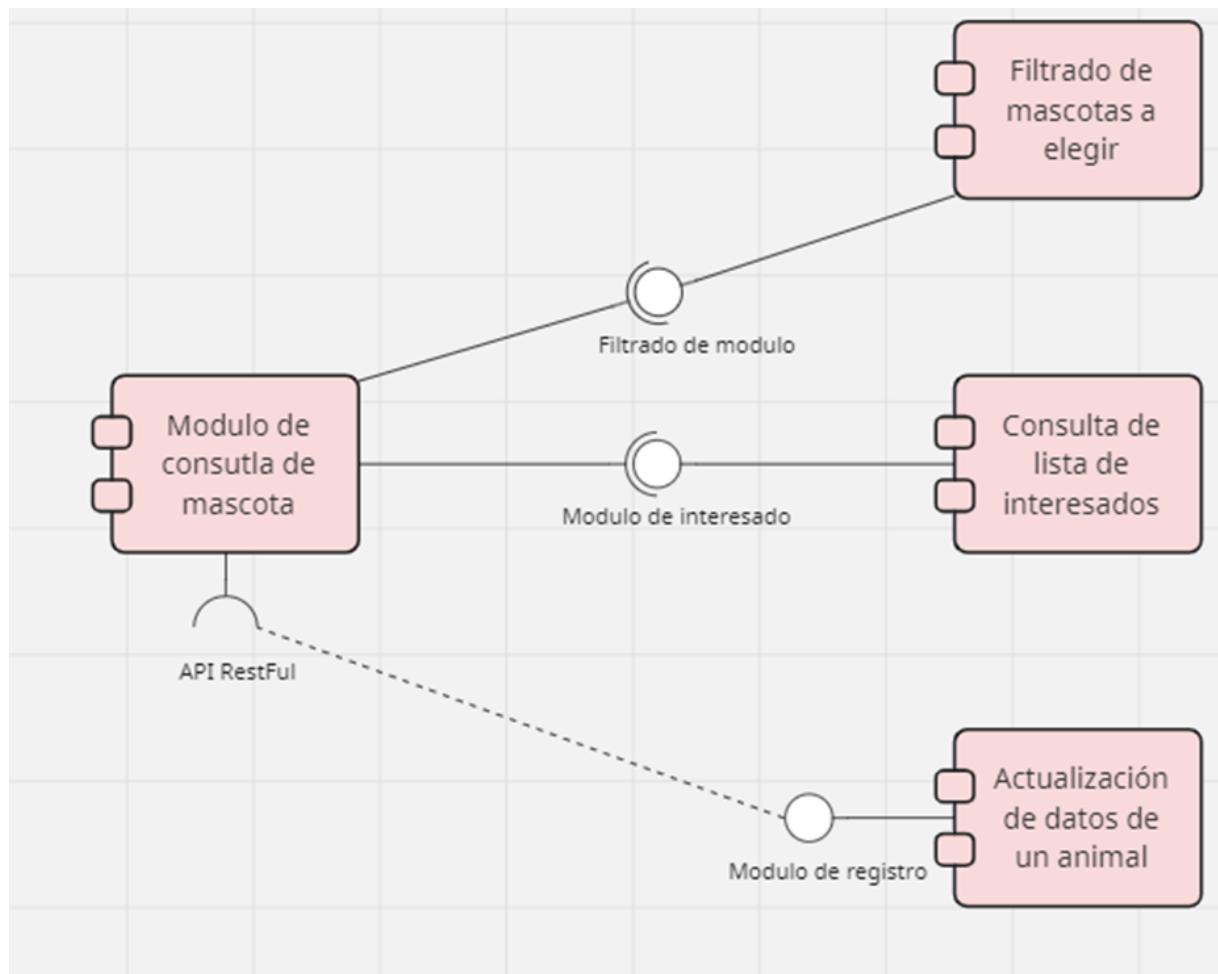
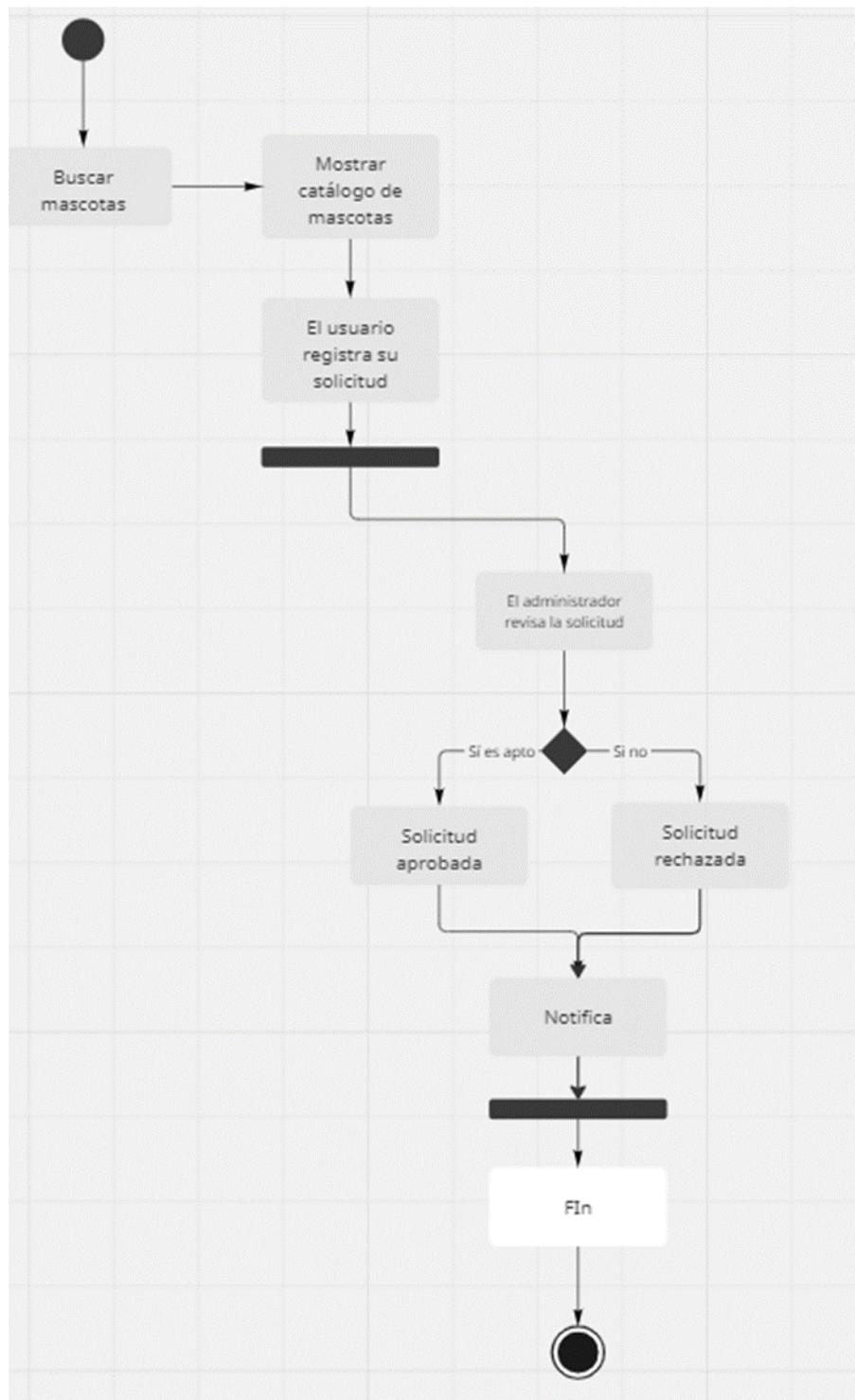


Figura 3-5

Diagrama de actividades



4. Desarrollo específico de la contribución

4.1 Requerimientos

En este apartado se describirán los requisitos funcionales y no funcionales del Sistema a desarrollar, además de esto, se especificarán los casos de uso del sistema en conjunto con los perfiles de los usuarios que se entienden usarán el sistema.

Requerimientos Funcionales

- RF1.1: El sistema debe proporcionar un endpoint GET /api/mascotas que devuelva una lista de animales disponibles para adopción con información básica (nombre, especie, raza, edad, sexo, y descripción breve).
- RF2.1: El sistema debe permitir a los usuarios consultar los detalles de un animal específico mediante el endpoint GET /api/mascotas?mascotaid={id}, devolviendo información completa del animal seleccionado.
- RF3.1: El sistema debe ofrecer un endpoint GET /api/mascotas que permita aplicar filtros por especie, raza, edad, tamaño, comportamiento, y necesidades especiales.
- RF3.2: Los filtros deben ser parámetros opcionales en la consulta y deben combinarse para refinar los resultados.
- RF4.1: El sistema debe proporcionar un endpoint POST /api/mascotas que permita registrar un nuevo animal, recibiendo información detallada del animal (nombre, especie, raza, edad, sexo, historia médica, fotos, etc.).
- RF4.2: El sistema debe devolver una confirmación y los detalles del animal registrado.

- RF5.1: El sistema debe permitir la actualización de la información de un animal mediante el endpoint PUT /api/mascotas?mascotId={id}, aceptando datos actualizados del animal.
- RF5.2: El sistema debe devolver una confirmación de la actualización realizada.
- RF6.1: El sistema debe permitir a los usuarios registrarse como interesados en adoptar mediante el endpoint POST /api/usuarios, recibiendo información del interesado (nombre, contacto, preferencias de adopción, etc.).
- RF6.2: El sistema debe devolver una confirmación del registro y los detalles del interesado.
- RF7.1: El sistema debe proporcionar un endpoint GET /api/usuarios que devuelva una lista de personas interesadas en adoptar animales.
- RF8.1: El sistema debe permitir consultar el estado de una solicitud de adopción mediante el endpoint GET /api/solicitudes?solicitudId={id}, devolviendo el estado actual de la solicitud.
- RF9.1: El sistema debe permitir la generación de reportes sobre adopciones mediante el endpoint GET /api/reportes/solicitudes, proporcionando datos estadísticos y analíticos relevantes.

Requerimientos no funcionales

- RNF1.1: El sistema debe utilizar autenticación y autorización para asegurar que solo los empleados del centro de bienestar animal puedan acceder a funcionalidades de registro y actualización.
- RNF1.2: Los datos sensibles deben ser cifrados tanto en tránsito como en reposo.
- RNF2.1: El sistema debe garantizar una disponibilidad del 99.9% para el API, asegurando que los usuarios puedan acceder a la información en todo momento.
- RNF3.1: El sistema debe ser capaz de escalar automáticamente para manejar incrementos en la carga de trabajo, especialmente durante campañas de adopción masiva.

- RNF4.1: Las respuestas del API deben ser rápidas, con un tiempo de respuesta inferior a 200 ms para la mayoría de las consultas.
- RNF4.2: El sistema debe ser capaz de manejar al menos 1000 solicitudes concurrentes sin degradación significativa del rendimiento.
- RNF5.1: El sistema debe ser compatible con los principales navegadores y dispositivos móviles, asegurando que los usuarios puedan acceder desde cualquier plataforma.
- RNF5.2: El API debe seguir los estándares REST y ser fácilmente integrable con otros sistemas mediante JSON.
- RNF6.1: El código del sistema debe estar bien documentado y seguir buenas prácticas de desarrollo para facilitar el mantenimiento y la evolución futura.
- RNF6.2: El sistema debe incluir pruebas automatizadas para garantizar que las nuevas actualizaciones no introduzcan errores.
- RNF7.1: El API debe ser fácil de usar e intuitivo, con una documentación clara y ejemplos de uso para desarrolladores externos.
- RNF8.1: El sistema debe contar con herramientas de monitoreo y generación de logs para rastrear el funcionamiento y detectar problemas en tiempo real.

Casos de Uso

Caso de Uso 1: Consultar lista de animales disponibles para adopción

- Objetivo: Permitir a los usuarios ver una lista de animales disponibles para adopción.
- Descripción: Los usuarios pueden acceder a una lista de animales que están disponibles para adopción con información básica como nombre, especie, raza, edad, sexo y una breve descripción.
- Actores: Usuarios públicos, empleados del centro de bienestar animal.
- Endpoint: GET /api/mascotas
- Respuesta: Lista de objetos de animales con sus características básicas.

Caso de Uso 2: Consultar detalle de un animal

- Objetivo: Proporcionar información detallada de un animal específico.
- Descripción: Al seleccionar un animal de la lista, los usuarios pueden ver información detallada sobre el animal, incluyendo historia médica, comportamiento, fotos y videos.
- Actores: Usuarios públicos, empleados del centro de bienestar animal.
- Endpoint: GET /api/mascotas?mascotId={id}
- Respuesta: Detalles completos del animal seleccionado.

Caso de Uso 3: Filtrar animales por características

- Objetivo: Permitir a los usuarios filtrar la lista de animales por características específicas.
- Descripción: Los usuarios pueden aplicar filtros como especie, raza, edad, tamaño, comportamiento, y necesidades especiales para encontrar el animal que mejor se adapte a sus preferencias.
- Actores: Usuarios públicos, empleados del centro de bienestar animal.
- Endpoint: GET
`/api/mascotas?especie={especie}&raza={raza}&edad={edad}&tamano={tamano}&comportamiento={comportamiento}`
- Respuesta: Lista filtrada de animales.

Caso de Uso 4: Registrar nuevo animal

- Objetivo: Permitir a los empleados del centro registrar nuevos animales en el sistema.
- Descripción: Los empleados pueden añadir nuevos animales con toda la información relevante para que estén disponibles para adopción.
- Actores: Empleados del centro de bienestar animal.
- Endpoint: POST /api/mascotas

- Entrada: Información detallada del animal (nombre, especie, raza, edad, sexo, historia médica, fotos, etc.)
- Respuesta: Confirmación del registro y detalles del animal registrado.

Caso de Uso 5: Actualizar información de un animal

- Objetivo: Permitir la actualización de la información de un animal.
- Descripción: Los empleados pueden actualizar la información de un animal, como cambios en su estado de salud o características.
- Actores: Empleados del centro de bienestar animal.
- Endpoint: PUT /api/mascotas?mascotald={id}
- Entrada: Información actualizada del animal.
- Respuesta: Confirmación de la actualización.

Caso de Uso 6: Registrar interesados en adoptar

- Objetivo: Permitir a los usuarios registrarse como interesados en adoptar un animal.
- Descripción: Los usuarios pueden registrarse y proporcionar información sobre ellos mismos y el tipo de animal que desean adoptar.
- Actores: Usuarios públicos.
- Endpoint: POST /api/usuarios
- Entrada: Información del interesado (nombre, contacto, preferencias de adopción, etc.)
- Respuesta: Confirmación del registro y detalles del interesado.

Caso de Uso 7: Consultar lista de interesados

- Objetivo: Permitir a los empleados ver una lista de personas interesadas en adoptar.
- Descripción: Los empleados pueden ver una lista de todas las personas registradas como interesadas en adoptar animales.
- Actores: Empleados del centro de bienestar animal.
- Endpoint: GET /api/usuarios

- Respuesta: Lista de objetos de interesados con sus características básicas.

Caso de Uso 8: Verificar estado de adopción

- Objetivo: Permitir a los empleados verificar el estado de una solicitud de adopción.
- Descripción: Los empleados pueden consultar el estado de una adopción para ver si está en proceso, aprobada, o completada.
- Actores: Empleados del centro de bienestar animal.
- Endpoint: GET /api/solicitudes?solicitudId={id}
- Respuesta: Detalles del estado de la solicitud de adopción.

Tabla 4-1

Casos de Uso - Requisitos

Caso de Uso	Requisitos Funcionales	Requisitos No Funcionales
Consultar lista de animales disponibles para adopción	RF1.1: GET /api/mascotas	RNF2.1: Disponibilidad del 99.9% RNF4.1: Tiempo de respuesta < 200 ms RNF5.1: Compatibilidad con navegadores y móviles RNF7.1: API intuitivo
Consultar detalle de un animal	RF2.1: GET /api/mascotas?mascotaId={id}	RNF2.1: Disponibilidad del 99.9% RNF4.1: Tiempo de respuesta < 200 ms RNF5.1: Compatibilidad con navegadores y móviles RNF7.1: API intuitivo
Filtrar animales por características	RF3.1: GET /api/mascotas con filtros RF3.2: Filtros opcionales	RNF2.1: Disponibilidad del 99.9% RNF4.1: Tiempo de respuesta < 200 ms RNF5.1: Compatibilidad con navegadores y móviles RNF7.1: API intuitivo

Registrar nuevo animal	RF4.1: POST /api/mascotas	RNF1.1: Autenticación y autorización
	RF4.2: Confirmación del registro	RNF1.2: Cifrado de datos RNF2.1: Disponibilidad del 99.9% RNF5.2: Cumplimiento REST RNF6.1: Buenas prácticas RNF6.2: Pruebas automatizadas
Actualizar información de un animal	RF5.1: PUT /api/mascotas?mascotaid={id}	RNF1.1: Autenticación y autorización RNF1.2: Cifrado de datos RNF2.1: Disponibilidad del 99.9%
	RF5.2: Confirmación de actualización	RNF5.2: Cumplimiento REST RNF6.1: Buenas prácticas RNF6.2: Pruebas automatizadas
Registrar interesados en adoptar	RF6.1: POST /api/usuarios	RNF1.1: Autenticación y autorización RNF1.2: Cifrado de datos RNF2.1: Disponibilidad del 99.9%
	RF6.2: Confirmación del registro	RNF5.2: Cumplimiento REST RNF6.1: Buenas prácticas RNF6.2: Pruebas automatizadas
Consultar lista de interesados	RF7.1: GET /api/usuarios	RNF2.1: Disponibilidad del 99.9% RNF4.1: Tiempo de respuesta < 200 ms RNF5.1: Compatibilidad con navegadores y móviles RNF7.1: API intuitivo
	RF8.1: GET /api/solicitudes?solicitudId={id}	RNF2.1: Disponibilidad del 99.9% RNF4.1: Tiempo de respuesta < 200 ms RNF5.1: Compatibilidad con navegadores y móviles RNF7.1: API intuitivo

Relación requerimientos con casos de uso, Fuente: Elaboración propia

Diagramas caso de uso

En esta sección veremos la relación de entre los casos de uso del proyecto y los perfiles de los usuarios que interactúan con el API del sistema de adopción. Estos perfiles son:

Tabla 4-2

Funciones por perfil

Perfil	Funciones
Solicitante	-Llenar formulario de solicitud -Completar test para adopción -Consulta Mascotas disponibles
Agente	-Dar seguimiento a las solicitudes -Registrar interesado -Consultar Mascotas disponibles
Admin	-Registrar Mascotas -Actualizar datos Mascotas -Dar seguimiento a las solicitudes -Registrar interesado -Consultar Mascotas disponibles

Funciones por perfil Fuente: Elaboración propia

En la **Figura 4-1** se puede visualizar el diagrama general enfocado en todos los perfiles enlazados cada caso de uso que se encuentra en el dominio de cada perfil según las funciones que realizarán al utilizar el sistema, posteriormente vemos de forma separada la relación de casos de usos por perfiles siendo que en la figura 4-2 vemos la relación de casos de uso con el perfil de Solicitante, la figura 4-3 con el perfil Agente y en la figura 4-4 con el perfil de Admin.

Figura 4-1

Diagrama de casos de uso todos los perfiles

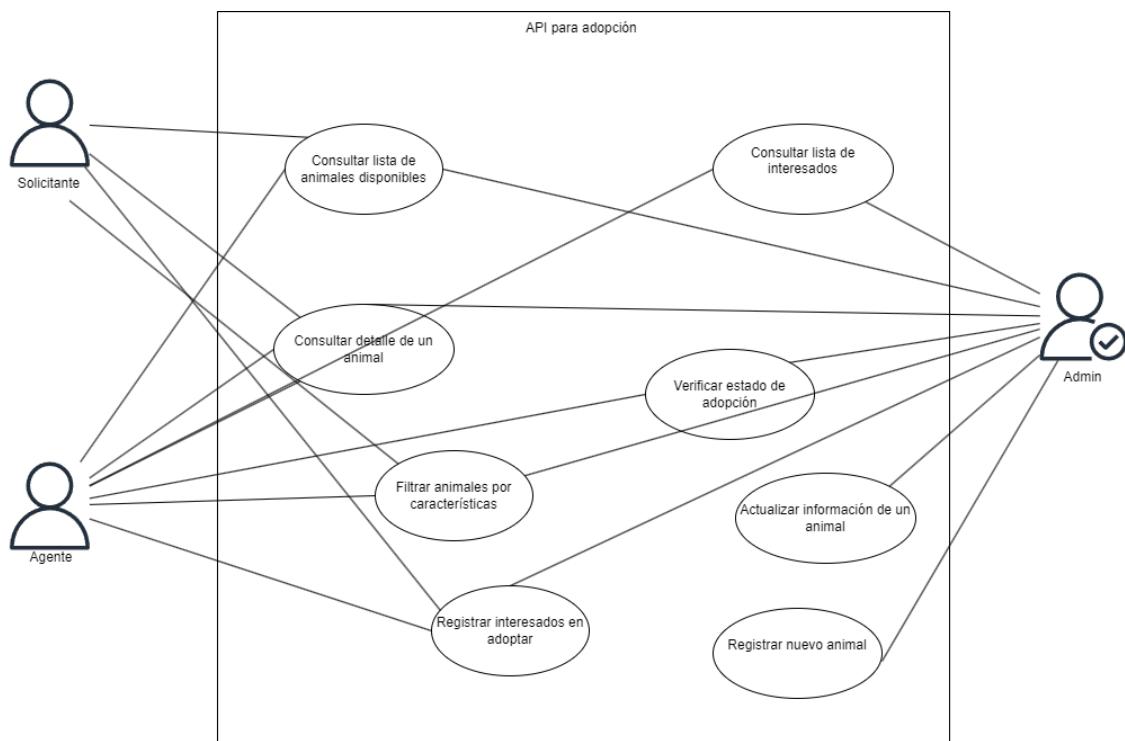


Figura 4-2

Diagrama de casos de uso perfil Solicitante



Figura 4-3

Diagrama de casos de uso perfil Agente

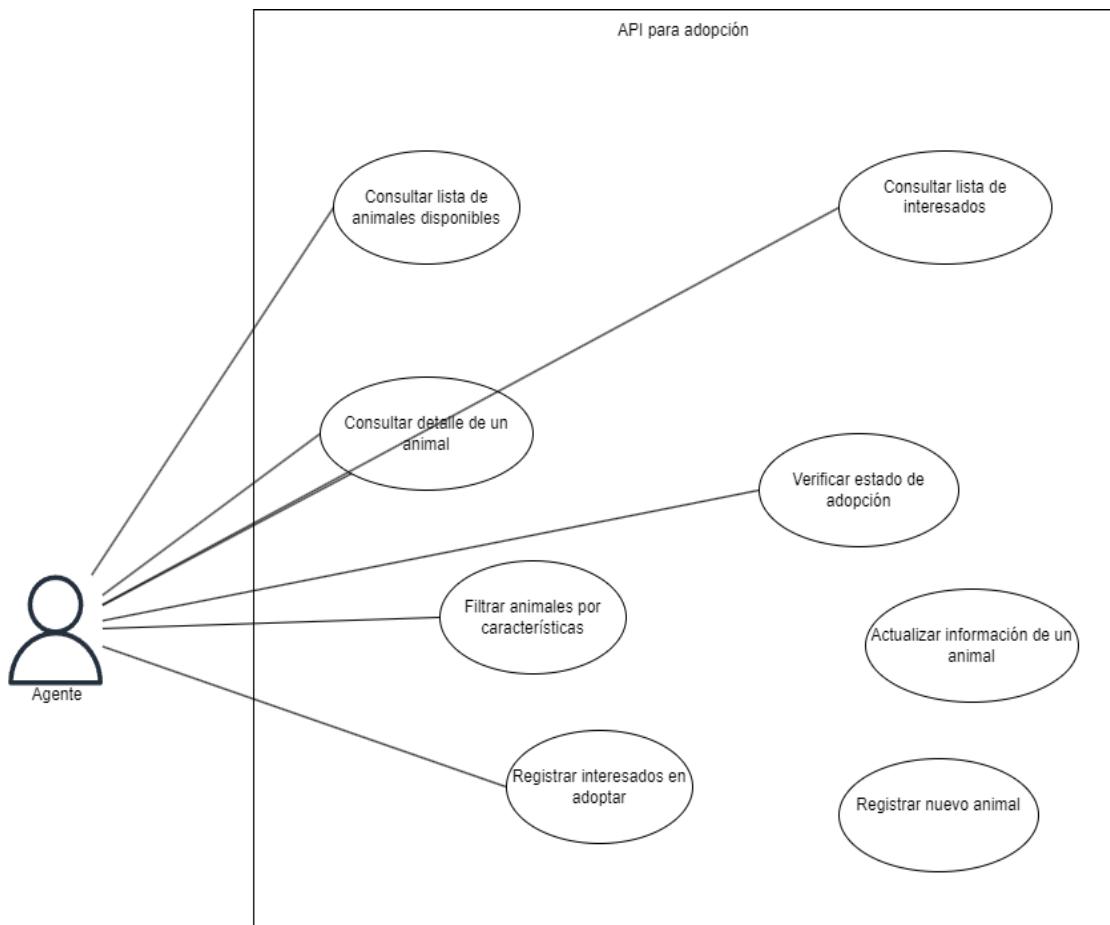
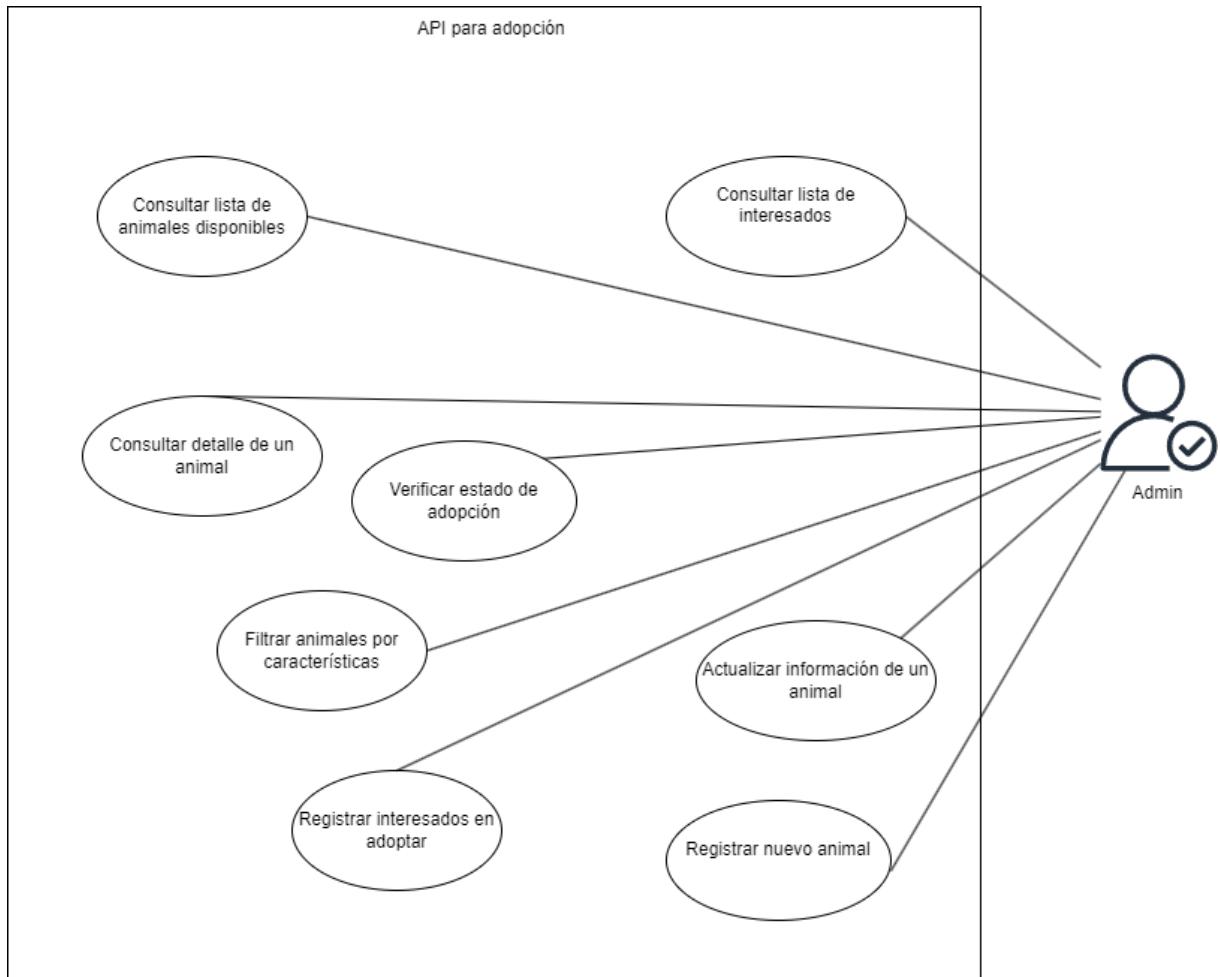


Figura 4-4

Diagrama de casos de uso perfil Admin



4.2 Modelo de Datos

Para este apartado se muestran dos formas de mostrar el funcionamiento de la API para los centros de adopción de mascotas, en primera instancia el diagrama de objetos de la **Figura 4-5** ilustra los datos específicos y las relaciones entre los objetos en un escenario concreto, a continuación, se presenta el diagrama de objetos que modela el sistema de adopción de mascotas.

Entidades clave

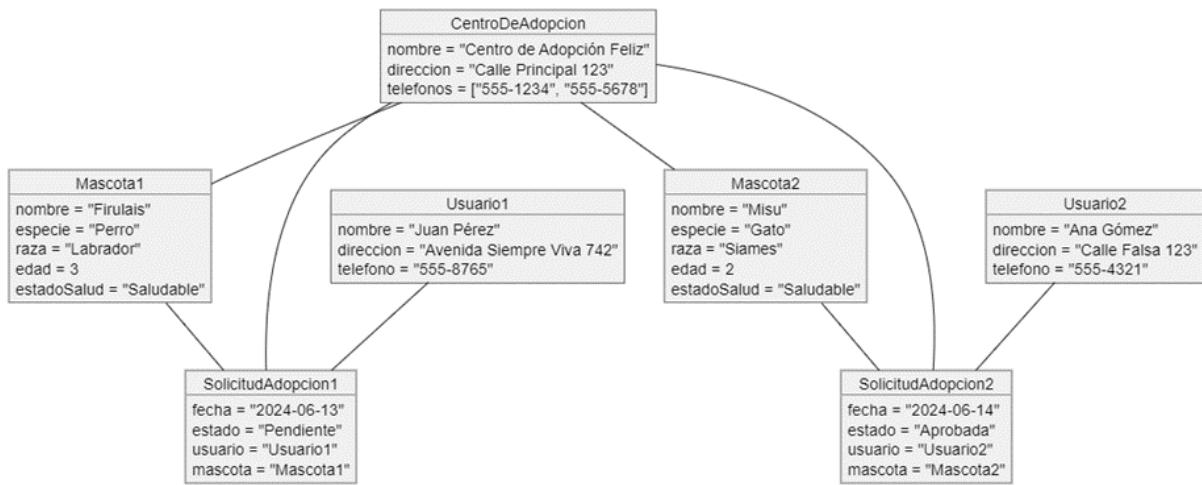
1. Centro de Adopción: Es la entidad que gestiona el proceso de adopción y alberga a las mascotas.
2. Usuario Adoptante: Son las personas interesadas en adoptar una mascota.
3. Mascota: Es la entidad que permite almacenar los datos de las mascotas disponibles para adopción.
4. Solicitud de Adopción: Es el proceso mediante el cual un usuario solicita la adopción de una mascota específica.

Instancias y sus relaciones:

- Centro de Adopción: Representado por una instancia con detalles específicos del centro.
- Usuarios: Dos instancias de usuarios interesados en adoptar mascotas, cada uno con sus datos particulares (adopción de un gato y un perro).
- Mascotas: Dos instancias de mascotas disponibles para adopción, con sus características individuales (un gato y un perro).
- Solicitudes de Adopción: Dos instancias del proceso de solicitud, cada una ligada a un usuario y una mascota específica (adopción aceptada y otra pendiente).

Figura 4-5

Diagrama de objetos del proceso de adopción de mascotas



Con la primera fase de desarrollo de estas entidades y sus funciones básicas se cubrirán las necesidades de los centros de adopción que es la digitalización del proceso de adopción.

En el diagrama de clases de la **Figura 4-6** se describe una representación estructural del sistema de adopción de mascotas. Este diagrama muestra las clases involucradas, sus atributos y métodos, así como las relaciones entre ellas. Este enfoque ayuda a comprender la organización y las responsabilidades de cada clase dentro del sistema, facilitando el desarrollo y mantenimiento del software.

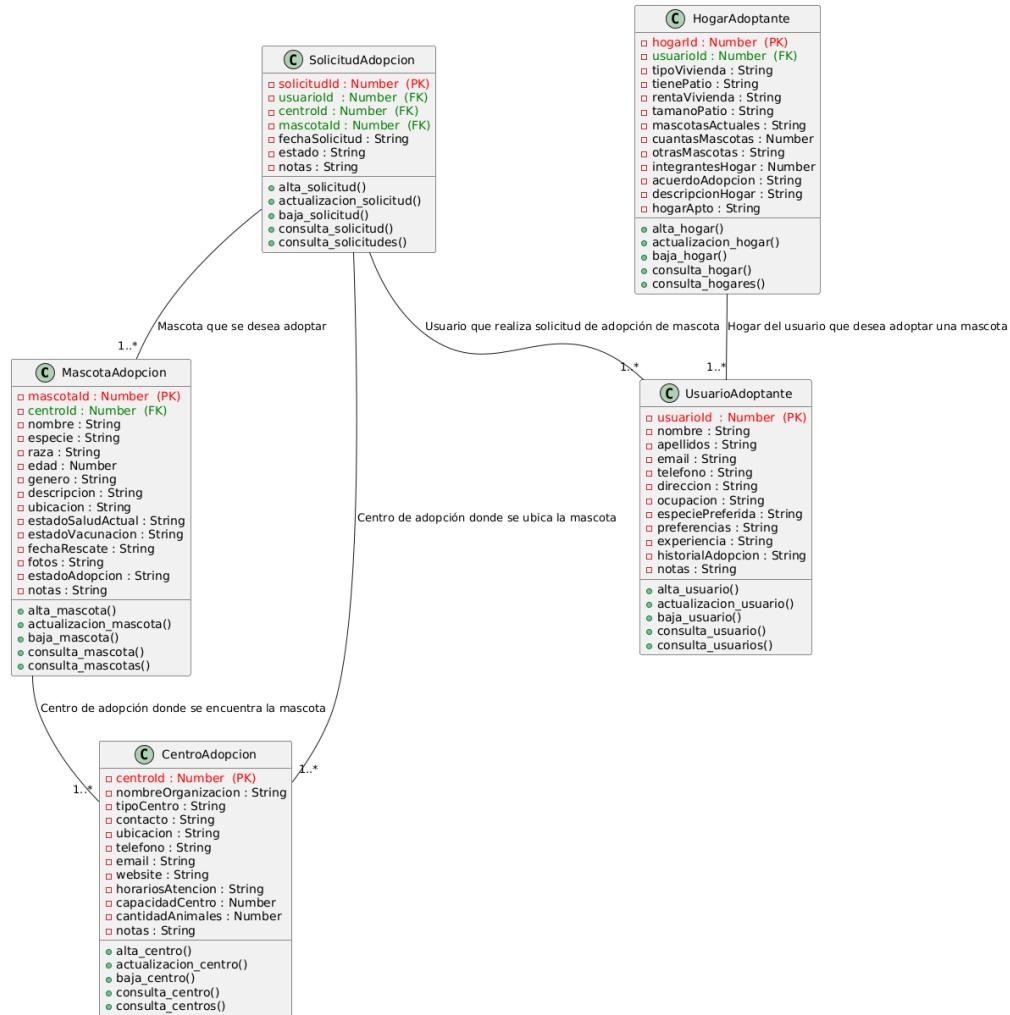
Clases y sus relaciones:

- CentroAdopcion:
 - Clase que representa el centro de adopción
 - Relación de contener múltiples mascotas y procesar múltiples solicitudes de adopción.
- UsuarioAdoptante:
 - Clase que representa a los usuarios interesados en adoptar mascotas.
 - Relación de realizar múltiples solicitudes de adopción de mascotas.

- HogarAdoptante:
 - Clase que representa las características del hogar del usuario adoptante.
 - Relación de tener una descripción del hogar del usuario adoptante.
- MascotaAdopcion:
 - Clase que representa a las mascotas disponibles para adopción.
 - Relación que puede estar involucrada en múltiples solicitudes de adopción.
- SolicitudAdopcion:
 - Clase que representa el proceso de solicitud de adopción.
 - Relación asociada con un usuario y una mascota específicos.

Figura 4-6

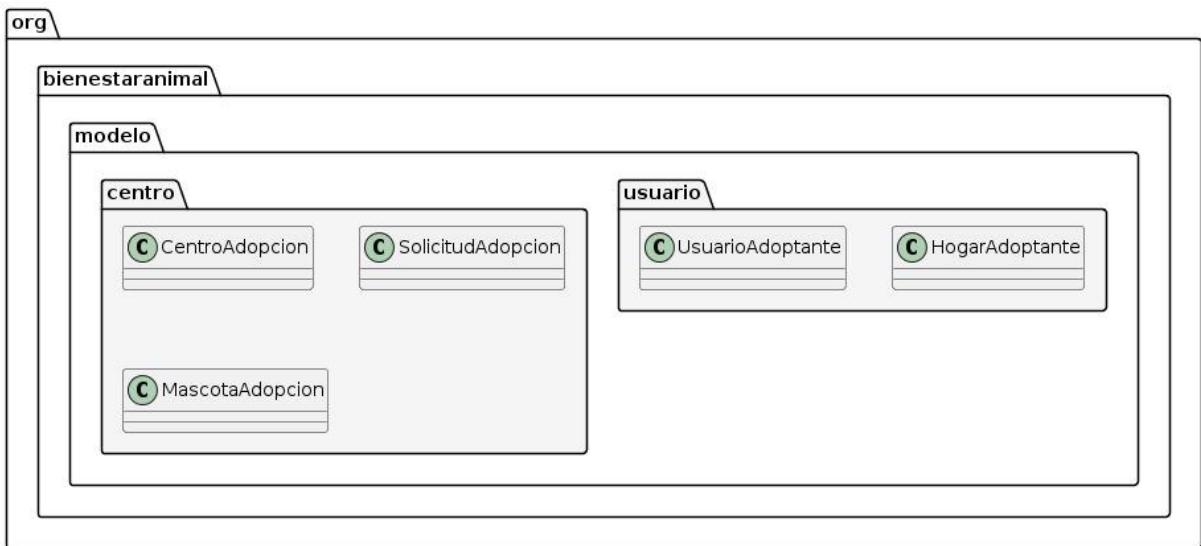
Diagrama de clases del proceso de adopción de mascotas



Finalmente, en el diagrama de paquetes de la **Figura 4-7**, se representa la estructura de paquetes y clases que tendrá este modelo de datos dentro del código.

Figura 4-7

Diagrama de paquetes del módulo de gestión de adopciones



4.3 Arquitectura

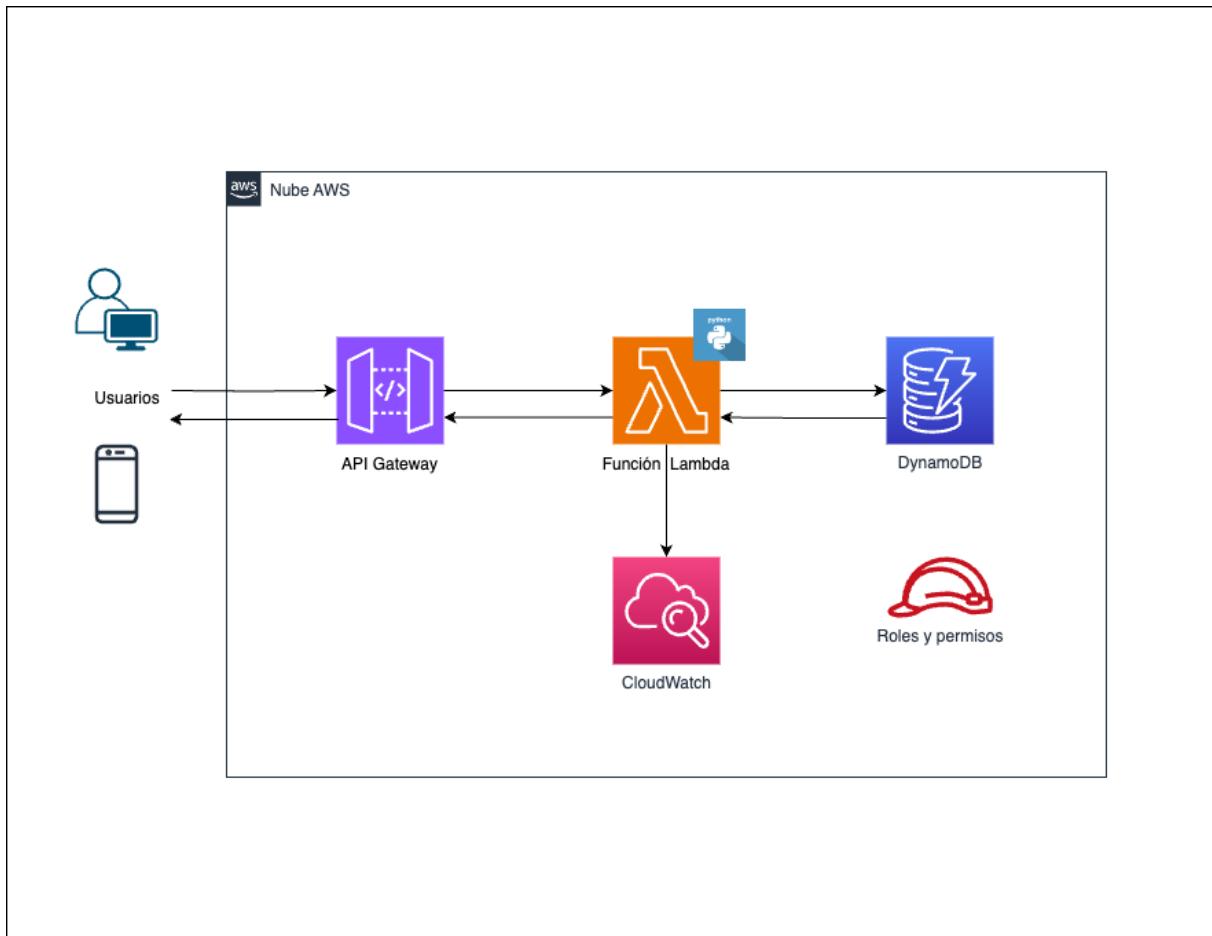
Como se puede observar en el diagrama de arquitectura de la **Figura 4-8**, el módulo de gestión de adopciones es una solución Back-End compuesta de diferentes servicios de AWS integrados entre sí y que residen en la nube de este proveedor.

El flujo de la información inicia y termina con el usuario, el cual lanzará una petición HTTP para consultar, registrar, modificar o eliminar una solicitud de adopción mediante el uso del cliente definido en esta solución, ya sea navegador web o aplicación móvil. Dicho cliente, contará con la URL predefinida para establecer contacto con el componente de API Gateway.

Posteriormente, API Gateway examinará el Path de la URL y los parámetros enviados desde la petición HTTP y con base en la especificación de REST API previamente definida en la solución, decidirá que función Lambda deberá llamar.

Figura 4-8

Arquitectura Back-End del módulo de gestión de adopciones



La función Lambda seleccionada contendrá la lógica de negocio (en lenguaje Python) para la petición HTTP, la cual permitirá uno de los siguientes dos casos:

1. Confirmar la consulta, creación, actualización o eliminación de una adopción junto a su respectiva consulta, creación, actualización o eliminación de registro en la base de datos NoSQL DynamoDB, y su posterior mensaje de confirmación hacia el usuario dentro de la respuesta HTTP, o bien,
2. Rechazar la petición y su correspondiente operación de registro en base de datos NoSQL DynamoDB, indicando el motivo a través de un mensaje de error hacia el usuario dentro de la respuesta HTTP.

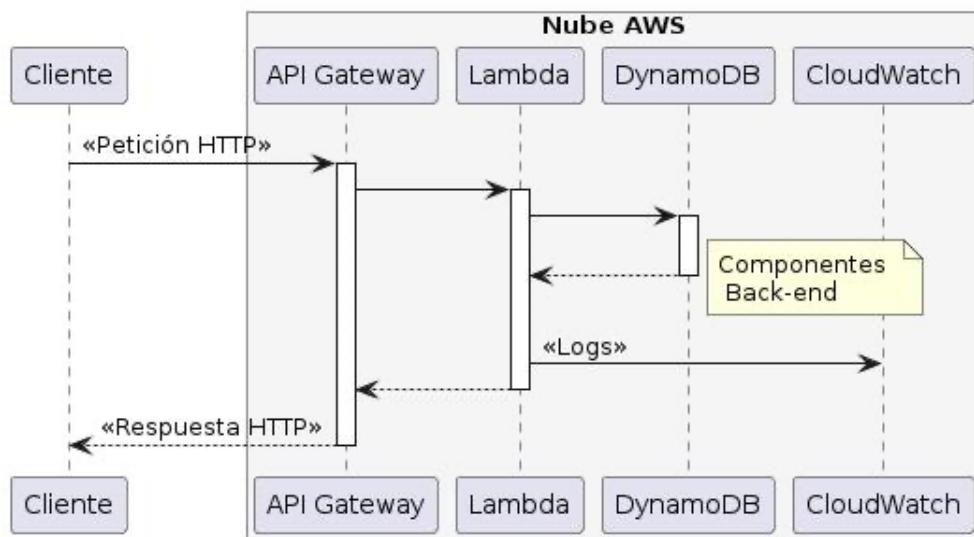
Cabe señalar que, durante este flujo de información, las interacciones de Lambda serán automáticamente registradas en los logs del servicio CloudWatch, lo cual será muy útil para analizar los datos utilizados en cada petición, en caso de que exista algún error.

Además, se deberá de verificar la adecuada configuración de cada uno de los componentes AWS: API Gateway, Lambda y DynamoDB, además de la creación de los roles y permisos necesarios entre ellos para su correcta integración y funcionamiento.

Finalmente, se puede observar una representación más visual del flujo de la información a través del diagrama de secuencia de la siguiente figura.

Figura 4-9

Diagrama de secuencia del módulo de gestión de adopciones



4.4 Desarrollo de interfaces de usuario

De lado del Front-End, se decidió utilizar el framework vue.js y axios para establecer la conectividad con las APIs de AWS.

A continuación, se muestra una parte de lo que está conformado el lado de cliente: las vistas para realizar la captura de una solicitud **Figura 4-10** y **Figura 4-11**, buscar una solicitud **Figura 4-12** y editar una solicitud de adopción en un ambiente de pruebas **Figura 4-13**, corroborando que se ejecutan las acciones de manera correcta **Figura 4-14**.

Figura 4-10

Página principal del proyecto

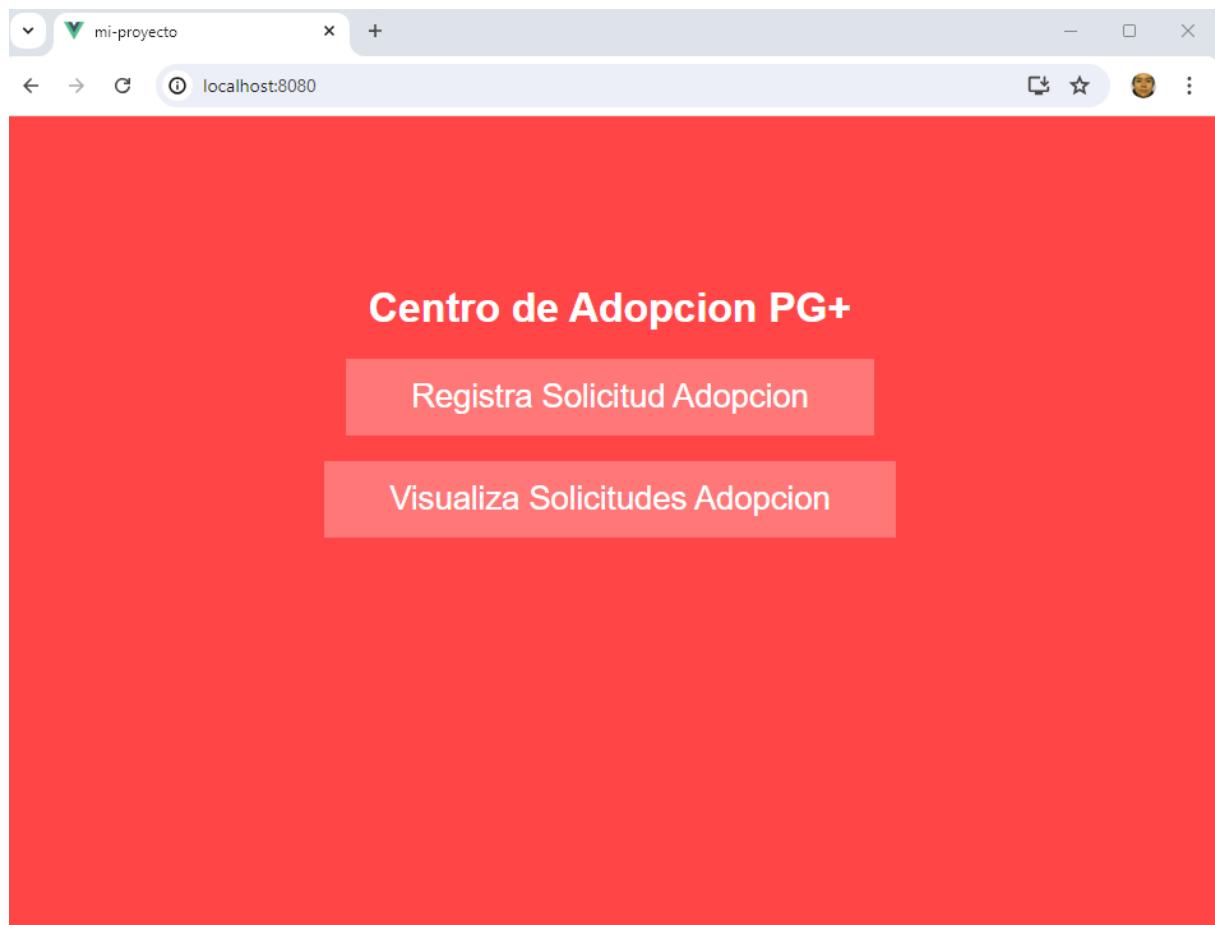


Figura 4-11

Captura de una solicitud de adopción y aviso de envío de datos de manera correcta

The figure consists of two screenshots of a web browser window titled "mi-proyecto". The URL in the address bar is "localhost:8080/insert".

Screenshot 1 (Top): This screenshot shows the "Registro Solicitud de Adopcion" form. The fields filled are:

- Nombre adoptante: Carlos Ruiz
- Centro Adopcion: CA - CDMX
- Nombre Animal: Perro Toby
- Notas: Alérgico al polen

Below the form are two buttons: "Inicio" and "Enviar".

Screenshot 2 (Bottom): This screenshot shows the same form after submission. A modal dialog box is displayed with the title "Solicitud Enviada" and the message "Tu solicitud ha sido enviada correctamente." At the bottom of the dialog is a "Cerrar" button.

Figura 4-12

Visor de solicitudes de adopción

Folio	Nombre Adoptante	Centro Adopción	Nombre Animal	Estatus	Notas	Acciones
001	Juan Pérez	CA - Central Abastos	Lalo - (Perro)	Pendiente	Ninguna	<button>Editar</button>
002	María López	CA - PG+	Lia - (Gata)	Aprobada	Vacunada	<button>Editar</button>
003	Carlos Ruiz	CA - CDMX	Toby - (Perro)	Rechazada	Alérgico al polen	<button>Editar</button>

Figura 4-13

Edición de una solicitud de adopción y aviso de edición de datos de manera correcta

Editar Solicitud

Nombre adoptante:

Centro Adopción:

Nombre Animal:

Estatus:

Notas:

Inicio Guardar

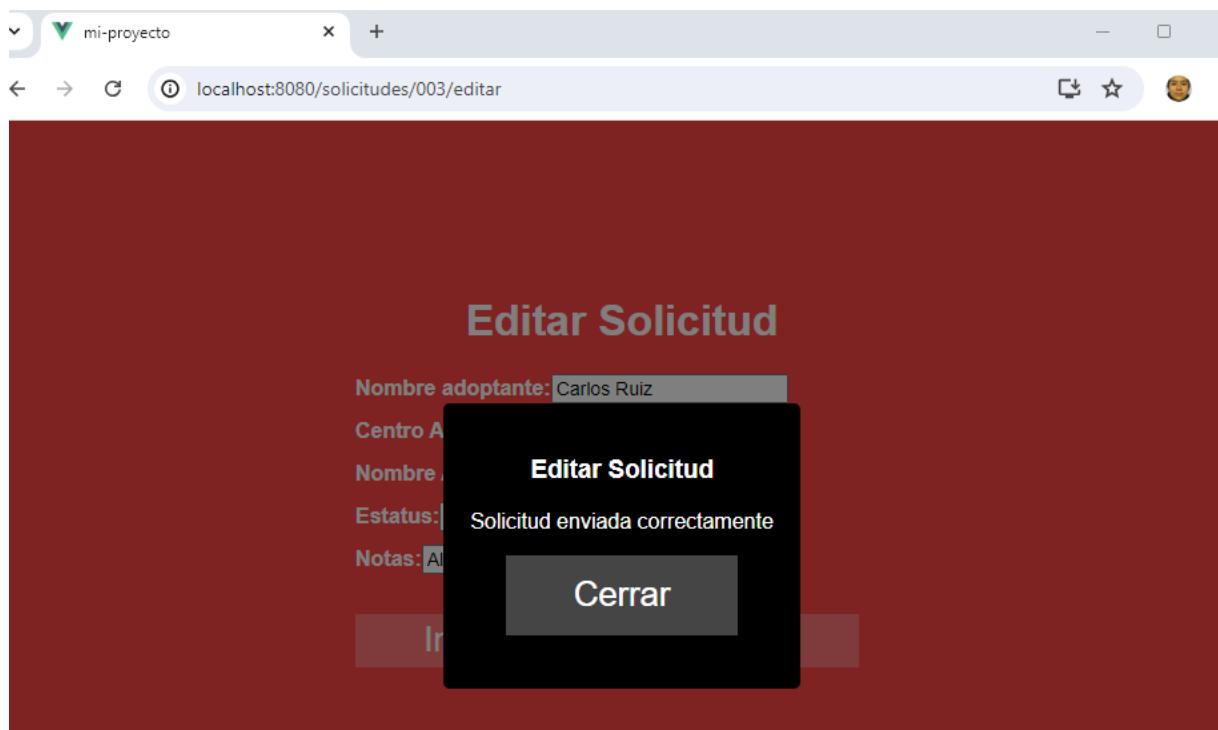


Figura 4-14

Visor con edición de una solicitud de adopción

Folio	Nombre Adoptante	Centro Adopción	Nombre Animal	Estatus	Notas	Acciones
001	Juan Pérez	CA - Central Abastos	Lalo - (Perro)	Pendiente	Ninguna	<button>Editar</button>
002	Maria López	CA - PG+	Lia - (Gata)	Aprobada	Vacunada	<button>Editar</button>
003	Carlos Ruiz	CA - CDMX	Toby - (Perro)	Aprobada	Alérgico al polen	<button>Editar</button>

4.5 Pruebas unitarias

Para las pruebas unitarias del lado del Front-End, se realiza la configuración del ambiente de pruebas en el proyecto (Figura 4-15), donde se deberán instalar los frameworks necesarios para que, acorde a la tecnología que se utiliza (Vue.js y Axios), se validen cada uno de los apartados desarrollados (Figura 4-16). En esta ocasión, solo se muestra la validación de UpdatePage.spec.js, que ayudará a validar:

1. Configuración del entorno de pruebas.
2. Prueba de renderizado: verifica que el componente se renderiza correctamente.
3. Prueba del título: verifica que el título se renderiza correctamente.
4. Prueba del campo deshabilitado: verifica que el campo nameUser está deshabilitado.
5. Prueba de actualización de datos: verifica que los datos de la solicitud se actualizan correctamente al enviar el formulario.
6. Prueba de navegación: verifica que la navegación a /solicitudes se realiza correctamente después de enviar el formulario.

Figura 4-15

Configuración del ambiente de pruebas unitarias en el proyecto

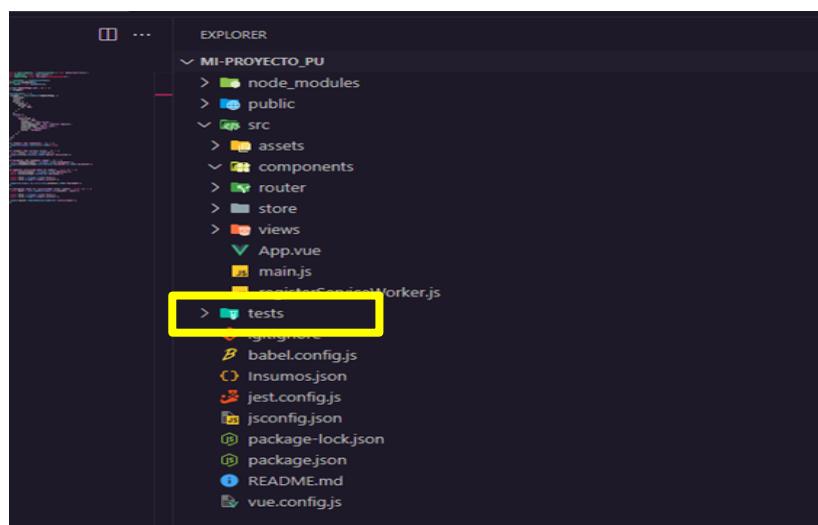
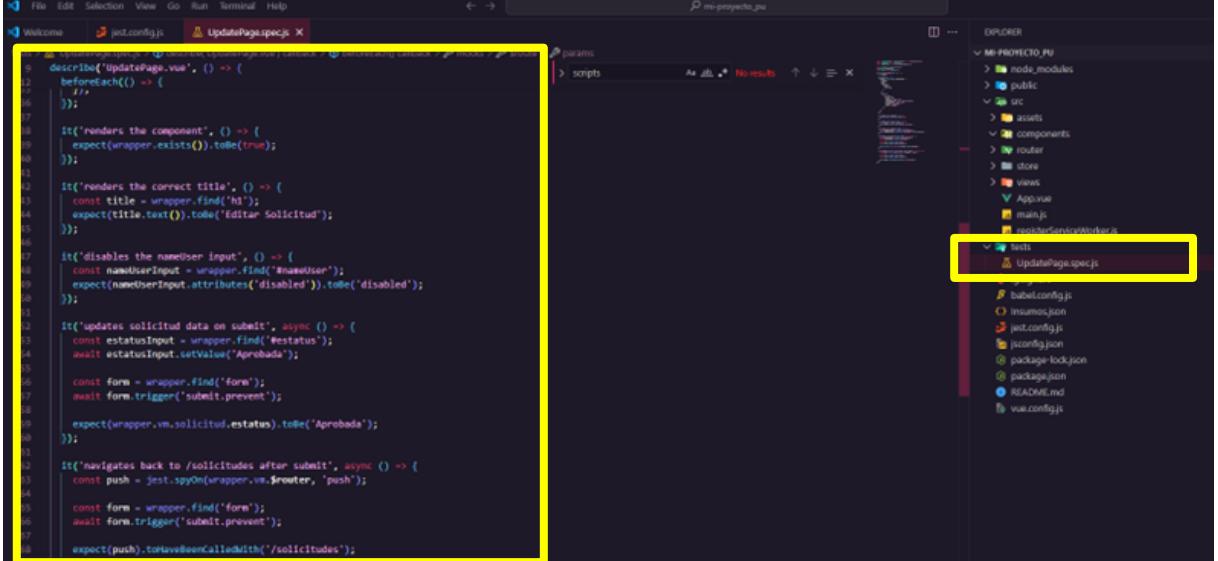


Figura 4-16

Elementos para la ejecución de pruebas unitarias *UpdatePage.vue*



```
describe('UpdatePage.vue', () => {
  beforeEach(() => {
    ...
  });

  it('renders the component', () => {
    expect(wrapper.exists()).toBe(true);
  });

  it('renders the correct title', () => {
    const title = wrapper.find('h1');
    expect(title.text()).toBe('Editar Solicitud');
  });

  it('disables the nameUser input', () => {
    const nameUserInput = wrapper.find('#nameUser');
    expect(nameUserInput.attributes('disabled')).toBe('disabled');
  });

  it('updates solicitud data on submit', async () => {
    const estatusInput = wrapper.find('#estatus');
    await estatusInput.setValue('Aprobada');

    const form = wrapper.find('form');
    await form.trigger('submit.prevent');

    expect(wrapper.vm.solicitud.estatus).toBe('Aprobada');
  });

  it('navigates back to /solicitudes after submit', async () => {
    const push = jest.spyOn(wrapper.vm.$router, 'push');

    const form = wrapper.find('form');
    await form.trigger('submit.prevent');

    expect(push).toHaveBeenCalledWith('/solicitudes');
  });
});
```

Por otro lado, para las pruebas unitarias del Back-End, cabe señalar que dentro de la función Lambda reside toda la lógica de enrutamiento de las APIs y que sus pruebas sólo podrán realizarse sobre la infraestructura AWS.

Para este propósito, se hará uso de la herramienta Postman y se definirá una nueva colección que incluya las 25 APIs de esta solución.

Una vez creada la colección, de su menú emergente se elegirá la opción “Generate tests”. Esto abrirá una nueva pestaña donde se crearán las plantillas de prueba para cada una de las APIs, las cuales se podrán modificar de acuerdo con las condiciones que se deseen probar, y por último se seleccionará la opción “Run Collection” de esta pestaña.

Figura 4-17

Generación automática de pruebas en Postman

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with a collection named 'BienestarAnimal_ServerlessApp'. A context menu is open over this collection, with the option 'Generate tests' highlighted. The main workspace shows a table with requests and their corresponding test cases. The requests listed are:

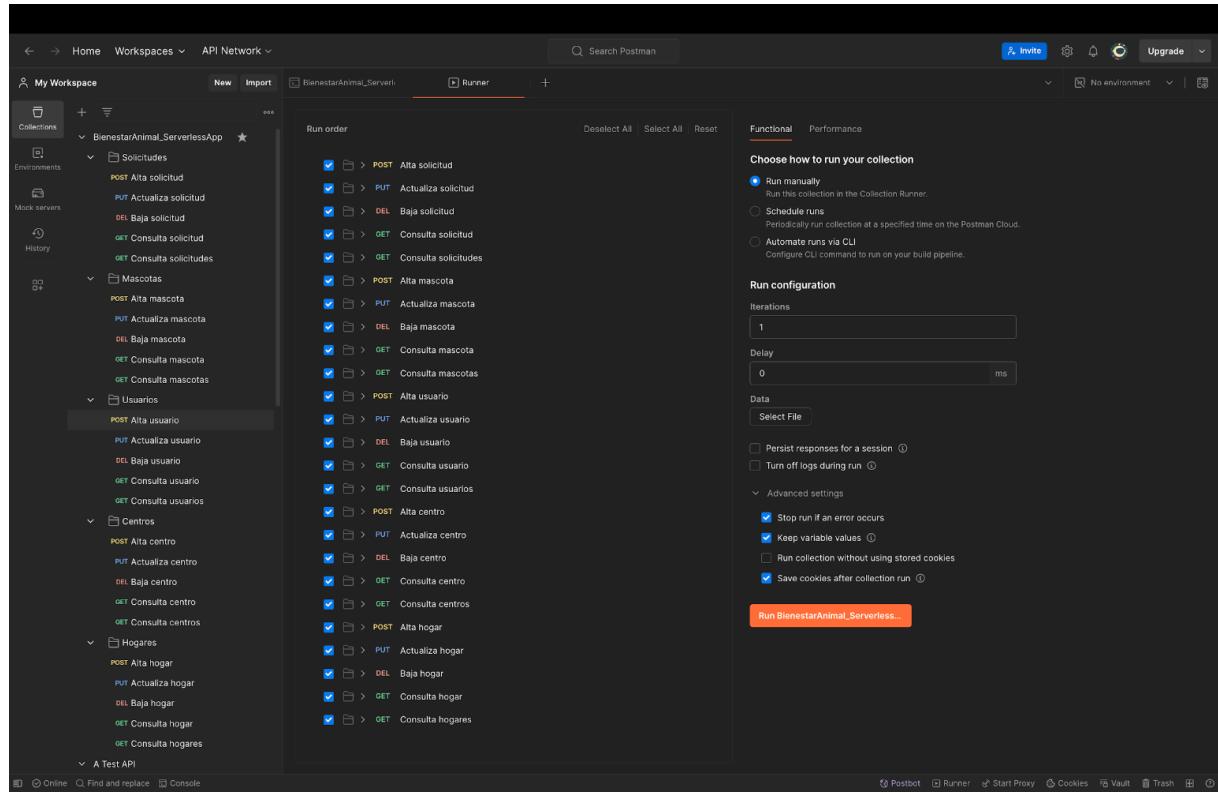
- POST Alta solicitud
- PUT Actualiza solicitud
- PUT Baja solicitud
- GET Consulta solicitud
- GET Consulta solicitudes
- POST Alta mascota
- PUT Actualiza mascota

For each request, there is a section titled with the method and endpoint, followed by a sub-section with a status code assertion (e.g., 'Status code is 200') and its associated JavaScript code snippet.

La anterior acción abrirá una nueva pestaña llamada “Runner”, donde se podrán ajustar diferentes parámetros de configuración para las pruebas como su orden de ejecución, número de iteraciones, tiempo de retraso entre cada prueba, carga de datos desde archivos, etc. Una vez realizados los ajustes necesarios, se procede a hacer clic en el botón “Run” de esta pestaña.

Figura 4-18

Configuración de parámetros para pruebas en Postman



Finalmente, se ejecutará secuencialmente cada una de las pruebas, mostrando el código HTTP de su respuesta, su tiempo de ejecución en milisegundos, el resultado general de la prueba (PASS o FAIL) y al final, se mostrará un resumen del tiempo total y promedio de la ejecución de la colección de pruebas.

Figura 4-19

Resultados de la ejecución de pruebas en Postman

The screenshot shows the Postman interface with the 'BienestarAnimal_ServerlessApp' collection selected. The left sidebar lists various API endpoints categorized by resource (Solicitudes, Mascotas, Usuarios, Centros, Hogares). The main panel displays the 'Run results' for a single run. It shows 25 successful tests (all green) with a total duration of 3s 749ms. The results are grouped by method and endpoint, with detailed status codes and response times.

Method	Endpoint	Status	Duration	Size
GET	/Centros	PASS	200 OK	108 ms 755 B
GET	/Centros	PASS	200 OK	104 ms 768 B
POST	/Alta hogar	PASS	201 Created	113 ms 575 B
PUT	/Actualiza hogar	PASS	200 OK	99 ms 696 B
DELETE	/Baja hogar	PASS	200 OK	107 ms 635 B
GET	/Consulta hogar	PASS	200 OK	111 ms 517 B
GET	/Consulta hogares	PASS	200 OK	102 ms 530 B

4.6 Pruebas de integración

Los casos de prueba de esta sesión fueron diseñados en base a los casos de uso previamente definidos.

Caso de Uso 1: Consultar lista de animales disponibles para adopción

Caso de Prueba 1.1: Verificar que se muestra la lista de animales

Descripción: Validar que al realizar una solicitud GET a /api/mascotas, se devuelve una lista de animales con sus características básicas.

Precondición: Debe haber animales registrados en el sistema.

Pasos: Realizar una solicitud GET a /api/mascotas.

Resultado esperado: La respuesta debe contener una lista de objetos de animales con las propiedades nombre, especie, raza, edad, sexo y una breve descripción.

Caso de Prueba 1.2: Verificar que se maneja correctamente una lista vacía

Descripción: Validar que, al no haber animales registrados, la respuesta de la solicitud GET a /api/mascotas sea una lista vacía.

Precondición: No debe haber animales registrados en el sistema.

Pasos: Realizar una solicitud GET a /api/mascotas.

Resultado esperado: La respuesta debe ser una lista vacía.

Caso de Uso 2: Consultar detalle de un animal

Caso de Prueba 2.1: Verificar que se muestra el detalle de un animal existente

Descripción: Validar que al realizar una solicitud GET a /api/mascotas?mascotaid={id} con un ID válido, se devuelve la información detallada del animal.

Precondición: Debe haber un animal registrado con el ID especificado.

Pasos: Realizar una solicitud GET a /api/mascotas?mascotaid={id} con un ID válido.

Resultado esperado: La respuesta debe contener la información detallada del animal incluyendo historia médica, comportamiento, fotos y videos.

Caso de Prueba 2.2: Verificar que se maneja correctamente un ID inválido

Descripción: Validar que al realizar una solicitud GET a /api/mascotas?mascotaid={id} con un ID inválido, se devuelve un mensaje de error apropiado.

Precondición: El ID especificado no debe corresponder a ningún animal registrado.

Pasos: Realizar una solicitud GET a /api/mascotas?mascotaid={id} con un ID inválido.

Resultado esperado: La respuesta debe ser un mensaje de error indicando que el animal no existe.

Caso de Uso 3: Filtrar animales por características

Caso de Prueba 3.1: Verificar que se puede filtrar por especie

Descripción: Validar que al aplicar un filtro de especie en la solicitud GET a /api/mascotas, se devuelve una lista de animales que cumplen con ese filtro.

Precondición: Debe haber animales de diferentes especies registrados en el sistema.

Pasos: Realizar una solicitud GET a /api/mascotas?especie=Perro.

Resultado esperado: La respuesta debe contener una lista de animales de la especie "Perro".

Caso de Prueba 3.2: Verificar que se puede filtrar por múltiples características

Descripción: Validar que al aplicar múltiples filtros en la solicitud GET a /api/mascotas, se devuelve una lista de animales que cumplen con todos los filtros.

Precondición: Debe haber animales registrados que cumplan con las características especificadas.

Pasos: Realizar una solicitud GET a

/api/mascotas?especie=Perro&edad=Joven&tamano=Mediano.

Resultado esperado: La respuesta debe contener una lista de animales que sean perros, jóvenes y de tamaño mediano.

Caso de Uso 4: Registrar nuevo animal

Caso de Prueba 4.1: Verificar que se puede registrar un nuevo animal

Descripción: Validar que al realizar una solicitud POST a /api/mascotas con la información completa de un animal, se registra correctamente el nuevo animal.

Precondición: La información del animal debe ser válida y completa.

Pasos: Realizar una solicitud POST a /api/mascotas con los datos del nuevo animal.

Resultado esperado: La respuesta debe ser una confirmación del registro y los detalles del animal registrado.

Caso de Prueba 4.2: Verificar que se maneja correctamente una solicitud con datos incompletos

Descripción: Validar que al realizar una solicitud POST a /api/mascotas con datos incompletos, se devuelve un mensaje de error apropiado.

Precondición: Los datos del animal proporcionados deben estar incompletos.

Pasos: Realizar una solicitud POST a /api/mascotas con datos incompletos.

Resultado esperado: La respuesta debe ser un mensaje de error indicando que la información proporcionada es insuficiente.

Caso de Uso 5: Actualizar información de un animal

Caso de Prueba 5.1: Verificar que se puede actualizar la información de un animal

Descripción: Validar que al realizar una solicitud PUT a /api/mascotas?mascotId={id} con información actualizada, se actualiza correctamente la información del animal.

Precondición: Debe haber un animal registrado con el ID especificado y la información proporcionada debe ser válida.

Pasos: Realizar una solicitud PUT a /api/mascotas?mascotId={id} con la información actualizada del animal.

Resultado esperado: La respuesta debe ser una confirmación de la actualización.

Caso de Prueba 5.2: Verificar que se maneja correctamente una solicitud con ID inválido

Descripción: Validar que al realizar una solicitud PUT a /api/mascotas?mascotId={id} con un ID inválido, se devuelve un mensaje de error apropiado.

Precondición: El ID especificado no debe corresponder a ningún animal registrado.

Pasos: Realizar una solicitud PUT a /api/mascotas?mascotId={id} con un ID inválido.

Resultado esperado: La respuesta debe ser un mensaje de error indicando que el animal no existe.

Caso de Uso 6: Registrar interesados en adoptar

Caso de Prueba 6.1: Verificar que se puede registrar un interesado en adoptar

Descripción: Validar que al realizar una solicitud POST a /api/usuarios con la información del interesado, se registra correctamente el interesado.

Precondición: La información del interesado debe ser válida y completa.

Pasos: Realizar una solicitud POST a /api/usuarios con los datos del interesado.

Resultado esperado: La respuesta debe ser una confirmación del registro y los detalles del interesado.

Caso de Prueba 6.2: Verificar que se maneja correctamente una solicitud con datos incompletos

Descripción: Validar que al realizar una solicitud POST a /api/usuarios con datos incompletos, se devuelve un mensaje de error apropiado.

Precondición: Los datos del interesado proporcionados deben estar incompletos.

Pasos: Realizar una solicitud POST a /api/usuarios con datos incompletos.

Resultado esperado: La respuesta debe ser un mensaje de error indicando que la información proporcionada es insuficiente.

Caso de Uso 7: Consultar lista de interesados

Caso de Prueba 7.1: Verificar que se muestra la lista de interesados

Descripción: Validar que al realizar una solicitud GET a /api/usuarios, se devuelve una lista de interesados con sus características básicas.

Precondición: Debe haber interesados registrados en el sistema.

Pasos: Realizar una solicitud GET a /api/usuarios.

Resultado esperado: La respuesta debe contener una lista de objetos de interesados con las propiedades nombre y contacto.

Caso de Uso 8: Verificar estado de adopción

Caso de Prueba 8.1: Verificar que se puede consultar el estado de una adopción

Descripción: Validar que al realizar una solicitud GET a /api/solicitudes?solicitudId={id} con un ID válido, se devuelve el estado de la solicitud de adopción.

Precondición: Debe haber una solicitud de adopción registrada con el ID especificado.

Pasos: Realizar una solicitud GET a /api/solicitudes?solicitudId={id} con un ID válido.

Resultado esperado: La respuesta debe contener los detalles del estado de la solicitud de adopción.

Caso de Prueba 8.2: Verificar que se maneja correctamente una solicitud con ID inválido

Descripción: Validar que al realizar una solicitud GET a /api/solicitudes?solicitudId={id} con un ID inválido, se devuelve un mensaje de error apropiado.

Precondición: El ID especificado no debe corresponder a ninguna solicitud de adopción registrada.

Pasos: Realizar una solicitud GET a /api/solicitudes?solicitudId={id} con un ID inválido.

Resultado esperado: La respuesta debe ser un mensaje de error indicando que la solicitud de adopción no existe.

4.7 Pruebas de usabilidad

Siendo el objetivo principal la implementación de un módulo de gestión de adopciones para centros de bienestar animal, y con el fin de mejorar la calidad y claridad del proceso de adopción, se cuentan con los siguientes objetivos de usabilidad:

1. Desarrollar una aplicación fácil de integrar con sistemas existentes para gestionar el proceso de adopción de animales en situación de calle.
2. Establecer un canal que permita compartir la información de los animales disponibles para adoptar de forma clara y detallada.
3. Clasificar los animales a ser dados en adopción por sus características permitiendo así que los interesados puedan filtrar sus datos.

A continuación, se presentan los objetivos que la metodología adoptada para este proyecto, que detallaremos uno por uno.

1. Establecer un canal único para la adopción de mascotas de manera clara y detallada para su mejor comprensión y uso.

Con el fin de tener una mejor experiencia para el usuario el objetivo es mantener un único canal, es decir un solo sistema en el que el mismo se pueda registrar solicitudes, visualizar datos de las mascotas, así mismo como del solicitante.

2. Aplicación fácil de integrar con sistemas existentes para la gestión de la adopción.

La flexibilidad y adaptabilidad que tiene la aplicación para integrarse a otros sistemas existentes si es que el desarrollo a futuro lo requiere.

3. Registro de datos amigables con el usuario.

Contar con un sistema amigable y fácil de entender para los solicitantes de adopción, así como para los administradores de este, es un objetivo importante.

Como resultado de estas premisas, se obtiene un desarrollo con sistema amigable con los solicitantes y administradores. También se observa que el apartado del registro de solicitudes es fácil y sencillo de utilizar, y la información que se muestra para el administrador también es clara y concisa.

Por todo lo antes mencionado, se puede concluir que existe un único canal de adopción de mascotas donde el solicitante puede obtener la información necesaria para registrar sus solicitudes, demostrando que se cumplen con los objetivos de usabilidad planteados, así como la optimización de los procesos, la mejora continua, la calidad y mantenibilidad de un sistema amigable y fácil de utilizar el registro, evaluación y seguimiento de mascotas en adopción en México.

4.8 Análisis de ciberseguridad

En cuanto a las pruebas de ciberseguridad para este desarrollo, se plantean diferentes estrategias y la utilización de diversas herramientas a diferentes niveles de implementación, como por ejemplo el uso de técnicas de codificación segura, la validación de datos y parámetros de entrada tanto en interfaz de usuario como dentro de la lógica de negocio en las APIs, la implementación de un mecanismo de autenticación basado en token para permitir la comunicación entre las APIs, etc.

Dichas técnicas se plantean para las fases posteriores de este desarrollo, no obstante, a continuación se abordan a mayor detalle dos de ellas.

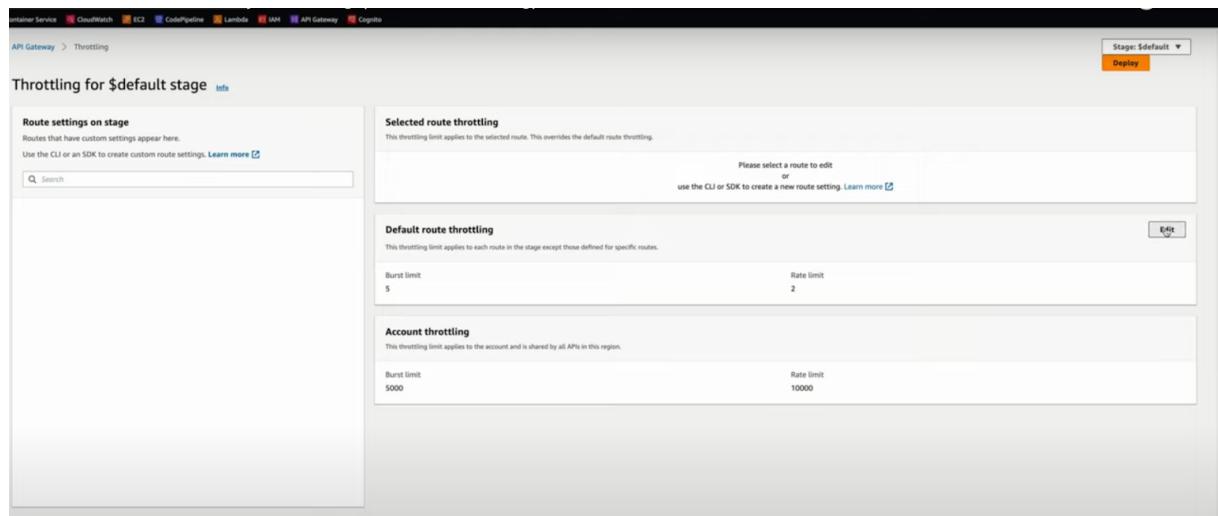
Para el Back-End, se plantea la habilitación de un mecanismo que establece límites en las cuotas de solicitudes HTTP, llamado API Throttling.

Este mecanismo es proporcionado por el servicio AWS API Gateway, y una de sus principales funciones es evitar la sobre utilización de las APIs, por ejemplo, durante un ataque DoS (Denial of Service).

Para habilitarlo, se selecciona la API deseada dentro del servicio de API Gateway, y dentro de la opción Protect > Throttling, se selecciona el Stage apropiado. Posteriormente, se actualizan los límites de “Burst Limit” y “Rate Limit” cuya combinación sea la adecuada a las necesidades de utilización de las APIs y se guardan los cambios. En el ejemplo de la siguiente figura, se utilizan los valores de 5 y 2 respectivamente.

Figura 4-20

Configuración de límites de API Throttling dentro de servicio AWS API Gateway

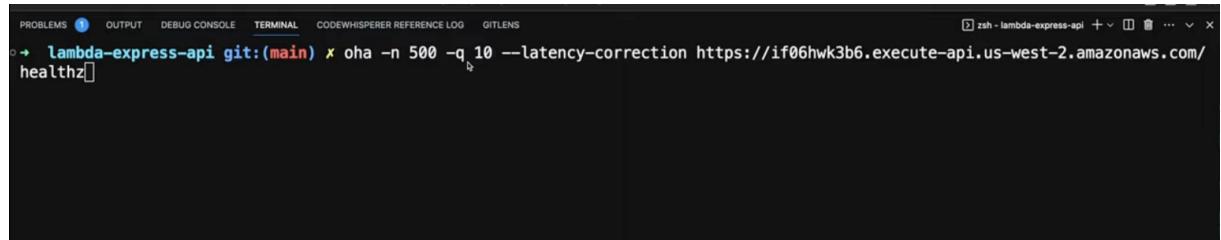


Adaptado de AWS HTTP API Gateway Throttling (API Rate Limiting) [Archivo de Vídeo], por Canal Alex Rusin, 2023, Youtube (<https://www.youtube.com/watch?v=h45UVqAgg-M>). CC BY 2.0

Posteriormente, a través de la herramienta de línea de comandos oha, se realizará una prueba de estrés sobre una de las APIs que tenga habilitada la opción de API Throttling. En el comando oha de ejemplo de la siguiente figura, se realizarán 500 solicitudes a una tasa de 10 solicitudes por segundo.

Figura 4-21

Ejemplo de comando oha para realización de prueba de estrés



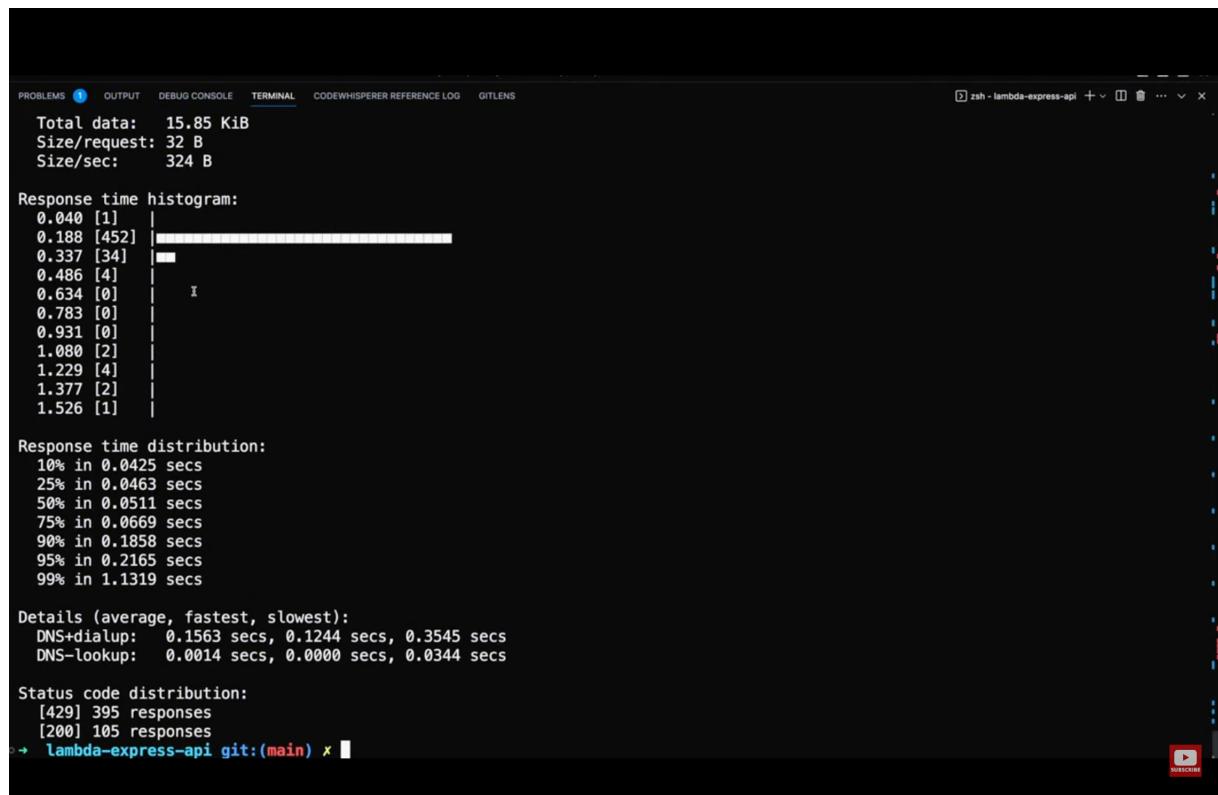
A screenshot of a terminal window from a code editor. The terminal tab is active, showing the command: `lambda-express-api git:(main) ✘ oha -n 500 -q 10 --latency-correction https://if06hwk3b6.execute-api.us-west-2.amazonaws.com/healthz`. The terminal interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, CODEWHISPERER, REFERENCE LOG, and GITLENS. The status bar at the bottom right shows "zsh - lambda-express-api".

Adaptado de *AWS HTTP API Gateway Throttling (API Rate Limiting)* [Archivo de Vídeo], por Canal Alex Rusin, 2023, Youtube (<https://www.youtube.com/watch?v=h45UVqAgg-M>). CC BY 2.0

Finalmente, como resultado de esta prueba de estrés, se mostrarán las estadísticas de los tiempos de respuesta de las solicitudes y la distribución de sus códigos de estado HTTP entre las que fueron satisfactorias (HTTP status 200: OK) y las que fueron rechazadas (HTTP status 429: Too Many Requests).

Figura 4-22

Ejemplo de resultados de prueba de estrés



The screenshot shows a terminal window with the following output:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL CODEWHISPERER REFERENCE LOG GITLENS
Total data: 15.85 KiB
Size/request: 32 B
Size/sec: 324 B

Response time histogram:
0.040 [1]
0.188 [452] ━━━━━━
0.337 [34] ━
0.486 [4]
0.634 [0]
0.783 [0]
0.931 [0]
1.080 [2]
1.229 [4]
1.377 [2]
1.526 [1]

Response time distribution:
10% in 0.0425 secs
25% in 0.0463 secs
50% in 0.0511 secs
75% in 0.0669 secs
90% in 0.1858 secs
95% in 0.2165 secs
99% in 1.1319 secs

Details (average, fastest, slowest):
DNS+dialup: 0.1563 secs, 0.1244 secs, 0.3545 secs
DNS-lookup: 0.0014 secs, 0.0000 secs, 0.0344 secs

Status code distribution:
[429] 395 responses
[200] 105 responses
λ lambda-express-api git:(main) ✘
```

Adaptado de AWS HTTP API Gateway Throttling (API Rate Limiting) [Archivo de Vídeo], por Canal Alex Rusin, 2023, Youtube (<https://www.youtube.com/watch?v=h45UVqAgg-M>). CC BY 2.0

Por otro lado, para el Front-End (aunque también es posible su aplicación en el Back-End), se plantea la utilización de herramientas de análisis de código estático (SAST) durante la etapa de codificación de este proyecto.

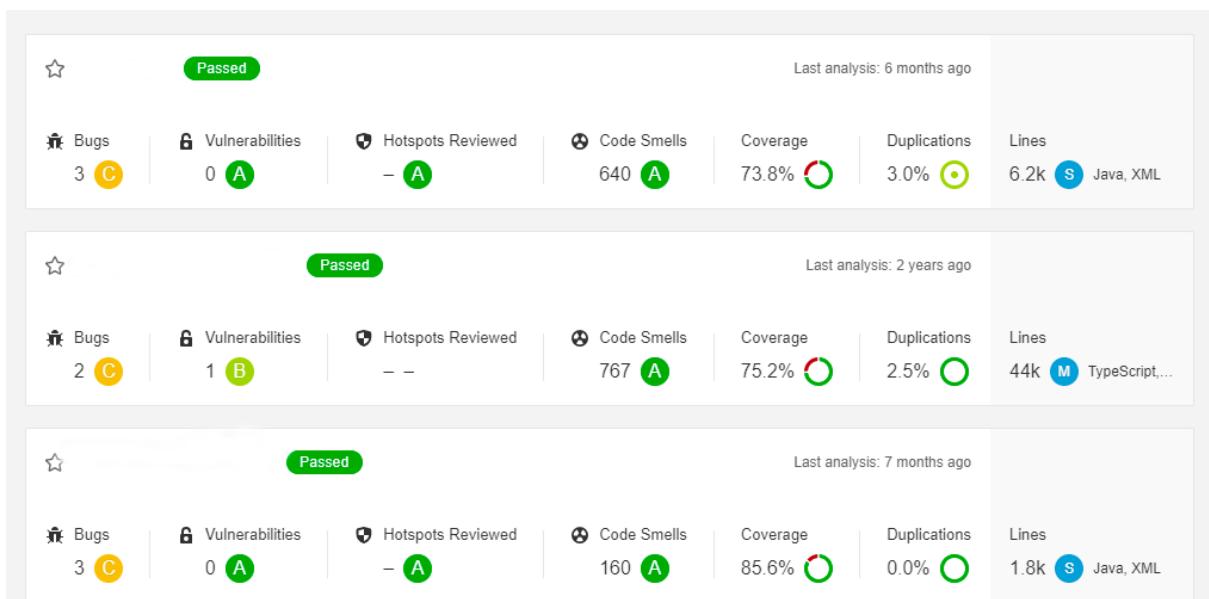
SonarQube es una plataforma de código abierto que, mediante el uso de estos analizadores de código estático, se encarga de inspeccionar la calidad del código fuente de un proyecto, detectando posibles problemas y vulnerabilidades, y proporcionando información adicional para poder ser corregidos posteriormente por los desarrolladores.

Además de ser ofrecida como un servicio en la nube (SonarCloud), otra alternativa que esta plataforma ofrece es la de poder ser instalada localmente.

Una vez que SonarQube es iniciada localmente, sobre esta se podrán ejecutar análisis de proyectos locales en hasta 19 tipos diferentes de lenguajes (incluidos JavaScript, Python, HTML, Java, etc.). Seguidamente, se podrán verificar sus resultados en cada uno de los apartados del reporte generado.

Figura 4-23

Ejemplo de resultados de pruebas de análisis en SonarQube



Adaptado de *Qué es SonarQube: Verifica y analiza la calidad de tu código* [Web], por Sentrio Labs, 2021, Sentrio (<https://sentrio.io/blog/que-es-sonarqube/>). Todos los derechos reservados 2021 por Sentrio Labs.

5. Conclusiones y trabajo futuro

5.1. Conclusiones

El desarrollo del sistema Serverless para la gestión de adopciones tuvo un progreso significativo, mostrando mejoras prometedoras en la eficiencia operativa de los centros de adopción durante la fase de pruebas internas. A continuación, se detallan los resultados obtenidos:

Pruebas Funcionales

Las pruebas funcionales del sistema se llevaron a cabo para asegurar que todas las funcionalidades clave se ejecutaran según lo esperado. Estas pruebas incluyeron la creación, consulta, actualización y eliminación de registros de animales y adoptantes. Se verificó que el sistema manejara correctamente las solicitudes HTTP a través de API Gateway, garantizando respuestas rápidas y precisas.

Pruebas de Integración

Se realizaron pruebas de integración para evaluar la interacción entre los diferentes componentes del sistema. Las pruebas confirmaron que AWS Lambda, API Gateway y DynamoDB trabajan juntos de manera cohesiva. Los endpoints para la consulta de animales disponibles, el registro de nuevos animales y la actualización de información funcionaron sin problemas, demostrando la robustez de la arquitectura Serverless.

Monitoreo y Logística

La implementación de AWS CloudWatch permitió un monitoreo detallado del rendimiento del sistema. Se configuraron dashboards para visualizar métricas críticas como el tiempo de respuesta, el número de solicitudes procesadas y la tasa de errores. Las alertas configuradas en CloudWatch ayudaron a identificar y resolver problemas potenciales de manera proactiva, antes de que afectaran al usuario final.

Análisis de Desempeño

El análisis del desempeño mostró que el sistema es capaz de manejar un alto volumen de solicitudes concurrentes sin degradación significativa en el rendimiento. Durante las pruebas de carga, el sistema mantuvo un tiempo de respuesta inferior a 200 milisegundos para la mayoría de las consultas, demostrando su capacidad para escalar eficientemente durante períodos de alta demanda.

Discusión

Los resultados obtenidos subrayan la importancia de las soluciones digitales en la gestión de adopciones. La adopción de tecnologías Serverless y metodologías ágiles facilitó la adaptación a cambios y la entrega continua de valor. Comparando con estudios previos, este enfoque mostró mejoras significativas en eficiencia y efectividad.

El uso de Kanban como metodología de desarrollo demostró ser particularmente beneficioso en este contexto. La visualización del trabajo en curso y la capacidad de ajustar rápidamente las prioridades permitieron al equipo de desarrollo responder de manera ágil a las necesidades cambiantes del proyecto.

Sin embargo, la dependencia de la infraestructura de AWS presenta algunas limitaciones. Aunque AWS ofrece una alta disponibilidad y escalabilidad, el costo de los servicios puede aumentar significativamente con el tiempo, especialmente si no se optimizan adecuadamente las funciones y recursos utilizados. Además, la necesidad de formación especializada para gestionar y mantener el sistema puede ser una barrera para algunos centros de bienestar animal con recursos limitados.

Comparación con Estudios Previos

Estudios anteriores han explorado diversas tecnologías para la gestión de adopciones, pero pocos han implementado una solución completamente Serverless. La comparación de nuestros resultados con los de estudios que utilizaron arquitecturas tradicionales muestra una clara ventaja en términos de escalabilidad y eficiencia operativa. Además, la flexibilidad y rapidez de implementación del framework Kanban superaron a metodologías más rígidas utilizadas en proyectos similares.

Comentarios finales

La implementación de tecnología Serverless en el módulo de gestión de adopciones ha demostrado ser una solución de Software eficaz para mejorar la eficiencia operativa de los centros de bienestar animal en México. El sistema desarrollado facilita el proceso de adopción desde el registro de animales hasta el seguimiento post adopción, asegurando su transparencia y efectividad. Además, la capacidad de integrar este sistema con plataformas existentes también abre la puerta a futuras mejoras e innovaciones.

El uso de los servicios Serverless AWS Lambda, API Gateway y DynamoDB, junto con la metodología Kanban, permitió crear una solución de Software robusta y escalable para la problemática descrita, no obstante, es preciso señalar que dicha solución técnica puede ser fácilmente adaptada hacia diferentes contextos, problemáticas y/o necesidades, siempre y cuando se utilicen los modelos de datos adecuados.

5.2. Trabajo futuro

El trabajo a futuro para esta solución podría implicar su integración con otros servicios AWS para la realización de propósitos más complejos. Ejemplos de ello serían la integración de AWS Cognito como mecanismo de autenticación y registro para nuevos usuarios, o bien, la automatización de los pasos para la creación de los recursos AWS necesarios para la solución, a través de plantillas de código en SAM (Serverless Application Model).

Por otro lado, futuras investigaciones podrían explorar la integración de tecnologías de inteligencia artificial para mejorar la predicción de adopciones exitosas y la personalización de recomendaciones para adoptantes potenciales. Además, se podría investigar la aplicación de Blockchain para mejorar la trazabilidad y la transparencia en el proceso de adopción.

El análisis de costos y la optimización de recursos en un entorno Serverless también representan áreas de interés para futuras investigaciones. Evaluar el costo-beneficio a largo plazo de estas tecnologías puede proporcionar insights valiosos para organizaciones con recursos limitados.

6. Referencias bibliográficas

6.1. Referencias

Amazon Web Services, Inc. (s.f.). *Throttle requests to your REST APIs for better throughput in API Gateway*. Amazon API Gateway Developer Guide.

<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-request-throttling.html>

Canal Alex Rusin. (16 de junio de 2023). *AWS HTTP API Gateway Throttling (API Rate Limiting)*. [Archivo de Vídeo]. Youtube.

<https://www.youtube.com/watch?v=h45UVqAgg-M>

Castellano, L. (2019). Kanban. Metodología para aumentar la eficiencia de los procesos. *3C Tecnología. Glosas de innovación aplicadas a la pyme*, 8(1), 30-41.

<http://dx.doi.org/10.17993/3ctecno/2019.v8n1e29/30-41>

Celaya, B. (18 de diciembre de 2022). *En México domina el abandono y el maltrato animal*. Aristegui Noticias.

<https://aristeguinoticias.com/mexico/18-dec-2022/en-mexico-domina-el-abandono-y-el-maltrato-animal/>

Gracia, H. y Yanes, O. (2012). Bases de Datos NoSQL. *Revista Telem@tica*, 11(3), 21-33.

<https://revistatelematica.cujae.edu.cu/index.php/tele/article/view/74/74>

Lozano, R. (3 de mayo de 2022). *Iniciativa con proyecto de decreto* [Archivo PDF]. Congreso de la Ciudad de México.

<https://www.congresocdmx.gob.mx/media/documentos/166333e0b2b6a546c0ae82ff6bed20681a7f1615.pdf>

Sandoval, A. (13 de marzo de 2023). *Mascotas en México, un sector invisible para las estadísticas*. Infobae. <https://www.infobae.com/mascotas/2023/03/13/mascotas-en-mexico-un-sector-invisible-para-las-estadisticas/>

Sentrio Labs. (15 de diciembre de 2021). *Qué es SonarQube: Verifica y analiza la calidad de tu código*. Sentrio Labs.

<https://sentrio.io/blog/que-es-sonarqube/>

Susnjara, S. y Smalley, I. (10 de junio de 2024). *What is serverless computing?*. IBM. <https://www.ibm.com/topics/serverless>

Synergy Research Group (1 de febrero de 2024). *Cloud Market Gets its Mojo Back; AI Helps Push Q4 Increase in Cloud Spending to New Highs*. SRGResearch. <https://www.srgresearch.com/articles/cloud-market-gets-its-mojo-back-q4-increase-in-cloud-spending-reaches-new-highs>

Yépez, E. y Armijos, K. (2020). *Aplicación de la metodología Kanban en el desarrollo del software para generación, validación y actualización de reactivos, integrado al sistema informático de control académico UNACH* [Tesis de Bachiller]. Universidad Nacional de Chimborazo.

<http://dspace.unach.edu.ec/handle/51000/6457>

6.2. Bibliografía

Adoptando un Amigo (s.f.). *Adopción de perros y gatos*.

<https://adoptandounamigo.mx/>

Alamilla, L., Pérez, V., Sosa, S., y Valentín, J. (2021).

Arquitectura REST para el desarrollo de aplicaciones web empresariales.

Revista Electrónica sobre Tecnología, Educación y Sociedad, 8(15).

<https://www.ctes.org.mx/index.php/ctes/article/view/748/909>

Amazon Web Services, Inc. (s.f.). *AWS Support: Acelere los resultados empresariales en la nube*. Amazon Web Services.

https://aws.amazon.com/es/premiumsupport/?nc2=h_m_bc

Amazon Web Services, Inc. (s.f.). *Nivel Gratuito de AWS: Adquiera experiencia práctica y gratuita con los productos y servicios de AWS*. Amazon Web Services.

<https://aws.amazon.com/es/free/>

Antiñanco, M. (9 de junio de 2014). Bases de Datos NoSQL: Escalabilidad y alta disponibilidad a través de patrones de diseño [Trabajo de especialización].

Universidad Nacional de La Plata.

<https://sedici.unlp.edu.ar/handle/10915/36338>

Canal Tech with Hitch. (6 de noviembre de 2023). *Build a REST API (CRUD) with AWS Lambda, API Gateway & DynamoDB Using Python | Step-by-Step Guide* [Archivo de Vídeo].

Youtube.

<https://www.youtube.com/watch?v=7bgUF6YESxA>

Castillo, M. y Guaña, E. (2024). Kanban: Una metodología ágil para la gestión eficiente del

flujo de trabajo en el desarrollo de software, una revisión sistemática.

Revista Ingenio global, 3(1), 17-28.

<https://doi.org/10.62943/rig.v3n1.2024.68>

Ellis, F. (20 de julio de 2023). *AWS Certified Developer - Associate (DVA-C02)* [Curso en línea].

Pluralsight / A Cloud Guru.

<https://www.pluralsight.com/cloud-guru/courses/aws-certified-developer-associate-dva-c02>

Gobierno de la Ciudad de México (s.f.). *Bienestar animal* [Archivo PDF].

Secretaría del Medio Ambiente.

<https://www.sedema.cdmx.gob.mx/storage/app/media/Cuadernillos-Ambientales/5-BIENESTAR-ANIMAL.pdf>

León, A., y Checa, M. (2022). Uso de tableros Kanban como apoyo para el desarrollo de las metodologías ágiles. *Universidad y Sociedad*, 14(S2), 208-214.

<https://rus.ucf.edu.cu/index.php/rus/article/view/2760>

Mell, P. y Grance, T. (septiembre de 2011). *The NIST Definition of Cloud Computing* [Archivo PDF]. National Institute of Standards and Technology.

<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf>

Molina, J., Honores, J., Pedreira-Souto, N., y Pardo, H. (2021). Estado del arte: metodologías de desarrollo de aplicaciones móviles.

3C Tecnología. Glosas de innovación aplicadas a la pyme, 10(2), 17-45.

<https://doi.org/10.17993/3ctecno/2021.v10n2e38.17-45>

Oha. (13 de mayo de 2024). *Ohayou(おはよう), HTTP load generator*. Github.

<https://github.com/hatoo/oha?tab=readme-ov-file>

Pérez, S., Quispe, J., Mullicundo, F. y Lamas, D. (2021). *Herramientas y tecnologías para el desarrollo web desde el FrontEnd al BackEnd* [Ponencia]. XXIII Workshop de

Investigadores en Ciencias de la Computación, Chilecito, La Rioja, Argentina.

Procuraduría Ambiental y del Ordenamiento Territorial (s.f.). *Adopta a tu Perro o Gato*.

AdoptaCDMX. Gobierno de la Ciudad de México.

<https://paot.org.mx/adoptacdmx/>

Proyecto Salvavidas Mx AC. (s.f.) *Adopta a un rescatado*.

<https://proyecto-salvavidas.com/>

SonarSource. (s.f.). *SonarQube 10.6 Documentation*.

<https://docs.sonarsource.com/sonarqube/latest/>

Anexo A. Artículo

Implementación de tecnología Serverless Computing en módulo de gestión de adopciones para centros de bienestar animal en México



Autores

Gerardo Alberto Cataño Cañizales

Jessica Herrera Gutiérrez

Betania Estebania Jimenez Jimenez

Luis Diego Fierros Zamarripa

Institución

Universidad Internacional de La Rioja

Fecha

12 de agosto de 2024

Resumen

Este estudio aborda la implementación de tecnología Serverless Computing en un módulo de gestión de adopciones para centros de bienestar animal en México. Utilizando servicios de AWS, se desarrolló una solución que permite gestionar de manera eficiente el registro, seguimiento y promoción de adopciones. Por medio de la cual se busca una mejora significativa en la eficiencia operativa de los centros de adopción y una mayor transparencia en el proceso de adopción.

Palabras Clave

Adopción de mascotas, API REST, AWS, Gestión de bienestar animal, Serverless Computing

Introducción

El abandono de animales en México es un problema crítico, con millones de perros y gatos viviendo en las calles. Este estudio se centra en la implementación de un sistema de gestión de adopciones que facilite la conexión entre animales sin hogar y posibles adoptantes, utilizando tecnología avanzada para asegurar un proceso eficiente y seguro.

Justificación

México enfrenta una situación alarmante de maltrato y abandono animal, siendo el primer lugar en abandono en Latinoamérica. La creación de un sistema eficiente para gestionar adopciones es crucial para reducir la población de animales callejeros y mejorar la salud pública. Según estadísticas, el abandono de animales ha incrementado en un 20% en los últimos cinco años, lo que resalta la urgencia de implementar soluciones tecnológicas para este problema social.

Planteamiento del Problema

El abandono de animales no solo causa sufrimiento a los animales, sino que también representa riesgos para la salud pública. Los animales abandonados pueden ser portadores de enfermedades zoonóticas que afectan a humanos, como la rabia y la leptospirosis. La falta de un sistema centralizado y eficiente para gestionar adopciones contribuye a este problema, dificultando la conexión entre animales necesitados y posibles adoptantes responsables. Además, muchos refugios de animales carecen de recursos y tecnología para gestionar adecuadamente el proceso de adopción, lo que resulta en inefficiencias y errores.

Objetivos del Estudio

El objetivo principal del estudio es implementar un módulo de gestión de adopciones utilizando tecnología Serverless Computing para mejorar la eficiencia operativa de los centros de bienestar animal en México. Los objetivos específicos incluyen:

- Desarrollar una aplicación web accesible y fácil de usar para gestionar adopciones.
- Mejorar la transparencia y el seguimiento del proceso de adopción.
- Reducir los tiempos de espera y procesamiento en el sistema de adopciones.
- Facilitar la integración con otras plataformas y sistemas existentes de gestión de animales.

Materiales y Métodos

Para el desarrollo del módulo de gestión de adopciones, se adoptó una arquitectura Serverless utilizando servicios de AWS, específicamente AWS Lambda, API Gateway y DynamoDB. AWS Lambda se utilizó para ejecutar el código sin necesidad de gestionar servidores, permitiendo una escalabilidad automática y un modelo de pago por uso. API Gateway se encargó de gestionar las solicitudes HTTP, proporcionando una interfaz REST para la comunicación con los usuarios. DynamoDB, una base de datos NoSQL, se utilizó para almacenar la información de los animales y los usuarios.

El desarrollo se llevó a cabo siguiendo una metodología ágil, específicamente el framework Kanban. Esta metodología, derivada del concepto "Just in time", se centra en producir lo necesario en el momento preciso, con la calidad requerida y sin desperdiciar recursos. Kanban facilita la gestión y flujo de recursos/materiales a través de las distintas etapas del proceso sin incurrir en retrasos. Durante el desarrollo, se utilizaron tableros Kanban para visualizar el flujo de trabajo y garantizar la transparencia en el progreso del proyecto. Las tareas se dividieron en pequeñas unidades manejables que

se completaron en ciclos cortos, permitiendo ajustes rápidos y continuos.

El Front-End de la aplicación se desarrolló utilizando el framework Vue.js, mientras que Axios se empleó para manejar las solicitudes HTTP hacia las APIs de AWS. Esta combinación permitió crear una interfaz de usuario responsive y eficiente, accesible desde cualquier dispositivo con conexión a internet. Vue.js, conocido por su flexibilidad y rendimiento, facilitó la creación de componentes reutilizables y una experiencia de usuario fluida.

Para asegurar la calidad del sistema, se implementaron pruebas unitarias e integradas. Las pruebas de integración se centraron en verificar el correcto funcionamiento de los endpoints y la interacción entre los diferentes servicios de AWS, incluyendo la consulta de animales disponibles para adopción, el registro de nuevos animales y la actualización de información existente. Se crearon escenarios de prueba específicos para cada funcionalidad clave del sistema, asegurando que todas las rutas de código estuvieran cubiertas.

El proceso de recolección de datos incluyó la implementación de logs detallados a través de AWS CloudWatch, lo que permitió monitorear el rendimiento del sistema y detectar posibles problemas en tiempo real. Los datos recopilados se utilizaron para ajustar la configuración de los servicios y mejorar la eficiencia general del sistema. Se establecieron alertas y dashboards en CloudWatch para supervisar métricas críticas como el tiempo de respuesta y el número de solicitudes procesadas.

Resultados

El desarrollo del sistema Serverless para la gestión de adopciones ha progresado significativamente, mostrando mejoras prometedoras en la eficiencia operativa de los centros de adopción durante la fase de pruebas internas. A continuación, se detallan los resultados obtenidos hasta la fecha:

Pruebas Funcionales

Las pruebas funcionales del sistema se llevaron a cabo para asegurar que todas las funcionalidades clave operen según lo esperado. Estas pruebas incluyeron la creación, consulta, actualización y eliminación de registros de animales y adoptantes. Se verificó que el sistema maneje correctamente las solicitudes HTTP a través de API Gateway, garantizando respuestas rápidas y precisas.

Pruebas de Integración

Se realizaron pruebas de integración para evaluar la interacción entre los diferentes componentes del sistema. Las pruebas confirmaron que AWS Lambda, API Gateway y DynamoDB trabajan juntos de manera cohesiva. Los endpoints para la consulta de animales disponibles, el registro de nuevos animales y la actualización de información funcionaron sin problemas, demostrando la robustez de la arquitectura Serverless.

Monitoreo y Logística

La implementación de AWS CloudWatch permitió un monitoreo detallado del rendimiento del sistema. Se configuraron dashboards para visualizar métricas críticas como el tiempo de respuesta, el número de solicitudes procesadas y la tasa de errores. Las alertas configuradas en CloudWatch ayudaron a identificar y resolver problemas potenciales de manera proactiva, antes de que afectaran al usuario final.

Análisis de Desempeño

El análisis del desempeño mostró que el sistema es capaz de manejar un alto volumen de solicitudes concurrentes sin degradación significativa en el rendimiento. Durante las pruebas de carga, el sistema mantuvo un tiempo de respuesta inferior a 200 milisegundos para la mayoría de las consultas, demostrando su capacidad para escalar eficientemente durante períodos de alta demanda.

Discusión

Los resultados obtenidos subrayan la importancia de las soluciones digitales en la gestión de adopciones. La adopción de tecnologías Serverless y metodologías ágiles facilitó la adaptación a cambios y la entrega continua de valor. Comparando con

estudios previos, este enfoque mostró mejoras significativas en eficiencia y efectividad.

El uso de Kanban como metodología de desarrollo demostró ser particularmente beneficioso en este contexto. La visualización del trabajo en curso y la capacidad de ajustar rápidamente las prioridades permitieron al equipo de desarrollo responder de manera ágil a las necesidades cambiantes del proyecto.

Sin embargo, la dependencia de la infraestructura de AWS presenta algunas limitaciones. Aunque AWS ofrece una alta disponibilidad y escalabilidad, el costo de los servicios puede aumentar significativamente con el tiempo, especialmente si no se optimizan adecuadamente las funciones y recursos utilizados. Además, la necesidad de formación especializada para gestionar y mantener el sistema puede ser una barrera para algunos centros de bienestar animal con recursos limitados.

Comparación con Estudios Previos

Estudios anteriores han explorado diversas tecnologías para la gestión de adopciones, pero pocos han implementado una solución completamente Serverless. La comparación de nuestros resultados con los de estudios que utilizaron arquitecturas tradicionales muestra una clara ventaja en términos de escalabilidad y eficiencia operativa. Además, la flexibilidad y rapidez de implementación del framework Kanban superaron a metodologías más rígidas utilizadas en proyectos similares.

Conclusiones

La implementación de tecnología Serverless en la gestión de adopciones ha demostrado ser una solución eficaz para mejorar la eficiencia operativa de los centros de bienestar animal en México. El sistema desarrollado facilita el proceso de adopción desde el registro de animales hasta el seguimiento post adopción, asegurando su transparencia y efectividad.

El uso de AWS Lambda y DynamoDB, junto con la metodología Kanban, permitió crear una solución robusta y escalable que puede adaptarse a diferentes contextos y necesidades. La capacidad de integrar

este sistema con plataformas existentes también abre la puerta a futuras mejoras e innovaciones.

Sugerencias para Futuras Investigaciones

Futuras investigaciones podrían explorar la integración de tecnologías de inteligencia artificial para mejorar la predicción de adopciones exitosas y la personalización de recomendaciones para adoptantes potenciales. Además, se podría investigar la aplicación de Blockchain para mejorar la trazabilidad y la transparencia en el proceso de adopción.

El análisis de costos y la optimización de recursos en un entorno Serverless también representan áreas de interés para futuras investigaciones. Evaluar el costo-beneficio a largo plazo de estas tecnologías puede proporcionar insights valiosos para organizaciones con recursos limitados.

Apéndices

Apéndice A: Código fuente de la aplicación.

Apéndice B: Resultados detallados de las pruebas de usuario.

Agradecimientos

Agradecemos al Dr. José Eduardo Ferrer Cruz por su dirección y apoyo durante el desarrollo de este proyecto. Así como a cada miembro del equipo por cumplir con su rol en el desarrollo de este noble proyecto.

Conflictos de Interés

Los autores declaran no tener conflictos de interés en la publicación de este estudio.

Referencias o Bibliografía

Celaya, J. (2022). Informe sobre el maltrato y abandono de animales en México.

Sandoval, P. (2023). Estadísticas de abandono de perros y gatos en México.

Gobierno de la CDMX (s.f.). Problemática de animales en situación de calle.

Anexo 1. Puesta en marcha

Para la puesta en marcha de este proyecto, se deben considerar previamente los siguientes requisitos:

- Conocimientos básicos de desarrollo web, Front-End y framework Vue.js.
- Conocimientos básicos de desarrollo web, Back-End y lenguaje de programación Python.
- Conocimientos básicos de computación en la nube y AWS.
- Tener acceso a una computadora con cualquier sistema operativo (Windows, Linux, MacOS, etc.) y tener instalado cualquier tipo de navegador web actual (Brave, Chrome, Firefox, Edge, Opera, etc.).
- Tener acceso a una cuenta de AWS con permisos de administrador.

Para configurar el ambiente Serverless de este proyecto, se debe de iniciar sesión en la cuenta de AWS. Una vez iniciada la sesión, a través de la consola se elegirá la región más adecuada y que ofrezca la menor latencia; en este caso: Ohio, us-east-2.

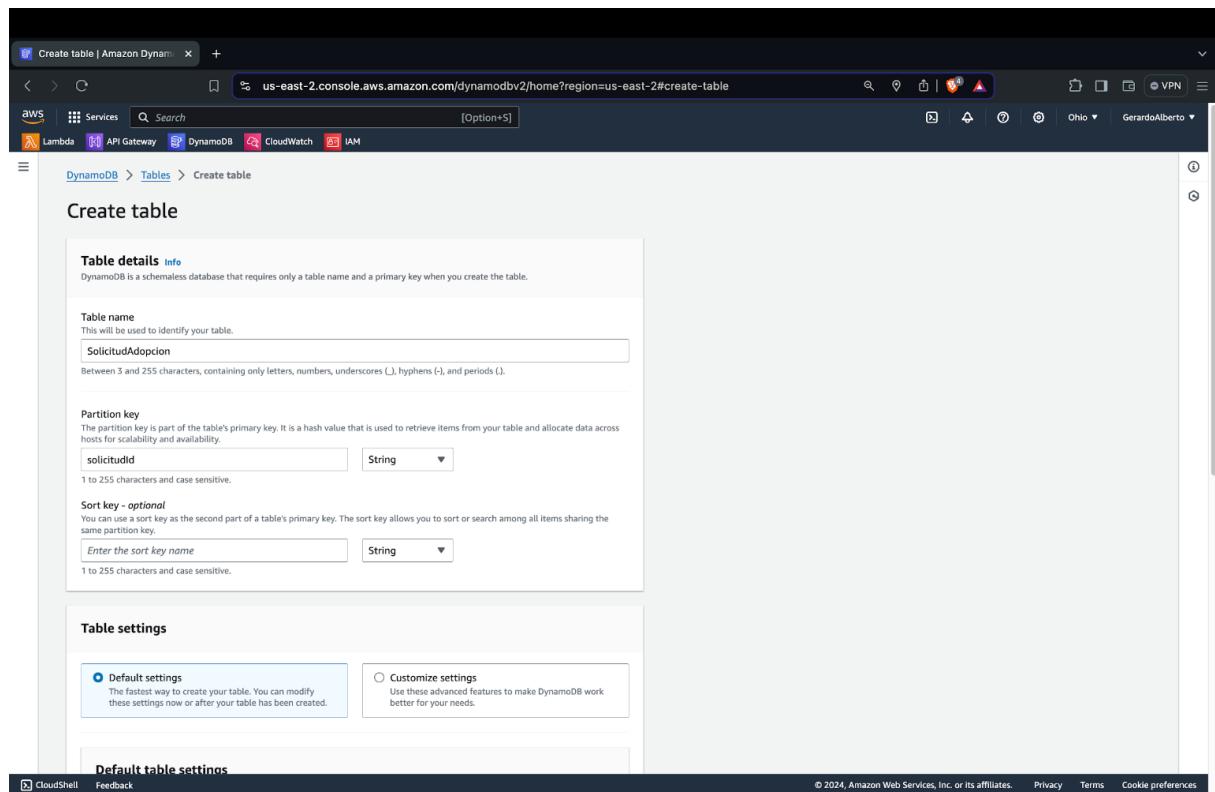
Posteriormente, se tendrán que realizar los siguientes pasos para cada uno de los siguientes servicios, manteniendo la misma región durante todo el proceso de configuración.

DynamoDB

Dentro de la página principal del servicio de DynamoDB, se hará clic en el botón “Create Table”. En la siguiente ventana, se elegirá el nombre de la tabla junto a su respectiva partition key, como se muestra en la **Figura A1-1**. Cabe señalar que en DynamoDB, una partition key funge como el atributo identificador de una tabla.

Figura A1-1

Creación de una tabla en DynamoDB



Posteriormente, se elegirá el resto de los valores por defecto y se hará clic en el botón “Create table”. Este proceso se repetirá para cada una de las tablas que se indican en la **Tabla A1-1**.

Tabla A1-1

Configuración de tablas y partition keys en DynamoDB

Tabla	Partition key
SolicitudAdopcion	solicitudId
MascotaAdopcion	mascotaid
UsuarioAdoptante	usuarioid
CentroAdopcion	centroId
HogarAdoptante	hogarId

Al finalizar, se tendrán las 5 tablas necesarias para este proyecto, como se muestra en la siguiente figura.

Figura A1-2

Tablas necesarias en DynamoDB

The screenshot shows the AWS DynamoDB console interface. The left sidebar has a 'Tables' section selected. The main area displays a table titled 'Tables (5) Info' with columns: Name, Status, Partition key, Sort key, Indexes, Deletion protection, Read capacity mode, Write capacity mode, and Total size. The table lists five tables: CentroAdopcion, HogarAdoptante, MascotaAdopcion, SolicitudAdopcion, and UsuarioAdoptante, all in Active status with Off deletion protection and Provisioned read/write capacity modes.

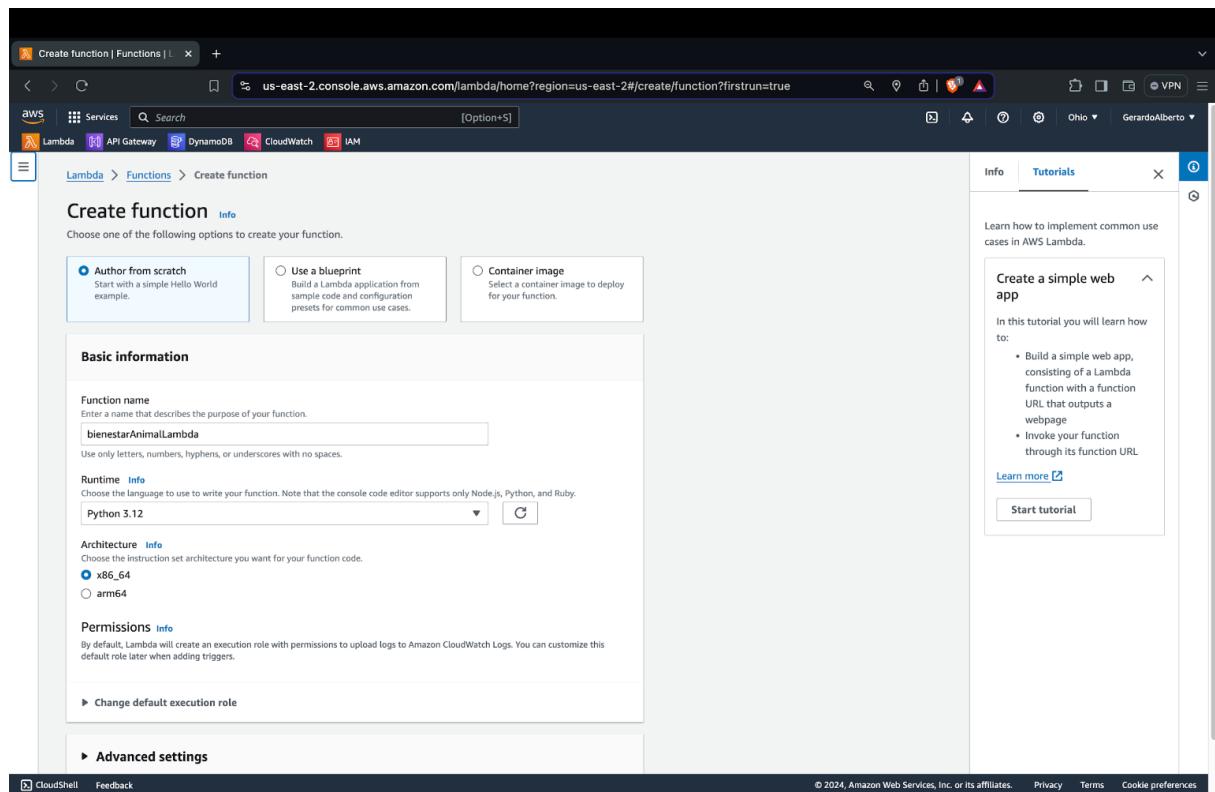
Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode	Total size
CentroAdopcion	Active	centroid (\$)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes
HogarAdoptante	Active	hogarId (\$)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes
MascotaAdopcion	Active	mascotaid (\$)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes
SolicitudAdopcion	Active	solicitudid (\$)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes
UsuarioAdoptante	Active	usuarioid (\$)	-	0	Off	Provisioned (5)	Provisioned (5)	0 bytes

Lambda

Dentro de la página principal del servicio Lambda, se hará clic en el botón “Create a Function”. En la siguiente ventana, se seleccionará la opción para crear una nueva función desde cero, introduciendo “bienestarAnimalLambda” como su nombre, el lenguaje Python 3.12 como Runtime, el resto de las opciones por defecto y finalmente, se hará clic en el botón “Create function”.

Figura A1-3

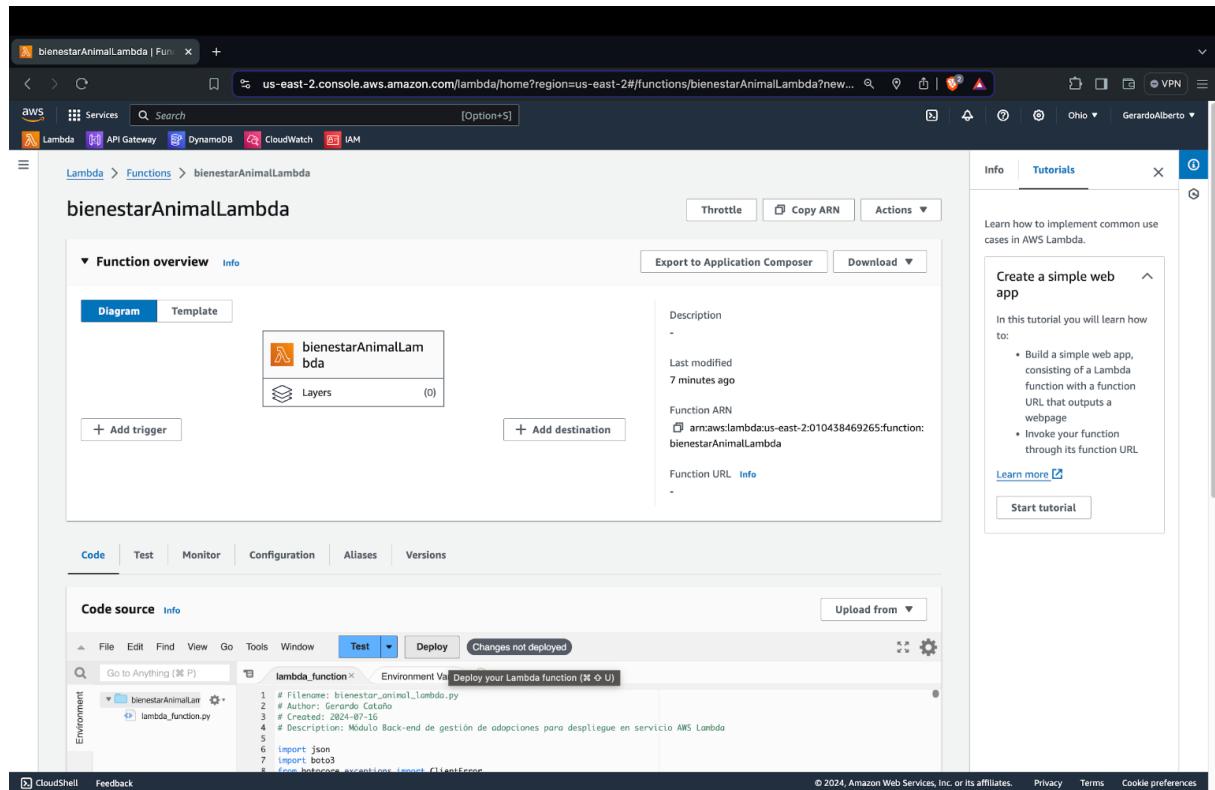
Creación de función Lambda



Después, en la sección “Code” de la función, se copiará el código fuente en lenguaje Python proporcionado en el archivo “**gestion_adopciones_lambda.py**” del **Anexo 4** para este proyecto y posteriormente, se desplegará a través del botón “Deploy”.

Figura A1-4

Despliegue de función Lambda



IAM

Dentro de la sección “Configuration” de la función Lambda, se elegirá la opción “Permissions” y dentro de ella, se hará clic sobre el rol creado por defecto: “bienestarAnimalLambda-role-XXX”. Dicha acción mostrará las características del rol dentro del servicio IAM, como se muestra en la **Figura A1-5**.

Dentro de esta misma página, se elegirá la pestaña “Permissions”, la opción “Add permissions” y posteriormente “Attach policies”.

Figura A1-5

Rol por defecto de la función Lambda dentro de servicio IAM

The screenshot shows the AWS IAM Roles page. The left sidebar is titled 'Identity and Access Management (IAM)' and includes sections for Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), Access reports (Access Analyzer, External access, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies), and Related consoles (IAM Identity Center, AWS Organizations). The main content area shows the details for the 'bienestarAnimalLambda-role-tks0oess' role. The 'Summary' tab is selected, displaying creation date (July 25, 2024, 00:06 (UTC-06:00)), ARN (arn:aws:iam::010438469265:role/service-role/bienestarAnimalLambda-role-tks0oess), and maximum session duration (1 hour). Below the summary, the 'Permissions' tab is active, showing one attached policy: 'AWSLambdaBasicExecutionRole-32468...'. The 'Add permissions' button is highlighted. Other tabs include Trust relationships, Tags, Access Advisor, and Revoke sessions.

La anterior acción abrirá una nueva página, donde se deberá realizar la búsqueda del elemento “AmazonDynamoDBFullAccess”. Posteriormente, se seleccionará y se añadirá al rol mediante el botón “Add permissions”.

Figura A1-6

Añadir permisos de DynamoDB al rol por defecto de la función Lambda dentro de servicio IAM

The screenshot shows the AWS IAM 'Add permissions' dialog. The URL is <https://us-east-1.console.aws.amazon.com/iam/home?region=us-east-1#/roles/details/bienestarAnimalLambda-role-tks0oess>. The search bar at the top contains 'Dynamo'. Below it, a table lists four AWS managed policies under the heading 'Other permissions policies (1/943)'. The policies are:

Policy name	Type	Description
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynamo...
<input type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon D...
<input type="checkbox"/> AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and read access to Dynam...
<input type="checkbox"/> AWSLambdaInvocation-DynamoDB	AWS managed	Provides read access to DynamoDB Str...

At the bottom right of the dialog are 'Cancel' and 'Add permissions' buttons.

La acción anterior otorgará a la función Lambda los accesos de lectura y escritura sobre las tablas de DynamoDB, a través del rol “bienestarAnimalLambda-role-XXX”. Además, este rol por defecto ya contará con los permisos de lectura y escritura necesarios para ver los logs de las operaciones dentro del servicio de CloudWatch.

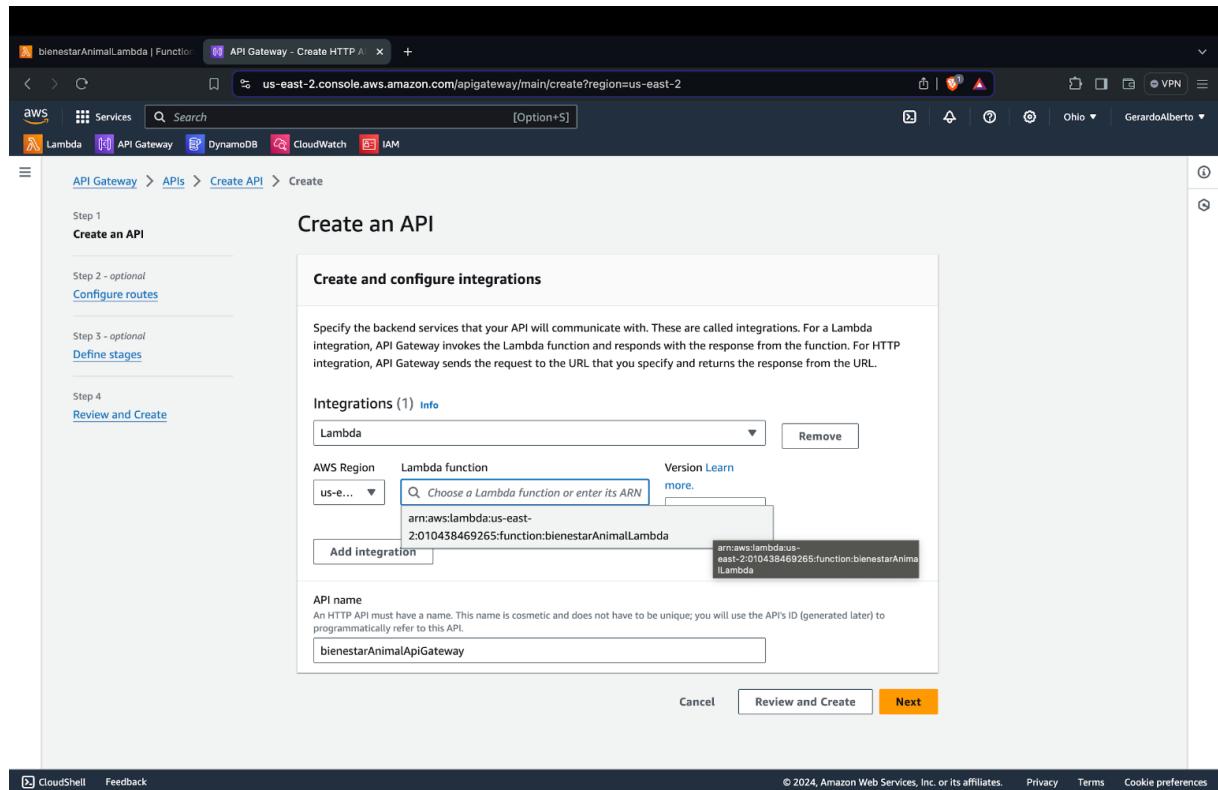
API Gateway

Dentro de la página principal del servicio de API Gateway, se creará un nuevo elemento de tipo “HTTP API”, haciendo clic en su botón “Build” correspondiente.

El siguiente paso será añadir una integración del tipo “Lambda”, seleccionando la función previamente creada: “bienestarAnimalLambda” e introduciendo el nombre de la API: “bienestarAnimalApiGateway”.

Figura A1-7

Integración de función Lambda en API Gateway



Posteriormente, se definirán las rutas de la API, cada una con sus respectivos métodos HTTP soportados y apuntando hacia la misma integración Lambda, definida en el paso anterior. Los valores de configuración de este paso se muestran en la **Tabla A1-2**.

Tabla A1-2

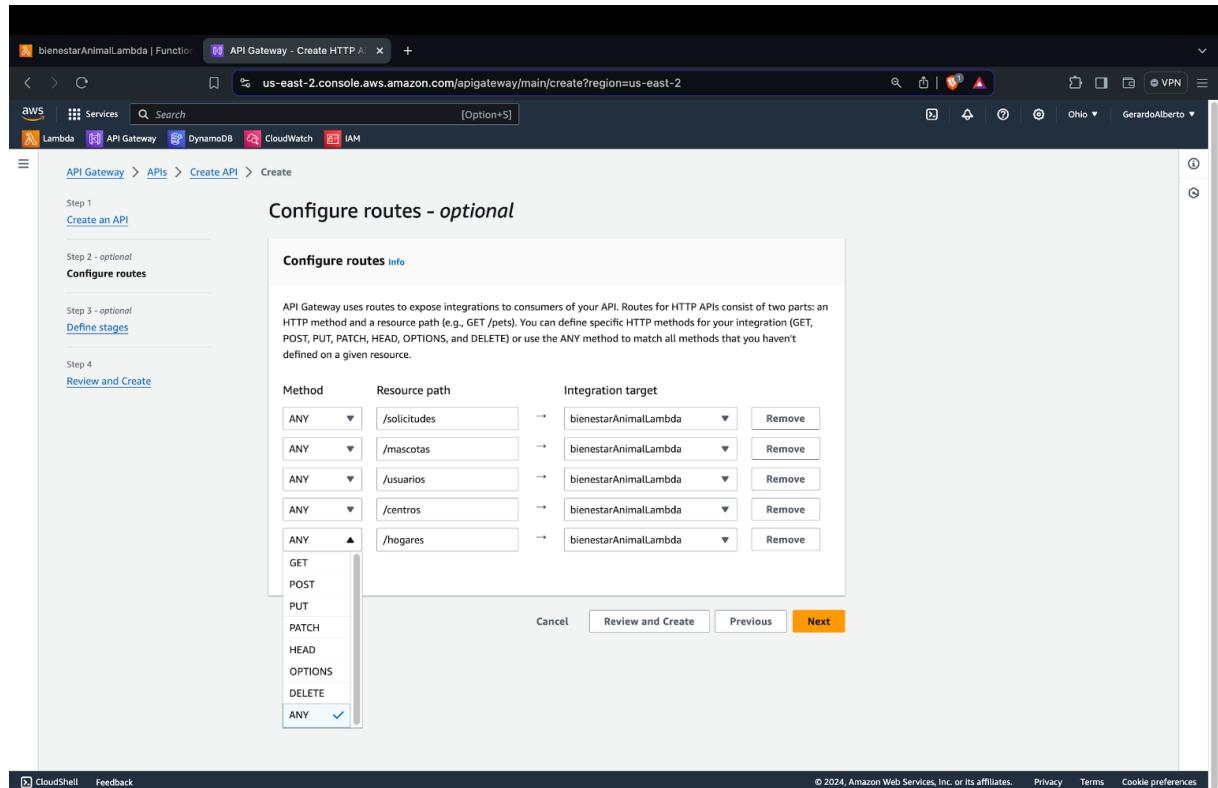
Configuración de métodos, rutas e integraciones en API Gateway

Método HTTP	Ruta	Integración
ANY	/solicitudes	bienestarAnimalLambda
ANY	/mascotas	bienestarAnimalLambda
ANY	/usuarios	bienestarAnimalLambda
ANY	/centros	bienestarAnimalLambda
ANY	/hogares	bienestarAnimalLambda

Por cuestiones de simplicidad, se elegirá la opción “ANY” para el método HTTP, la cual cubre los métodos GET, POST, PUT, DELETE, HEAD, OPTIONS y PATCH; no obstante, la lógica dentro del código de la función Lambda se encargará de aceptar sólo los métodos que fueron implementados, en este caso: GET, POST, PUT y DELETE.

Figura A1-8

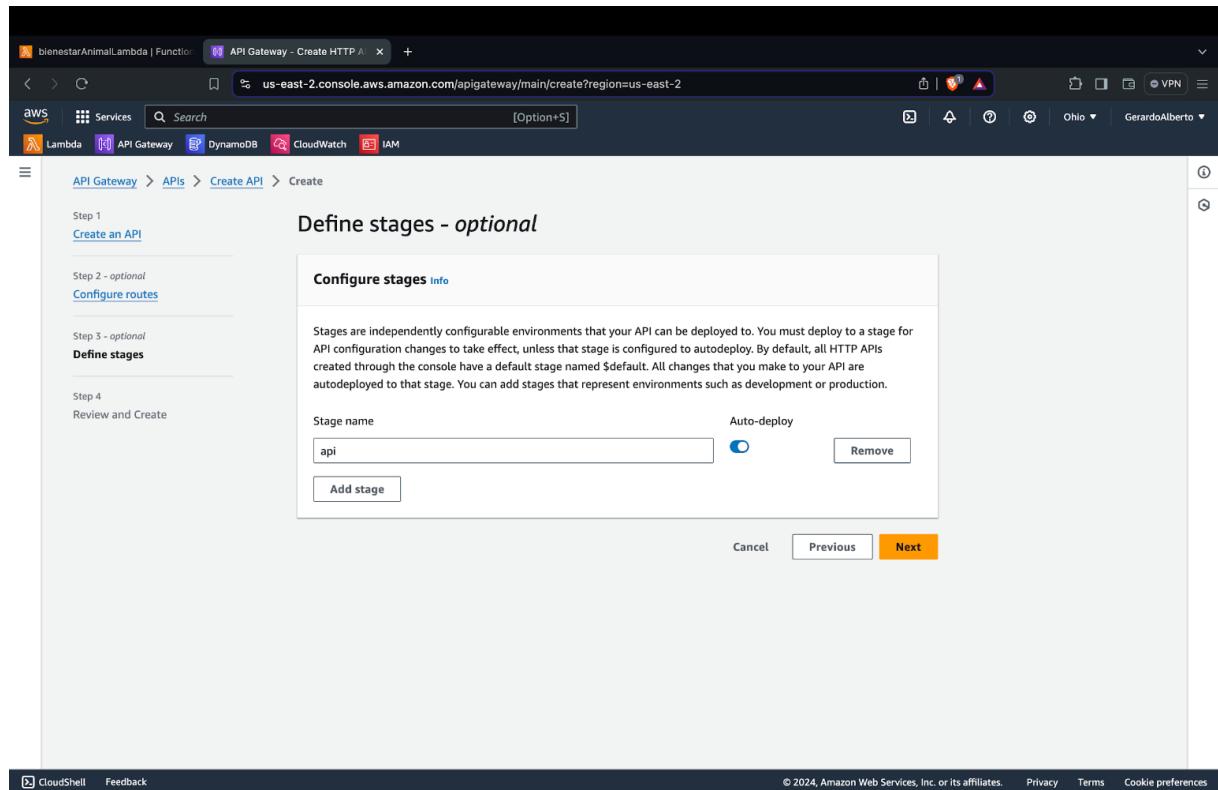
Configuración de rutas en API Gateway



El siguiente paso será definir un nuevo Stage con el nombre “api” y se seleccionará la opción de auto despliegue. Este nombre hará la función de ambiente, añadiendo un prefijo de ruta común para todas las rutas definidas: /api/.

Figura A1-9

Configuración de Stage en API Gateway



Una vez que se revisa la configuración y se hace clic en el botón “Create”, los cambios serán aplicados y el API Gateway será desplegado, por lo que al volver a la función bienestarAnimalLambda dentro del servicio Lambda y tras haber refrescado la página, dentro de la sección “Configuration” y al elegir la opción “Triggers”, se desplegará la ruta absoluta de cada una de las 5 APIs definidas, las cuales estarán listas para ser utilizadas.

Figura A1-10

Configuración de Triggers aplicados a la función Lambda

The screenshot shows the AWS Lambda function configuration page for 'bienestarAnimalLambda'. The left sidebar lists various configuration options: General configuration, Triggers (selected), Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, File systems, and State machines. The main content area displays the 'Triggers' section, which lists five triggers:

- API Gateway: bienestarAnimalApiGateway**
arn:aws:execute-api:us-east-2:010438469265:nhxdmdmrfj/*/usuarios
API endpoint: <https://hhxddmzmfj.execute-api.us-east-2.amazonaws.com/api/usuarios>
- API Gateway: bienestarAnimalApiGateway**
arn:aws:execute-api:us-east-2:010438469265:nhxdmdmrfj/*/mascotas
API endpoint: <https://hhxddmzmfj.execute-api.us-east-2.amazonaws.com/api/mascotas>
- API Gateway: bienestarAnimalApiGateway**
arn:aws:execute-api:us-east-2:010438469265:nhxdmdmrfj/*/hogares
API endpoint: <https://hhxddmzmfj.execute-api.us-east-2.amazonaws.com/api/hogares>
- API Gateway: bienestarAnimalApiGateway**
arn:aws:execute-api:us-east-2:010438469265:nhxdmdmrfj/*/solicitudes
API endpoint: <https://hhxddmzmfj.execute-api.us-east-2.amazonaws.com/api/solicitudes>
- API Gateway: bienestarAnimalApiGateway**
arn:aws:execute-api:us-east-2:010438469265:nhxdmdmrfj/*/centros
API endpoint: <https://hhxddmzmfj.execute-api.us-east-2.amazonaws.com/api/centros>

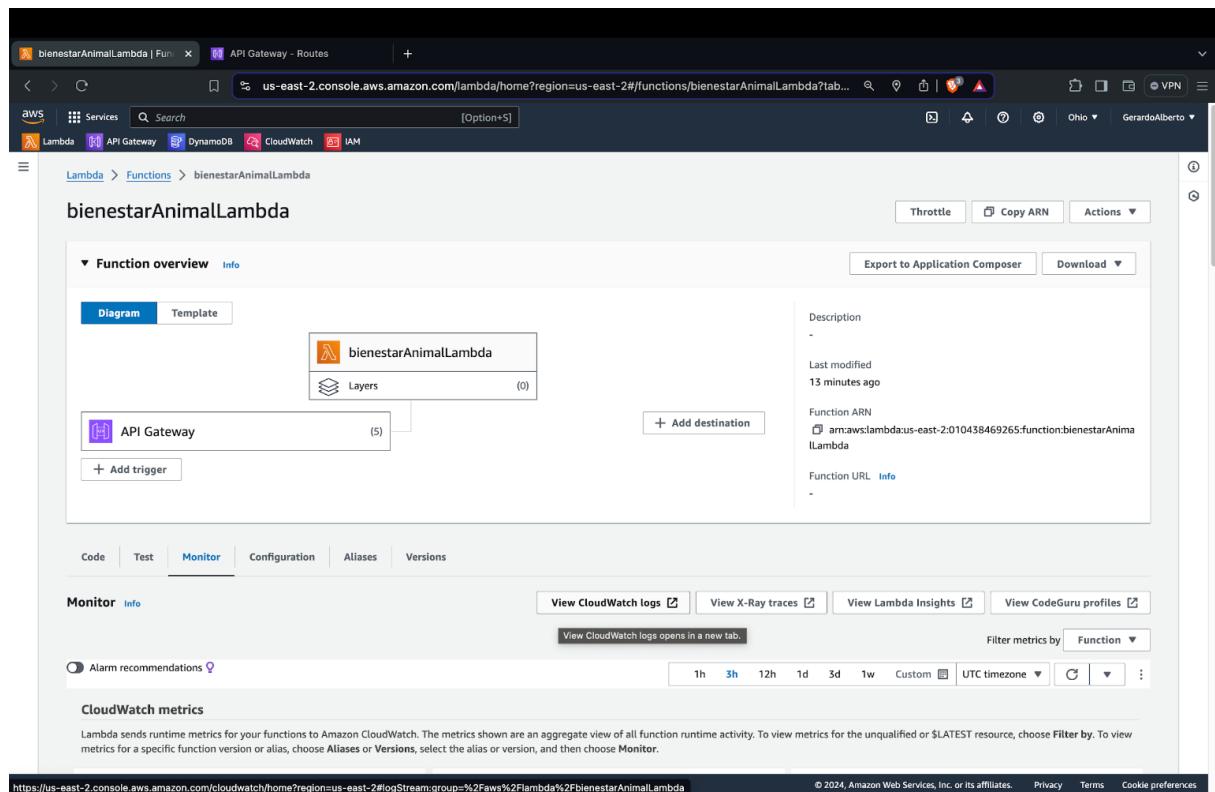
A right-hand sidebar titled 'Create a simple web app' provides a tutorial on how to implement common use cases in AWS Lambda.

CloudWatch

Finalmente, dentro de la sección “Monitor” de la función Lambda, se hará clic en el botón “View CloudWatch logs”, como se aprecia en la siguiente figura.

Figura A1-11

Opción para monitoreo de logs en CloudWatch desde función Lambda



Dicha acción abrirá una nueva pestaña en el navegador para el grupo de logs asociado a la función Lambda dentro del servicio CloudWatch.

Por lo tanto, cada vez que se ejecute una de las APIs que se han definido previamente, en CloudWatch quedará registrada la información general de dicha petición como la fecha y tiempo de inicio y fin de su ejecución (en UTC), su duración y un identificador único, así como información relevante definida a través del código de la función, en este caso, información del evento recibido por la función, la ruta y método utilizados y su respuesta tras procesar la solicitud.

Como ejemplo de ello, se puede observar la ejecución de la API de alta de usuario a través de un cliente Postman en la **Figura A1-12** y, por otro lado, el registro de la misma ejecución reflejada en los logs del servicio CloudWatch, como se puede apreciar en la **Figura A1-13**.

Figura A1-12

Ejecución de API para alta de usuario desde Postman

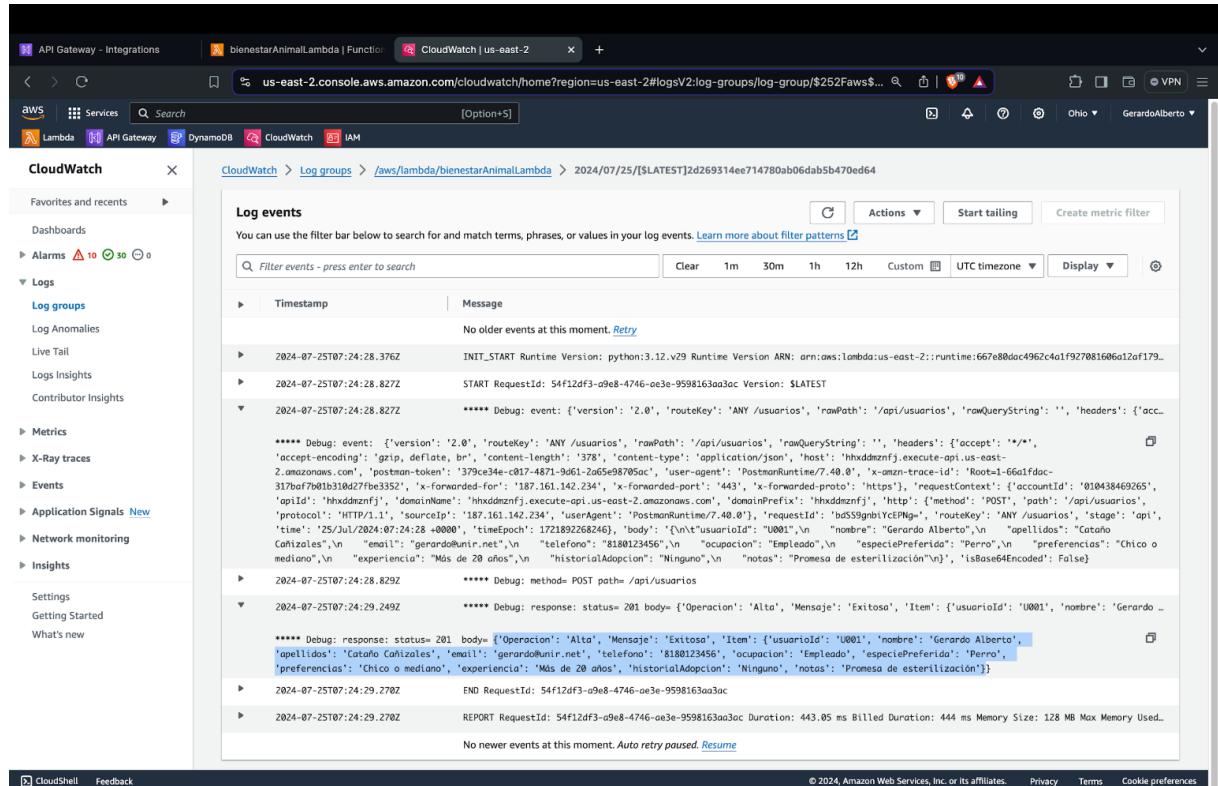
The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists several collections, with 'BienestarAnimal_ServerlessApp' expanded to show categories like 'Solicitudes', 'Mascotas', and 'Usuarios'. Under 'Usuarios', there are three items: 'POST Alta usuario', 'PUT Actualiza usuario', and 'DELETE Baja usuario'. The main workspace shows a POST request to '((LambdaUrl))/api/usuarios/Alta usuario'. The 'Body' tab displays a JSON payload:

```
1 {  
2     "usuarioId": "0001",  
3     "nombre": "Gerardo Alberto",  
4     "apellidos": "Cataño Cahizales",  
5     "mail": "gerardo@unir.net",  
6     "telefono": "+521800123456",  
7     "ocupacion": "Empleado",  
8     "preferencias": "Perro",  
9     "experiencia": "Más de 20 años",  
10    "historialAdopcion": "Ninguno",  
11    "notas": "Promesa de esterilización"  
12 }  
13 }
```

The 'Test Results' tab shows a successful response with status 201 Created, time 1100 ms, and size 581 B. The response body is identical to the request body.

Figura A1-13

Registro de logs para API de alta de usuario en CloudWatch



The screenshot shows the AWS CloudWatch Log Groups interface. The left sidebar navigation includes 'Log groups', 'Metrics', 'X-Ray traces', 'Events', 'Application Signals', 'Network monitoring', and 'Insights'. The main content area displays log events for a specific log group. The first event is an 'INIT_START' log, followed by several 'START RequestId' logs. One log entry is expanded to show detailed debug information about the API request, including headers like 'Content-Type: application/json', 'User-Agent: PostmanRuntime/7.40.0', and 'Host: hxxdmdznfj.execute-api.us-east-2.amazonaws.com'. The request body contains user data such as 'nombre: "Gerardo Alberto"', 'apellidos: "Cataño Cañizales"', 'correo: "gerardo@unir.net"', 'telefono: "8180123456"', 'ocupacion: "Empleado"', 'especiePreferida: "Perro"', 'preferencias: "Chico o mediano"', 'experiencia: "Más de 20 años"', and 'historialAdopcion: "Ninguno"'. The log also shows the response status '201' and the duration of the request.

```

2024-07-25T07:24:28.376Z INIT_START Runtime Version: python:3.12.v29 Runtime Version ARN: arn:aws:lambda:us-east-2:runtime:667e80ddc4962c4a1f927081606e12af179...
2024-07-25T07:24:28.827Z START RequestId: 54f12df3-a0e8-4746-a0e3-9598163aa3ac Version: $LATEST
2024-07-25T07:24:28.827Z ***** Debug: event: {'version': '2.0', 'routeKey': 'ANY /usuarios', 'rawPath': '/api/usuarios', 'rawQueryString': '', 'headers': {'accept': '*/*', 'accept-encoding': 'gzip, deflate, br', 'content-length': '378', 'content-type': 'application/json', 'host': 'hxxdmdznfj.execute-api.us-east-2.amazonaws.com', 'postman-token': '379ce34e-c017-4871-9a61-2a65e98705ec', 'user-agent': 'PostmanRuntime/7.40.0', 'x-amzn-trace-id': 'Root=1-66afdc-317b07901b210d277be352', 'x-forwarded-for': '187.161.142.234', 'x-forwarded-port': '443', 'x-forwarded-proto': 'https'}, 'requestContext': {'accountId': '010438469265', 'ipId': 'hxxdmdznfj', 'domainName': 'hxxdmdznfj.execute-api.us-east-2.amazonaws.com', 'domainPrefix': 'hxxdmdznfj', 'http': {'method': 'POST', 'path': '/api/usuarios', 'protocol': 'HTTP/1.1', 'sourceIp': '187.161.142.234', 'userAgent': 'PostmanRuntime/7.40.0'}, 'requestId': 'bd5599ab1c3d94a', 'routeKey': 'ANY /usuarios', 'stage': 'api', 'time': '25/Jul/2024:07:24:28 +0000', 'timeEpoch': 17189268246}, 'body': '{\n    "usuarioId": "0001",\n    "nombre": "Gerardo Alberto",\n    "apellidos": "Cataño Cañizales",\n    "correo": "gerardo@unir.net",\n    "telefono": "8180123456",\n    "ocupacion": "Empleado",\n    "especiePreferida": "Perro",\n    "preferencias": "Chico o mediano",\n    "experiencia": "Más de 20 años",\n    "historialAdopcion": "Ninguno",\n    "notas": "Promesa de esterilización"\n}', 'isBase64Encoded': false}
2024-07-25T07:24:28.829Z ***** Debug: method: POST path: /api/usuarios
2024-07-25T07:24:29.249Z ***** Debug: response: status: 201 body: {"Operacion": "Alta", "Mensaje": "Exitoosa", "Item": {"usuarioId": "0001", "nombre": "Gerardo Alberto", "apellidos": "Cataño Cañizales", "correo": "gerardo@unir.net", "telefono": "8180123456", "ocupacion": "Empleado", "especiePreferida": "Perro", "preferencias": "Chico o mediano", "experiencia": "Más de 20 años", "historialAdopcion": "Ninguno", "notas": "Promesa de esterilización"}}
2024-07-25T07:24:29.270Z END RequestId: 54f12df3-a0e8-4746-a0e3-9598163aa3ac Duration: 443.05 ms Billed Duration: 444 ms Memory Size: 128 MB Max Memory Used: 128
2024-07-25T07:24:29.270Z REPORT RequestId: 54f12df3-a0e8-4746-a0e3-9598163aa3ac Duration: 443.05 ms Billed Duration: 444 ms Memory Size: 128 MB Max Memory Used: 128

```

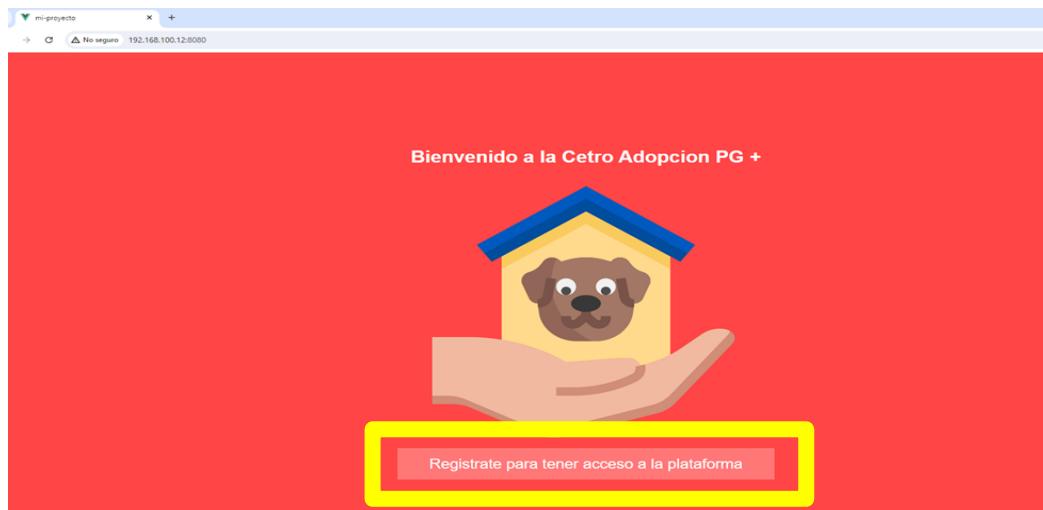
Anexo 2. Manual básico de usuario

A continuación se indica la forma inicial en donde el usuario podrá realizar la solicitud de adopción:

1. Ingresar a la plataforma Centro de Adopción PG +, Figura A2-1.

Figura A2-1

Sitio web - Centro de adopción PG+, página de bienvenida

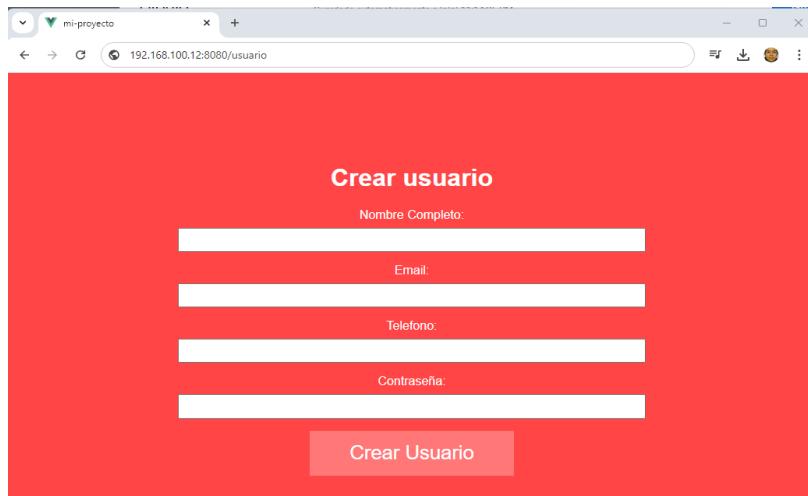


2. Dar clic en el botón Registrarse para tener acceso a la plataforma y crear un usuario,

Figura A2-2.

Figura A2-2

Formulario para la creación de un usuario



Crear usuario

Nombre Completo:

Email:

Teléfono:

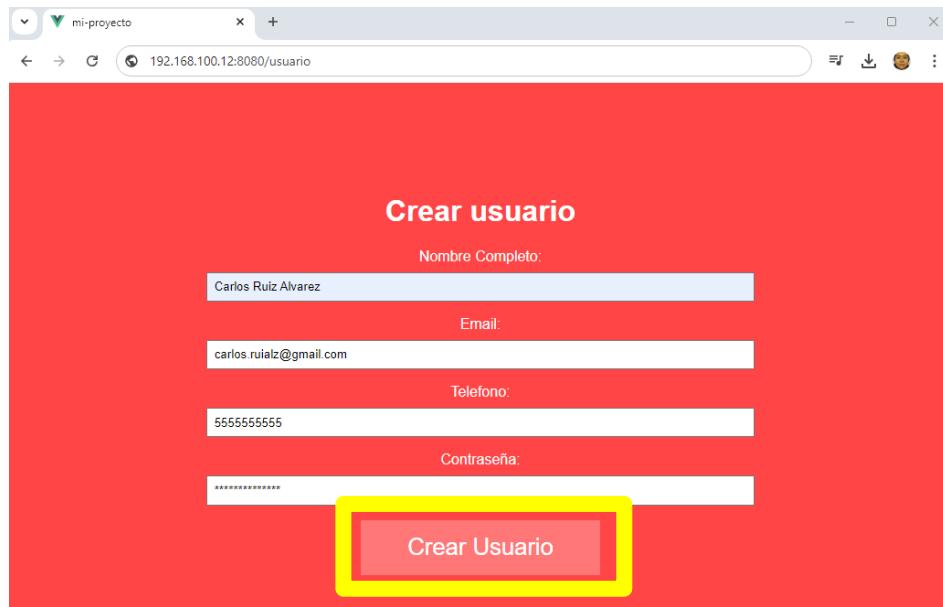
Contraseña:

Crear Usuario

3. Llenar los campos solicitados y dar clic en crear, Figura A2-3.

Figura A2-3

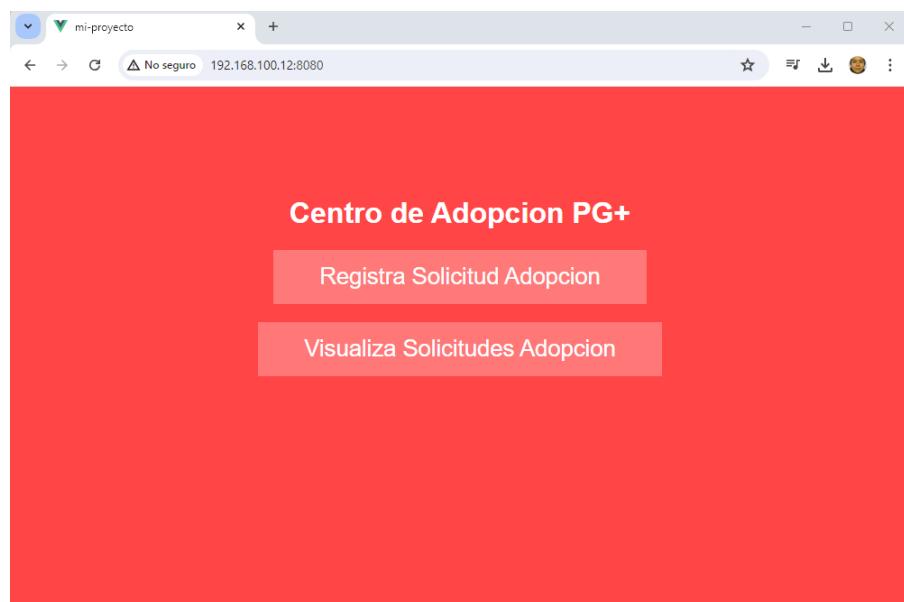
Ejemplo de creación de nuevo usuario en la plataforma



4. Una vez creado, el usuario podrá visualizar el portal de adopción PG+, Figura A2-4.

Figura A2-4

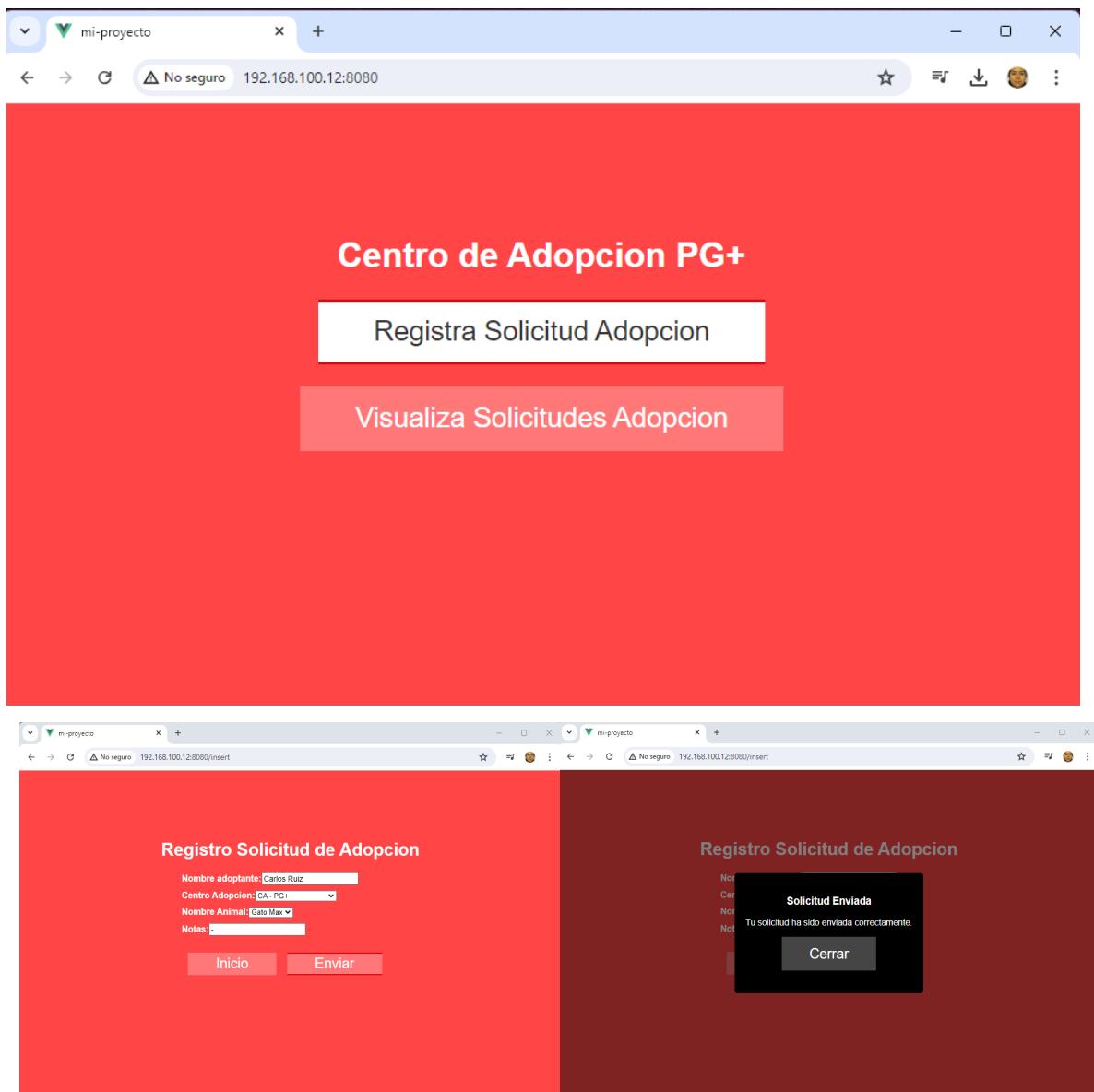
Ventana que se visualiza una vez creado su perfil en el portal



5. Crear una solicitud dando clic en Registrar Solicitud de adopción y realizar el llenado de los campos requeridos, Figura A2-5.

Figura A2-5

Realización de una solicitud de adopción, Captura y envío de solicitud



6. Una vez creada, se podrá dar seguimiento a la solicitud en Visualiza Solicitudes Adopción, Figura A2-6.

Figura A2-6

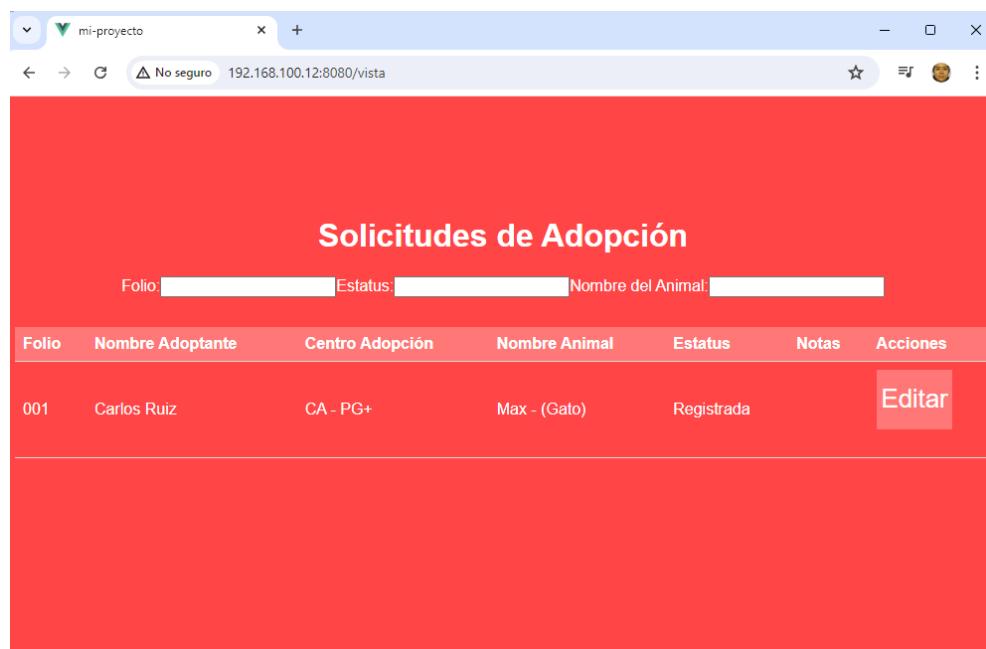
Ventana con la opción de visualizar solicitudes creadas



7. Acorde al perfil del usuario, es la información que se podrá visualizar, Figura A2-7.

Figura A2-7

Ejemplo de visualización de captura de solicitud de adopción



Folio	Nombre Adoptante	Centro Adopción	Nombre Animal	Estatus	Notas	Acciones
001	Carlos Ruiz	CA - PG+	Max - (Gato)	Registrada		<button>Editar</button>

Anexo 3. Guía básica de mantenimiento

Una de las principales ventajas de la solución propuesta en este documento, es precisamente la facilidad y conveniencia de su mantenimiento, ya que al tratarse de una arquitectura formada por servicios Serverless, el proveedor de nube AWS se encargará de la gestión y actividades de mantenimiento de la infraestructura que los soporta, como la instalación rutinaria de parches de seguridad en el Software y la alta disponibilidad, mecanismos de auto-recuperación y escalabilidad automática de los servicios, para adaptarse a cualquier pico de utilización por alta demanda.

Además, el coste asociado a estos servicios es únicamente por utilización en el caso de los servicios Lambda y API Gateway, y por cantidad de almacenamiento en el caso de DynamoDB.

Estos costos se generan mensualmente, no obstante, otra de las principales ventajas que ofrece esta solución es que AWS cuenta con una capa de nivel gratuito (Free Tier), donde se incluyen 25 GB de almacenamiento en DynamoDB y 1 millón de solicitudes en Lambda gratuitas por mes de por vida, y 1 millón de solicitudes en API Gateway gratuitas por mes durante los primeros 12 meses de la creación de una nueva cuenta. Al pasar estos límites, simplemente se pagarán los excedentes de utilización.

Figura A3-1

Nivel gratuito en servicios de AWS

The screenshot shows the AWS Free Tier landing page. On the left, there's a sidebar with filtering options for 'Tipo de nivel' (Selected: Destacado) and 'Categorías de productos'. The main content area is divided into four sections: ALMACENAMIENTO (Amazon S3 5 GB), BASE DE DATOS (Amazon DynamoDB 25 GB), COMPUTACIÓN (AWS Lambda 1 millón solicitudes), SOLUCIONES MÓVILES (Amazon SNS 1 millón de publicaciones), SOLUCIONES MÓVILES (Amazon API Gateway 1 millón de llamadas), and INTEGRACIÓN DE APLICACIONES (Amazon SQS 1 millón de solicitudes). Each service card includes a brief description and usage limits.

Debido a las anteriores razones, realmente no se necesitaría un soporte especializado para la solución descrita en este proyecto, no obstante, AWS cuenta con una amplia y detallada documentación para cada servicio disponible en su sitio, y también ofrece 4 diferentes planes de soporte, según el tamaño y las necesidades específicas de cada cliente, los cuales son: Developer, Business, Enterprise On-Ramp y Enterprise.

Figura A3-2

Planes de soporte en AWS

The screenshot shows a web browser displaying the AWS Premium Support page at aws.amazon.com/es/premiumsupport/?nc2=h_m_bc. The page title is "Planes de asistencia".

Soporte para desarrolladores
El soporte para desarrolladores está diseñado para estudiantes, usuarios ocasionales y experimentadores las que prueben o comiencen el desarrollo inicial en AWS y que requieran soporte de consultas.

[Más información »](#)

Business Support
AWS Business Support es el nivel de AWS Support mínimo recomendado si tiene cargas de trabajo de producción en AWS y necesita acceso las 24 horas del día, los 7 días de la semana, a soporte técnico en el contexto de sus casos de uso específicos.

[Más información »](#)

Enterprise On-Ramp
Recomendamos Enterprise On-Ramp si tiene cargas de trabajo críticas para la empresa o producción en AWS y quiere tener acceso a soporte técnico de ingenieros las 24 horas del día, los siete días de la semana, acceso a la API de Health, orientación arquitectónica consultiva y un grupo de gerentes técnicos de cuentas (TAM) para coordinar el acceso a los expertos en la materia de AWS.

[Más información »](#)

Enterprise Support
Se recomienda Enterprise Support si tiene cargas de trabajo esenciales o empresariales en AWS. Con Enterprise Support, obtiene soporte técnico las 24 horas del día, los 7 días de la semana, de ingenieros, herramientas y tecnología de alta calidad para administrar automáticamente el estado de su entorno. También recibirá la ayuda de un administrador técnico de cuentas (TAM) que le proporcionará orientación de consulta sobre la arquitectura y las operaciones en el contexto de sus aplicaciones junto con casos de uso para ayudarlo a aprovechar al máximo AWS.

[Más información »](#)

Anexo 4. Código Python para función Lambda

En este apartado se adjunta el código fuente en lenguaje de programación Python, el cual contiene la lógica de negocio del módulo de gestión de adopciones para su implementación como función dentro del servicio de AWS Lambda.

```
# Filename: gestion_adopciones_lambda.py
# Author: Gerardo Cataño
# Created: 2024-07-16
# Description: Módulo Back-end de gestión de adopciones para despliegue en servicio
# AWS Lambda

import json
import boto3
from botocore.exceptions import ClientError
from decimal import Decimal
from boto3.dynamodb.conditions import Key

#Inicializa rutas
solicitudes_path = '/api/solicitudes'
mascotas_path = '/api/mascotas'
usuarios_path = '/api/usuarios'
centros_path = '/api/centros'
hogares_path = '/api/hogares'

#Inicializa cliente DynamoDB y tablas
dynamodb = boto3.resource('dynamodb', region_name='us-east-2')      #interfaz de
recursos: actualizar region
solicitudes_table = dynamodb.Table('SolicitudAdopcion')
mascotas_table = dynamodb.Table('MascotaAdopcion')
usuarios_table = dynamodb.Table('UsuarioAdoptante')
centros_table = dynamodb.Table('CentroAdopcion')
hogares_table = dynamodb.Table('HogarAdoptante')
```

```
def lambda_handler(event, context):
    print('***** Debug: event: ', event)
    print('***** Debug: method=', event['requestContext']['http']['method'], ' path=',
event['requestContext']['http']['path'])
    response = None

try:
    http_method = event['requestContext']['http']['method']
    path = event['requestContext']['http']['path']

    #Lógica basada en rutas y posteriormente en métodos
    if path == solicitudes_path:

        if http_method == 'GET':
            #Logica para saber si solicitudId existe en queryStringParameters de
            evento
            isSolicitudPresent = False
            for i in event:
                if 'queryStringParameters' in i:
                    solicitud_id = event['queryStringParameters']['solicitudId']
                    response = consulta_item(solicitudes_table, 'solicitudId',
solicitud_id)
                    isSolicitudPresent = True
                    break
            #not in for i:
            if isSolicitudPresent != True:
                response = consulta_items(solicitudes_table)

        elif http_method == 'POST':
            response = alta_item(solicitudes_table, json.loads(event['body']))

        elif http_method == 'PUT':
            body = json.loads(event['body'])
            solicitud_id = event['queryStringParameters']['solicitudId']
            response = actualiza_item(solicitudes_table, 'solicitudId',
solicitud_id, body['atributo'], body['valor'])

        elif http_method == 'DELETE':
            solicitud_id = event['queryStringParameters']['solicitudId']
            response = baja_item(solicitudes_table, 'solicitudId', solicitud_id)

    else:
        return build_response(405, '405: Metodo no permitido')


```

```
elif path == mascotas_path:

    if http_method == 'GET':
        #Logica para saber si mascotaId existe en queryStringParameters de
        evento
        isMascotaIdPresent = False
        for i in event:
            if 'queryStringParameters' in i:
                mascota_id = event['queryStringParameters']['mascotaId']
                response = consulta_item(mascotas_table, 'mascotaId',
mascota_id)
                isMascotaIdPresent = True
                break
        #not in for i:
        if isMascotaIdPresent != True:
            response = consulta_items(mascotas_table)

    elif http_method == 'POST':
        response = alta_item(mascotas_table, json.loads(event['body']))

    elif http_method == 'PUT':
        body = json.loads(event['body'])
        mascota_id = event['queryStringParameters']['mascotaId']
        response = actualiza_item(mascotas_table, 'mascotaId', mascota_id,
body['atributo'], body['valor'])

    elif http_method == 'DELETE':
        mascota_id = event['queryStringParameters']['mascotaId']
        response = baja_item(mascotas_table, 'mascotaId', mascota_id)

    else:
        return build_response(405, '405: Metodo no permitido')

elif path == usuarios_path:

    if http_method == 'GET':
        #Logica para saber si usuarioId existe en queryStringParameters de
        evento
        isUsuarioIdPresent = False
        for i in event:
            if 'queryStringParameters' in i:
                usuario_id = event['queryStringParameters']['usuarioId']
```

```
        response = consulta_item(usuarios_table, 'usuarioId',
usuario_id)
        isUsuarioIdPresent = True
        break
    #not in for i:
    if isUsuarioIdPresent != True:
        response = consulta_items(usuarios_table)

    elif http_method == 'POST':
        response = alta_item(usuarios_table, json.loads(event['body']))

    elif http_method == 'PUT':
        body = json.loads(event['body'])
        usuario_id = event['queryStringParameters']['usuarioId']
        response = actualiza_item(usuarios_table, 'usuarioId', usuario_id,
body['atributo'], body['valor'])

    elif http_method == 'DELETE':
        usuario_id = event['queryStringParameters']['usuarioId']
        response = baja_item(usuarios_table, 'usuarioId', usuario_id)

    else:
        return build_response(405, '405: Metodo no permitido')

    elif path == centros_path:

        if http_method == 'GET':
            #Logica para saber si centroId existe en queryStringParameters de
evento
            isCentroIdPresent = False
            for i in event:
                if 'queryStringParameters' in i:
                    centro_id = event['queryStringParameters']['centroId']
                    response = consulta_item(centros_table, 'centroId',
centro_id)
                    isCentroIdPresent = True
                    break
            #not in for i:
            if isCentroIdPresent != True:
                response = consulta_items(centros_table)

        elif http_method == 'POST':
            response = alta_item(centros_table, json.loads(event['body']))
```

```
        elif http_method == 'PUT':
            body = json.loads(event['body'])
            centro_id = event['queryStringParameters']['centroId']
            response = actualiza_item(centros_table, 'centroId', centro_id,
body['atributo'], body['valor'])

        elif http_method == 'DELETE':
            centro_id = event['queryStringParameters']['centroId']
            response = baja_item(centros_table, 'centroId', centro_id)

    else:
        return build_response(405, '405: Metodo no permitido')

elif path == hogares_path:

    if http_method == 'GET':
        #Logica para saber si hogarId existe en queryStringParameters de evento
        isHogarIdPresent = False
        for i in event:
            if 'queryStringParameters' in i:
                hogar_id = event['queryStringParameters']['hogarId']
                response = consulta_item(hogares_table, 'hogarId', hogar_id)
                isHogarIdPresent = True
                break
        #not in for i:
        if isHogarIdPresent != True:
            response = consulta_items(hogares_table)

    elif http_method == 'POST':
        response = alta_item(hogares_table, json.loads(event['body']))

    elif http_method == 'PUT':
        body = json.loads(event['body'])
        hogar_id = event['queryStringParameters']['hogarId']
        response = actualiza_item(hogares_table, 'hogarId', hogar_id,
body['atributo'], body['valor'])

    elif http_method == 'DELETE':
        hogar_id = event['queryStringParameters']['hogarId']
        response = baja_item(hogares_table, 'hogarId', hogar_id)

else:
```

```
        return build_response(405, '405: Metodo no permitido')

    else:
        response = build_response(404, '404: Recurso no encontrado')      #####not
reachable

except Exception as e:
    print('Error:', e)
    response = build_response(400, 'Error en peticion HTTP')

return response


def consulta_item(dynamodb_table, item_key, item_id):
    try:
        response = dynamodb_table.get_item(Key={item_key: item_id})
        #Logica para saber si Item existe en response
        isItemPresent = False
        for i in response:
            if 'Item' in i:
                return build_response(200, response.get('Item'))
                isItemPresent = True
                break
        #not in for i:
        if isItemPresent != True:
            return build_response(404, '404: Item no encontrado')
    except ClientError as e:
        print('Error:', e)
        return build_response(400, e.response['Error']['Message'])




def consulta_items(dynamodb_table):
    try:
        scan_params = {
            'TableName': dynamodb_table.name
        }
        return build_response(200, scan_dynamo_records(dynamodb_table, scan_params,
[]))
    except ClientError as e:
        print('Error:', e)
        return build_response(400, e.response['Error']['Message'])




def scan_dynamo_records(dynamodb_table, scan_params, item_array):
    response = dynamodb_table.scan(**scan_params)
```

```
item_array.extend(response.get('Items', []))

if 'LastEvaluatedKey' in response:
    scan_params['ExclusiveStartKey'] = response['LastEvaluatedKey']
    return scan_dynamo_records(scan_params, item_array)
else:
    return {'items': item_array}

def alta_item(dynamodb_table, request_body):
    try:
        dynamodb_table.put_item(Item=request_body)
        body = {
            'Operacion': 'Alta',
            'Mensaje': 'Exitosa',
            'Item': request_body
        }
        return build_response(201, body)
    except ClientError as e:
        print('Error:', e)
        return build_response(400, e.response['Error']['Message'])

def actualiza_item(dynamodb_table, item_key, item_id, update_key, update_value):
    try:
        response = dynamodb_table.update_item(
            Key={item_key: item_id},
            UpdateExpression=f'SET {update_key} = :value',
            ExpressionAttributeValues={':value': update_value},
            ReturnValues='UPDATED_NEW'
        )
        body = {
            'Operacion': 'Actualizacion',
            'Mensaje': 'Exitosa',
            'UpdatedAttributes': response
        }
        return build_response(200, body)
    except ClientError as e:
        print('Error:', e)
        return build_response(400, e.response['Error']['Message'])

def baja_item(dynamodb_table, item_key, item_id):
    try:
        response = dynamodb_table.delete_item(
```

```
        Key={item_key: item_id},
        ReturnValues='ALL_OLD'
    )
body = {
    'Operacion': 'Baja',
    'Mensaje': 'Exitosa',
    'Item': response
}
return build_response(200, body)
except ClientError as e:
    print('Error:', e)
    return build_response(400, e.response['Error']['Message'])

class DecimalEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Decimal):
            # Check if it's an int or a float
            if obj % 1 == 0:
                return int(obj)
            else:
                return float(obj)
        # Let the base class default method raise the TypeError
        return super(DecimalEncoder, self).default(obj)

def build_response(status_code, body):
    print('***** Debug: response: status=', status_code, ' body=', body)
    return {
        'statusCode': status_code,
        'headers': {
            'Content-Type': 'application/json'
        },
        'body': json.dumps(body, cls=DecimalEncoder)
    }
```