ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE UN SISTEMA WEB Y UNA APLICACIÓN MÓVIL PARA GESTIONAR LA ADOPCIÓN DE MASCOTAS EN LA CIUDAD DE QUITO

DESARROLLO DEL BACKEND

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN DESARROLLO DE SOFTWARE

CARLOS ANDRES QUEL REDROBÁN

DIRECTOR: DR. RICHARD PAUL RIVERA GUEVARA

DMQ, septiembre 2022

CERTIFICACIONES

Yo, Carlos Andres Quel Redrobán declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

CARLOS ANDRES QUEL REDROBÁN

carlos.quel@epn.edu.ec

and.quel5@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por Carlos Andres Quel Redrobán, bajo mi supervisión.

DR. RICHARD PAUL RIVERA GUEVARA

DIRECTOR

richard.rivera01@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Carlos Andres Quel Redrobán

DEDICATORIA

El presente proyecto va dedicado a toda la comunidad politécnica que ha estado en constante aprendizaje, con muy buenos profesores de excelente enseñanza. También, va dedicado a todas las personas en general que buscan desarrollos personales y empresariales. Ha sido un proyecto de mucho interés que puede servir para la integración en cualquier proyecto relacionado a las mascotas, y por obvias razones se incluye la dedicación a los animales que esperan un mundo mejor.

QUEL REDROBÁN CARLOS ANDRES

AGRADECIMIENTO

Agradezco a Dios por haberme dado una hermosa familia y por este enorme logro que ha costado mucho esfuerzo, sin duda un esfuerzo que consta de mucho apoyo. Entonces, doy gracias a toda mi familia que ha estado en constante lucha a mi lado para culminar mi etapa de estudios con bastante confianza y fuertes ánimos para seguir. Además, se agradece a la Universidad EPN, a los profesores y compañeros de estudios por haberme permitido integrar mis conocimientos.

QUEL REDROBÁN CARLOS ANDRES

ÍNDICE DE CONTENIDO

RESUMEN

Las plataformas digitales han ido incrementando día tras día para solucionar problemas sociales. Un ejemplo es el desempleo en Ecuador que obliga a las personas en busca del comercio y servicio en línea para solventar las necesidades de los demás. El objetivo de este proyecto es desarrollar una aplicación web que pueda ser usada por usuarios finales interesados en los animales. El sistema tiene como conjunto principal al *backend* que es el responsable de los datos y en el cual se va a especializar este servicio interno.

¿Por qué tantos animales en las calles? La situación económica en el país es grave lo cual afecta en cada uno de los hogares, por lo que lleva a personas con animales a abandonarlos. También, problemas sociales provocan descuidos hacia las mascotas sin un chequeo veterinario. Entonces, la solución a esto es aplicar herramientas que faciliten al *frontend* parte del sistema; tanto en la presentación como en el manejo de los datos ser obtenidos de forma correcta.

Herramienta fundamental para el proyecto ha sido la utilización de Laravel con todas las implementaciones. Provee un magnífico uso compartido con el gestor de base de datos PostgreSQL en la información almacenada. Además, el proyecto presenta el uso de la metodología *Scrum* por la fácil manipulación en las actividades con revisiones preventiva; adaptable y susceptible a cambios. A continuación, en los capítulos la presentación de la *API REST* desarrollada con resultados logrados y el producto expuesto al cliente en un despliegue ya en producción.

PALABRAS CLAVE: *backend*, Laravel, PostgreSQL, *Scrum*, *API REST*, mascotas.

ABSTRACT

Digital platforms have been increasing day after day for solving social problems. One example is unemployment in Ecuador that forces to people looking for online trading and service to meet the needs of others. The goal of this project is to develop a web application that can be used by interested final users in animals. The system has as main set to backend that's responsible for the data and in which is going to specialize this internal service.

¿Why are there many animals at streets? The economic situation at country is serious, which affects each of the households; therefore, it takes people with animals to abandon them. Too, social problems cause carelessness towards pets without a veterinary checkup. So, the solution to this is to apply tools that help frontend system part; both in the presentation and in the handling of data to be gotten correctly.

A fundamental tool for the project has been the use of Laravel with all of implementations. It provides great sharing with the database management PostgreSQL in the stored information. Also, for the project presents the use of the Scrum methodology for easy handling in the activities with preventive reviews, adaptable and sensitive to changes. Next, in chapters the presentation of the developed API REST with achieved results and the product exposed to the customer in a deployment already in production.

KEYWORDS: backend, Laravel, PostgreSQL, Scrum, API REST, pets.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

Actualmente, ha sido de mucha importancia las plataformas digitales; debido a que, la gran mayoría de las personas pasan más tiempo conectadas a internet con más de 30 millones de dispositivos [1]. Por tal motivo, exigen una demanda de migrar a la era tecnológica donde la vida cotidiana de la gente cambia radicalmente [2]. Añadiendo, la tecnología trata en lo más posible día tras día de actualizarse y facilitar el desarrollo de los sistemas en su totalidad.

Quito cuenta con un 45% de mascotas que se encuentran en las calles abandonadas o perdidas en la ciudad sin un destino fijo y estable [3]. A tal alcance, la ciudad necesita una mayor regulación en el cuidado de los animales con mayor responsabilidad e información sobre mascotas rescatadas. El objetivo es llegar a disminuir esas mascotas desorientadas, y brindarles un nuevo hogar o recuperarles su antiguo hogar con su estabilidad emocional.

En este sentido, tener una idea concisa de precisar la búsqueda de una mascota perdida; o adquirir una mascota nueva para una compañía única. Se lo tiene que hacer directamente desde los medios digitales permitiendo de esta forma a las personas un mayor interés en el internet [4], que ofrecen grandes noticias e información en general. Entonces, ya existen fundaciones con su propio sistema para recaudar fondos, publicar y donar varios animales; que son muy bien cuidados. Sin embargo, no brindan esa experiencia de uno a uno como usuarios.

La experiencia de uno a uno es la interacción entre personas para comunicarse sobre un tema, más comúnmente llamado usuario que se comunica con otro usuario a través de ordenadores en cualquier parte específicamente de la ciudad de Quito [5]. Obviamente, debe mostrar nuevas características de interacción permitiendo abordar las necesidades de cada usuario en la parte personal. Ahora, el interés va a hacer tanto por la tecnología como por la mascota y por el usuario que tan respetuoso sea frente a los demás usuarios.

Enfocados en lo mencionado anteriormente, gracias a la evolución de la tecnología que permite utilizar herramientas de desarrollo muy interesantes en la actualidad para la respectiva implementación del *backend* probada desde una *API REST* que ayuda con la comunicación entre el sistema y la base de datos. Dentro de las tecnologías válidas para presentar y enviar información se requiere también cuidar la integridad, consistencia y seguridad de los datos [6].

1.1 Objetivo general

Desarrollar un sistema web y una aplicación móvil para gestionar la adopción de mascotas en la ciudad de Quito.

1.2 Objetivos específicos

1. OE1 Determinar los requerimientos globales.
2. OE2 Diseñar el modelo de la base de datos y la arquitectura.
3. OE3 Desarrollar el *API REST*.
4. OE4 Proporcionar pruebas de aceptación del *API REST*.
5. OE5 Desplegar el *API REST* a modo de producción.

1.3 Alcance

El presente proyecto propone el despliegue de una *API REST* por medio de herramientas funcionales, que en la mayor parte del desarrollo se busque resultados precisos y claros. Entonces, el sistema proveerá tres perfiles; el primero tiene como finalidad administrar en su totalidad tanto los usuarios como las publicaciones de ellos. El segundo, presenta un perfil específico para usuarios quienes publiquen las mascotas para que sean adoptadas a modo de elección limitada y concisa de las personas que mejor interés tengan. El último, ya es un perfil genérico que se encuentran en busca de mascotas para adoptarlas y darles un hogar digno de un animal, podrán elegir cualquier mascota de su preferencia y disponer de una entrevista mediante el llenado de un formulario en la sección de la mascota, y de tal manera los usuarios que han publicado las mascotas puedan conocer información adicional del usuario común y elegir a la persona indicada para que se lleve a la mascota.

Esta herramienta va a ayudar al usuario dueño de la mascota en una proyección principal de estadía en el *frontend* traído característicamente del *backend* con datos importantes que serán mostrados por parte de la mascota. Después de la recopilación de información por el presente formulario se evaluará y asignará el respectivo dueño que de tal manera se pondrá en contacto y obsequiarle la mascota con un nuevo hogar. Existirá la elección de solamente una mascota a la vez por usuario para la adopción.

1.4 Marco Teórico

Metodología

Una metodología puede hacer énfasis en la variedad de métodos, características y procesos que prioriza en la realización de técnicas útiles con el enfoque de cumplir ciertas

producciones en cuestión de tiempos reducidos. Las metodologías generalmente se clasifican por ser tradicionales y ágiles en cuanto a procesos de desarrollo más formal. Las tradicionales es cuando ya la documentación provee de mucha información que necesita u obliga a ser bien detallada; por otro lado, las ágiles se enfocan en que los procesos sean más rápidos y por lo tanto mucho menos detalle del procedimiento. El objetivo indiferentemente de cual metodología usar, ambas deben responder al ¿Qué? se debe hacer, y no al ¿Cómo? se lo debe hacer; un principio fundamental y tener en claro el tipo de proyecto a realizar [7].

Otra definición, ya en la metodología de proyectos abarca un punto más importante e interesante que es el poder trabajar en conjunto el cual ayude al grupo a idealizar de mejor manera el desarrollo del proyecto. Existen también, los riesgos por el hecho de ejercer periodos de tiempo un poco más largos, mayores discusiones e inversiones; dependiendo de la cantidad de integrantes que se unan o estén [8].

Metodología ágil

Las metodologías ágiles son un factor muy importante al momento de desarrollar un proyecto que dura mucho tiempo, al usar estas metodologías no es necesario tanta descripción; es decir, no se enfoca tanto en el plan que podría durar algún tiempo detallándolo. Se basa más en que el proceso sea rápido, preciso y seguro; tratando de alguna manera respetar los tiempos y verificar que las respuestas a los cambios sean inmediatas en un ciclo repetitivo del desarrollo. De tal manera, ayuda con la integración a nuevos equipos de trabajo y puedan demostrar cada uno las habilidades para poder solventar problemas en el proyecto; esto ayuda a mantener la relación con el cliente para satisfacer las necesidades en el transcurso del proyecto en conjunto [7].

Backend

El *backend* busca una forma de representar la base de datos en el cual se representa en una *API REST* que brinda conexión ya más directa para obtener y presentar los datos en el sistema web. Obviamente, funciona en segundo plano por el hecho de no mostrar el procedimiento que realiza a la vista del usuario; es decir, en segundo plano funciona únicamente entre el ordenador y el servidor [9].

La base para que una *API REST* pueda funcionar sin ningún problema es la cómoda navegación, de tal manera, que los datos se encuentren en constante viaje; se almacenen sin demora y de la misma forma se los exporte. La experiencia con el usuario debe ser amigable en el uso general, y la interacción debe ser rápida. Entonces, la arquitectura

REST es usualmente la más usada porque muestra menos dificultad y es mucho más suave al momento de la implementación; un desarrollo libre [10].

2 METODOLOGÍA

El caso de estudio es una de las técnicas que representa una gran parte de la investigación curiosa para proceder con una buena metodología. Se encarga en lo principal de brindar un buen diseño de calidad acorde a la petición del usuario final; también, integra las técnicas, métodos que se toman a consideración para un buen desarrollo. Entonces, ayuda a validar e indagar por todo el proyecto tomando en cuenta los puntos clave o puntos en los que exista complejidad. El caso de estudio es la parte abstracta pensada por parte del usuario traída hacia la realidad cumpliendo con los objetivos propuestos, que se enlaza con el alcance para la respectiva realización de cada una de las actividades en el proceso del desarrollo del proyecto [11]. El caso de estudio de este proyecto es aplicar una metodología de software ágil [12] en el desarrollo de un sistema web para para los usuarios que quieren adoptar mascotas en la ciudad de Quito, incrustada inicialmente desde la base de datos con métodos de entrada y salida de información gestionados por el *backend*.

2.1 Metodología de Desarrollo

Scrum se define como una metodología bastante útil en el desarrollo ágil de los proyectos, nombrando en esencia cada una de las etapas en ejecución; envuelve todo en una arquitectura entendible y flexible de manipular, describir. Estas etapas comúnmente son llamadas como *Sprints*, y estos son separados por tiempos, estados del desarrollador para realizar las actividades o tareas designadas. En el caso de indicar modificaciones en el transcurso del proyecto estas iteraciones facilitan la rápida actuación para que la duración de estas se mantenga en el límite de las 3 semanas de desarrollo [13].

Roles

Dentro de la metodología ágil *Scrum* ingresan varios aspectos importantes que son llamados roles, estos roles son caracterizados por exigir un incremento y una dedicación en el proyecto. Es decir, cada rol va a realizar su trabajo; en el caso de trabajos grupales todos los miembros van a estar en constante contacto revisando y verificando que se esté avanzando de manera correcta, con responsabilidad. Si se cumple con el producto desarrollado por cada rol existe la posibilidad de una retroalimentación, y de tal manera el equipo de *Scrum* puede enfatizar y mejorar [13].

Product Owner

El *Product Owner* es el principal e inicial manipulador en todo el plan del proyecto. Se encarga de que el equipo de desarrollo presente un buen producto final, es el único que tiene acceso a describir el *Product Backlog* donde se vuelve responsable de la gestión de

este. Entonces, se necesita historias de usuarios para la respectiva lista del funcionamiento del producto; para que relacionen en perfecto estado las metas y alcance en el proceso del *Product Backlog* [13]. Aquel rol, va asignado al Dr. Richard Rivera, la persona quien tiene conocimiento extenso sobre el plan del proyecto.

S*crum Master*

El *Scrum Master* es la persona quien va a liderar y guiar en toda la trayectoria del desarrollo del producto al equipo de desarrollo, cumpliendo los tiempos. Esta persona va a tener clara la visión sobre lo que se va a realizar, dictaminada por el *Product Owner*; también se encarga de compartir tanto el conocimiento teórico como practico y sus reglas para adaptarse a *Scrum* [13]. En este sentido, el rol va asignado al Dr. Richard Rivera, la persona quien brinda toda la información útil al equipo de desarrollo.

Development Team

El *Development Team* se enfoca específicamente en desarrollar el producto de la mejor manera con buenas prácticas de desarrollo. Independientemente de la cantidad de integrantes en el equipo de desarrollo lo que se busca es un incremento del trabajo desarrollado, terminado; es decir, cumpliendo con cada *Sprint* del *Backlog* y así llegar al *Sprint Goal* donde ya se puede poner a producción el producto [13].

En la **TABLA I** se observa las asignaciones respectivas al equipo *Scrum*, con sus referencias tanto en nombres como en roles.

TABLA I Asignación de roles

Rol	**Integrante**
Product Owner	Dr. Richard Rivera
Scrum Master	Dr. Richard Rivera
Development Team	Sr. Carlos Quel Sr. José Luis Colcha

Artefactos

La importancia de entender y proyectar de la mejor manera el proyecto se ve reflejado en los artefactos que son usados. En este caso, la documentación presenta todo el énfasis

del trabajo cumplido que ha permitido tener una buena organización, un valor agregado, las oportunidades de revisión y la transparencia total en el desarrollo [13].

Recopilación de Requerimientos

La importancia de los requerimientos en cualquier proyecto de desarrollo de software tiene el objetivo de satisfacer las necesidades del cliente y alcanzar las funcionalidades finales. También, en la parte de los requerimientos o requisitos presenta todas las actividades que llevan a una validación firme; por lo que aplicando técnicas para la obtención de ellos se logra entender cada tarea [14].

La recopilación de requerimientos para este proyecto ha sido realizada por ideas claves del *Product Owner*, la persona quien describió con claridad las necesidades del proyecto. Y así, proponer una solución relacionada con los requisitos puntuales para este proyecto; la información detallada de todos los requerimientos se encuentra en el **ANEXO II** Manual técnico.

Historias de Usuario

Este estándar de realizar las historias de usuarios viene juntamente con *Scrum*, y no es más que la representación detallada de cada uno de los requisitos. Entonces, ayuda aún más con el mejor entendimiento en una asociación en común con el usuario; que busca calidad y simplicidad en el desarrollo [15]. Cada historia de usuario indica la actividad descrita perteneciente a cada requerimiento como se muestra en la **TABLA II**. Las demás historias de usuarios se presentan en el **ANEXO II** Manual técnico.

TABLA II Historia de Usuario HU001

HISTORIA DE USUARIO	
Identificador (ID): HU001	**Usuario:** Protector y Adoptante
Nombre Historia: Registrar usuario	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 3	
Responsable: Carlos Quel	
Descripción: El protector y adoptante deberá proporcionar la funcionalidad de registro normal o con Google en el *backend* como usuario protector o adoptante, para que el usuario pueda registrarse en el sistema.	
Observación: El registro deberá contener los siguientes campos: nombre, email, contraseña, confirmar contraseña.	

Product Backlog

En el *Product Backlog* existe el orden de las actividades del proyecto cada una con su respectiva priorización, es decir, presenta el listado de los requerimientos del proyecto para que puedan ser supervisados por el *Product Owner*; para determinar el nivel de dificultad y la forma que se va a ir desarrollando para cumplir con la meta. El *Product Backlog* completo se presenta en el **ANEXO II** Manual técnico.

Sprint Backlog

En el *Sprint Backlog* entra el apartado de los *Sprints* que vienen desarrollados por cada una de las actividades en el *Product Backlog*, cuando se concluye con cada iteración se lo representa con el estado de finalizado haciendo énfasis en que el proyecto se va incrementando, y va cumpliendo con los objetivos de un buen producto. Las tareas son designadas por el *Scrum Master* a cada integrante del equipo de desarrollo para que cumplan con las responsabilidades y habilidades dentro de los tiempos establecidos. El *Sprint Backlog* completo se presenta en el **ANEXO II** Manual técnico.

2.2 Diseño de la arquitectura

En el diseño arquitectónico ingresa gran parte fundamental a la solución, desenvolvimiento y proyección del proyecto. Debido a que, muestra la estructura del modelado de los datos para satisfacer las necesidades del cliente. A continuación, se presenta el modelo arquitectónico en el que se está haciendo énfasis para el respectivo desarrollo.

Arquitectura de Datos

En este proyecto, entra el *framework* de Laravel que se basa en relacionar los elementos compuestos por la arquitectura de objetos; y, además permite producir una base de datos firme para las debidas interacciones entre los campos de las entidades de los datos. La adaptación es inmediata y bastante universal por el modelo que los compone, sus funcionalidades enriquecen el desarrollo del *backend* porque el ingreso y salida de la información son muy bien manipuladas. La ejecución es bastante simple y sencilla donde el tiempo esperado de la carga de los componentes es muy corto [16].

En la **Fig. 1** se observa en breve descripción las herramientas que forman parte del proyecto en desarrollo, y además una breve visualización del Modelo Vista Controlador (MVC).

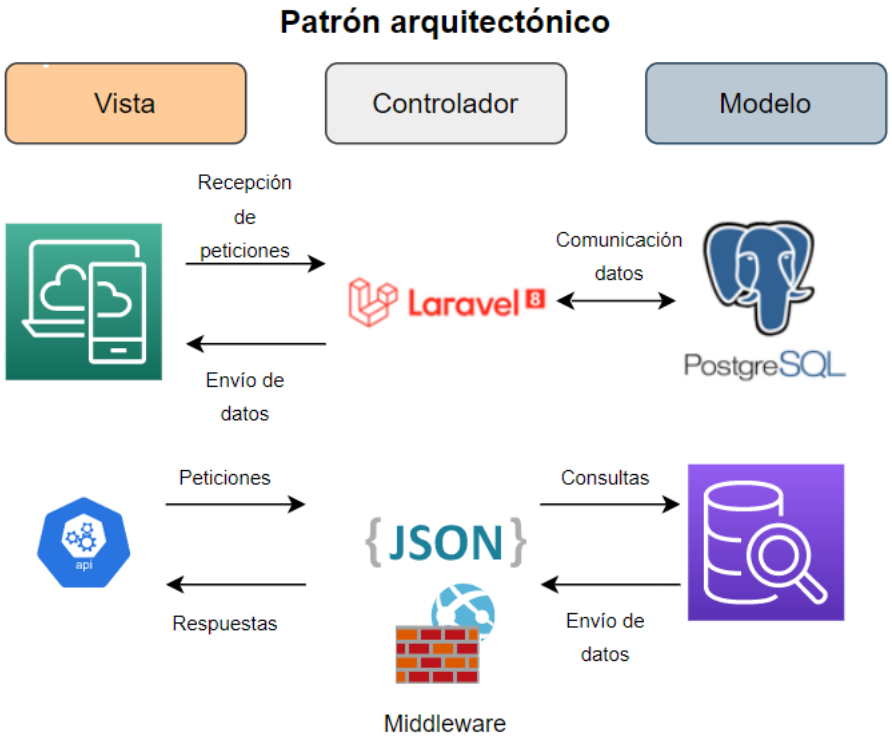

Fig. 1: Patrón arquitectónico de la *API REST*

Patrón arquitectónico Modelo Vista Controlador

El Modelo Vista Controlador (MVC) en las arquitecturas es de mucha y eficaz importancia porque genera patrones de diseño en el cual es el más cercano al modelo de cliente/servidor [17], que es el más utilizado en proyectos por su buen desempeño en el desarrollo de software. El modelo MVC garantiza la separación y relación total entre los elementos que conforman el producto. A continuación, se definen las 3 capas que componen al modelo:

Vista: es la representación del diseño de los componentes resultantes que hace uso el usuario para interactuar con el sistema [18].

Controlador: son todas las acciones generadas por la interacción del usuario con el sistema, de tal manera, que actúa internamente para manejar el sistema [18].

Modelo: es el almacenamiento de los datos que proporciona el usuario final con el objetivo de ofrecer un servicio rápido y seguro [18].

2.3 Herramientas de desarrollo

En la **TABLA III** se puede observar las herramientas que fueron elegidas, debido a la estructura del proyecto y los requerimientos de este; estas herramientas son necesarias para un correcto desarrollo del proyecto.

TABLA III Herramientas fundamentales del proyecto

Herramienta	Descripción	Justificación
Laravel	Laravel está basado en el lenguaje de programación PHP que representa a un *framework* bastante fácil de diseñar y entender la sintaxis, mantiene la relación entre los objetos en desarrollo [16].	El uso de este *framework* en el proyecto es porque tiene una gran demanda de sus componentes de desarrollo que serán muy sencillos de adaptarlos, tendrá presencia en las partes del *backend*.
Composer	Administra todos los paquetes que son empleados en el proyecto, se especializa en que las dependencias estén correctas y funcionando de PHP [16].	Es el más importante porque sin el gestor no se lograría la instalación, actualización de los paquetes para el desarrollo; se hace uso para el buen funcionamiento del *framework*.
XAMPP	Es un paquete completo de las implementaciones para el desarrollo web, brinda los servicios más importantes para las conexiones dentro de ellos están los más nombrados Apache, MySQL y PHP [19].	Es el servicio local en el ordenador que se especificará el almacenamiento de los datos, a que conexiones se van a asociar y los lenguajes por los que se entenderán; incluyen, los puertos a los que se unirán para presentar los servidores.
PostgreSQL	Es un sistema gestor de base de datos relacional, SQL; que permite la integración de datos mediante códigos script para generar el diseño y ejecución de la base de datos general [20].	Se usa este gestor de base de datos porque es simple de manejar, y ofrece la ventaja de cambiarse de gestor de base de datos; y por el libre uso en un servidor público.
Postman	Es un entorno de verificación de pruebas de datos, en el cual va orientado a validar que se puedan obtener y enviar la información mediante el *API* de desarrollo [21].	Es el programa que va a ayudar a la realización de pruebas que significan el funcionamiento de la *API* para la representación de las respuestas que son obtenidas correctamente.
Heroku	Es una plataforma web que brinda los servicio para gestionar servidores; es decir, que permite el almacenamiento de las bases de datos para que puedan ser expuestas a internet [22].	Se presenta el despliegue a producción por medio de este servicio manejable, que enriquece la parte final del proyecto.
Git	Es el sistema de versión controlada, que gestiona o administra el diseño del proyecto desde el código internamente en	La ejecución desde este sistema de control facilita la presentación de las versiones y el orden en el cual se subirá el código del proyecto, se trabaja por ramas y

	segundo plano por parte del ordenador [23].	puede aportar a cambios radicales.
GitHub	Es la plataforma web visual en la cual se toma a consideración las características de crear proyectos colaborativos en repositorios y sin dificultad a la presentación, pueden ser accedidos desde lugares diferentes por medio de internet [23].	Se visualiza el entorno del proyecto más gráficamente, y en cada una de las etapas permite administrar manualmente el código a preferencia del usuario final que acompañará al proyecto finalizado; listo para mandar a producción desde la rama principal.

3 RESULTADOS

A continuación, se detallan procedimientos y alcances obtenidos de las tareas presentadas en cada uno de los *Sprints* en su desarrollo.

Sprint 0. Configuración del ambiente de desarrollo

En cuanto al *Sprint Backlog* inicial hace referencia al entorno de desarrollo en el cual se va a desarrollar el proyecto; obviamente, ingresa la parte fundamental que es la indicación, preparación del correcto funcionamiento relacional. Aquellos, resultados abarcados en este *Sprint* inicial son:

- Diseño de la base de datos.
- XAMPP en funcionamiento.
- PhpStorm en funcionamiento.
- Proyecto en ejecución con Laravel.

Diseño de la base de datos

La creación del diseño se la ha implementado como parte esencial del *backend*. Para lo cual es de mucha importancia las relaciones, entendimiento de las entidades brevemente descritas que acumulan la base de datos con información de primer nivel de satisfacción por el lado del cliente identificado en los requisitos que se logra visualizar en la **Fig. 2**.

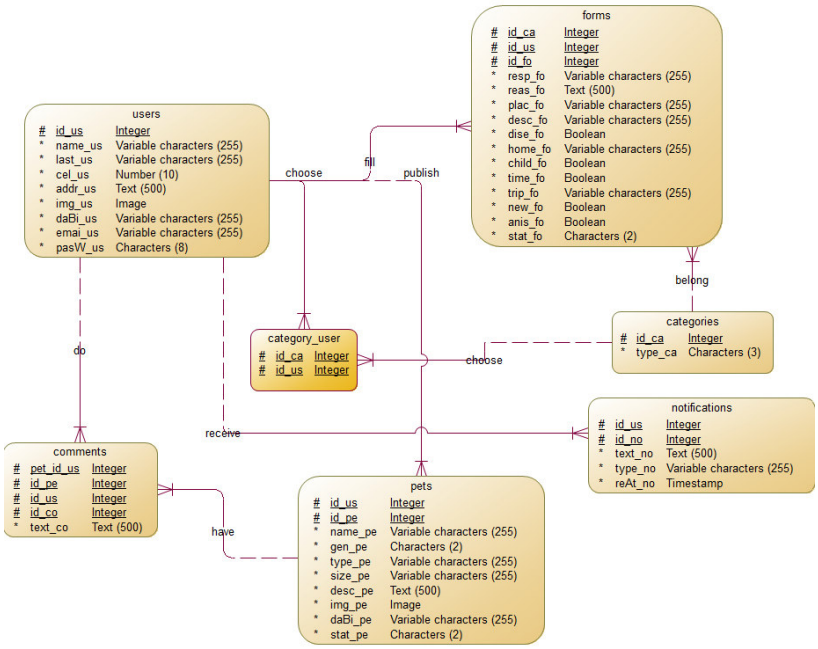

Fig. 2: Base de datos relacional del *API REST*

XAMPP en funcionamiento

Se prepara este programa para que exista un servidor interno en una máquina local. Se configura permitiendo conexiones salientes del *API* hacia la base de datos en modo de prueba con MySQL. Entonces, los únicos que se usan para poner a prueba el *backend* en el proyecto son Apache y MySQL correspondientes como se logra visualizar en la **Fig. 3**.

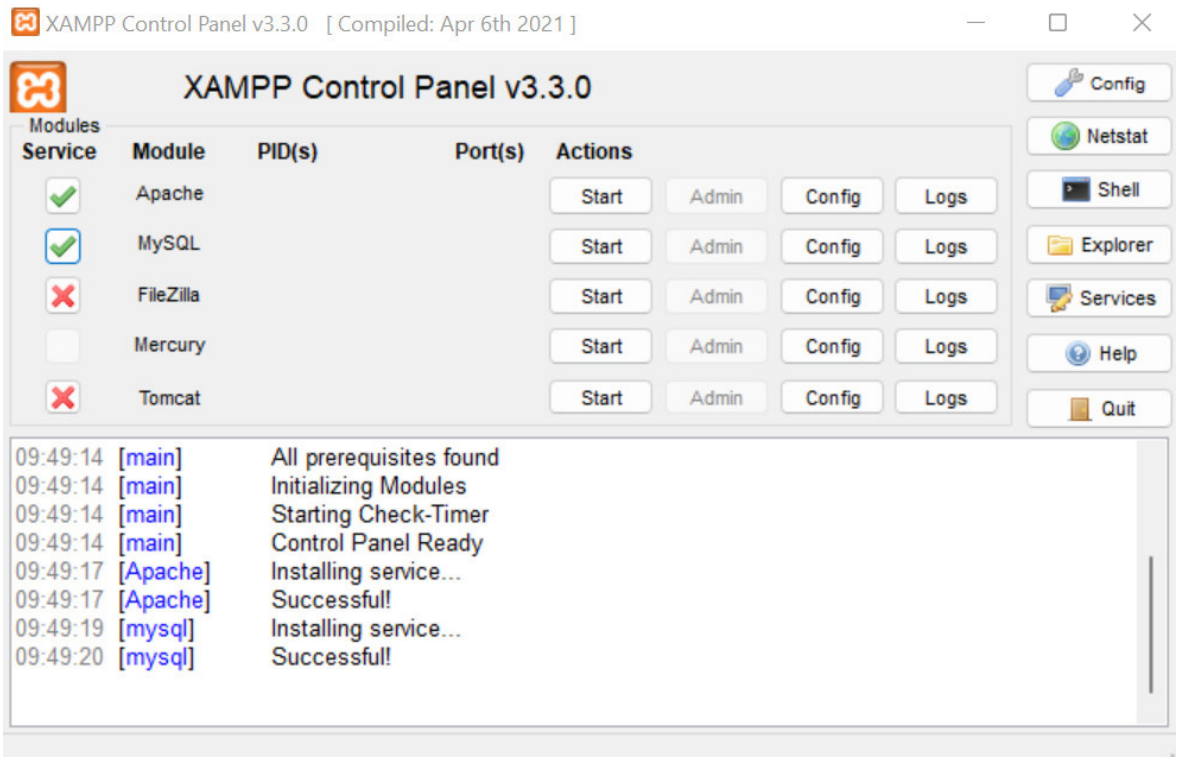

Fig. 3: Verificación del XAMPP

PhpStorm en funcionamiento

Este entorno de desarrollo (siglas en inglés IDE) es bastante bueno para proyectos en el cual se va a usar el lenguaje de programación PHP [24]. El funcionamiento de este IDE es mantener ordenados todos los directorios y ficheros por identificación en el proyecto, en cuanto al código poder identificar cada estructura como se muestra en la **Fig. 4**.

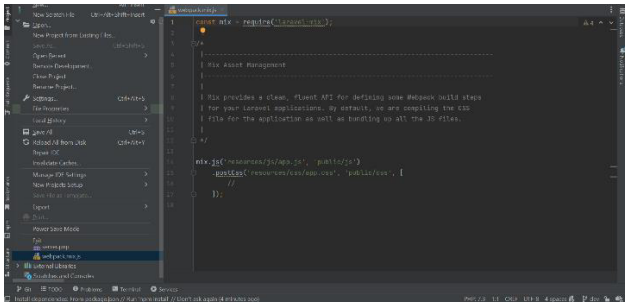

Fig. 4: Validación del IDE PhpStorm

Proyecto en ejecución con Laravel

Ahora mismo, ya se procede con la creación del proyecto inicial en Laravel por lo que es necesario describir bien el nombre del proyecto, completar con la implementación de la estructura de los directorios y ficheros como se muestra en la **Fig. 5**.

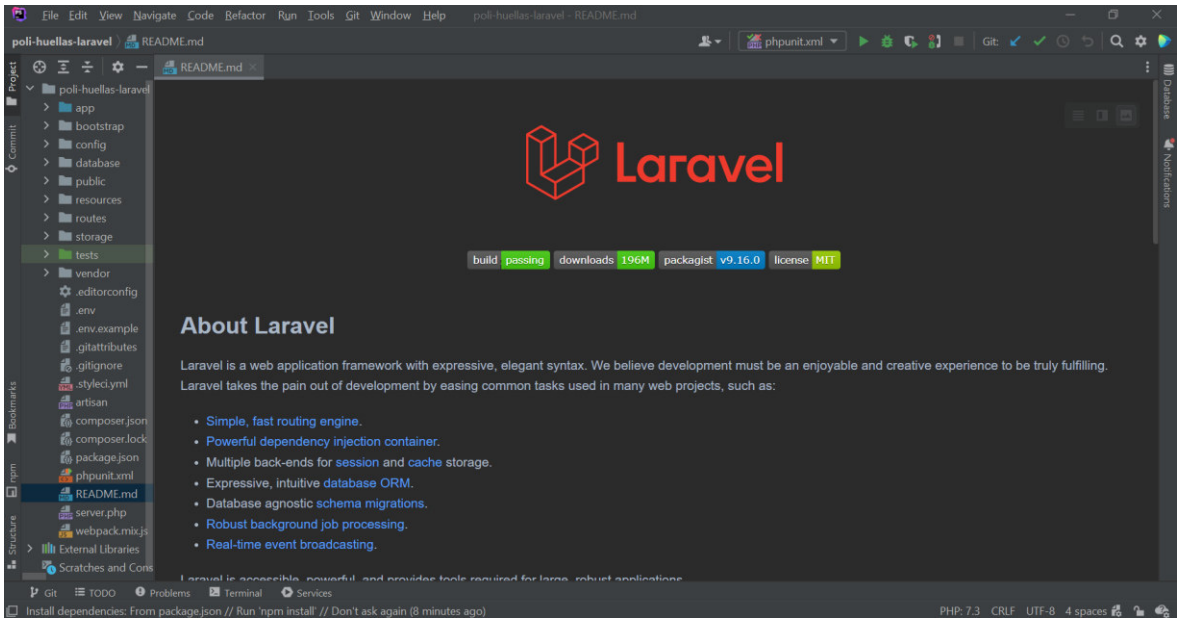

Fig. 5: Proyecto con Laravel inicial

Sprint 1. Implementación de la entidad usuarios con autenticación

En cuanto al *Sprint Backlog* entrante, es en el cual se va a desarrollar el modelo de los usuarios con sus roles y registros, inicios de sesión, cierres de sesión. La finalidad es cumplir con las tareas representadas por este *Sprint* en cada uno de sus procesos; haciendo énfasis en los campos necesarios para alcanzar el objetivo del usuario final. Se establece el orden de creación desde la tabla hasta la ruta que representa el consumo de los datos. Los resultados abarcados en este *Sprint* son:

- Ejecución de la migración y *seeder* de la entidad usuarios por rol con funciones de registro, inicio y cierre de sesión con campos validados.
- Ejecución de las rutas de registro, inicio, cierre de sesión y funciones de presentación de datos del usuario.

Ejecución de la migración y *seeder* de la entidad usuarios por rol con funciones de registro, inicio y cierre de sesión con campos validados

En este paso, se presenta la elaboración de la tabla para usuarios con sus respectivos campos. También, se muestra la elaboración del llenado de datos ficticios a la tabla usuarios que representan usuarios que se han registrado, iniciado sesión. Para lo siguiente, se usa un paquete que permite la autenticación correspondiente al registro e inicio de sesión de los usuarios a través de un token o texto cifrado de verificación; el cual este paquete es llamado como JSON Web Token (siglas en inglés JWT). Entonces, exige una mejor integridad de los datos con grandes seguridades [25]. A continuación, se visualiza en la **Fig. 6** la respectiva representación de la migración en Laravel del modelo usuarios.

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name', 100);
        $table->string('last_name', 100);
        $table->string('cellphone');
        $table->text('address');
        $table->date('date_of_birth');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Fig. 6: Migración usuarios

Lo siguiente a mostrar, es la integración del *seeder* de Laravel al modelo usuarios por rol con la meta de identificar a los usuarios y puedan ser completados de manera automática que verifican el esquema del sistema como se muestra en la **Fig. 7**.

```php
$admin->user()->create([
    'name' => 'Admin', 'last_name' => 'Main', 'cellphone' => '0988020103', 'address' => 'Fcs. de Que
    'image' => $faker->imageUrl(600, 500, null, false), 'date_of_birth' => $faker->date(),
    'email' => 'admin@prueba.com', 'password' => $password, 'role' => 'ROLE_ADMIN'
]);
// Generate some users for app
for ($i = 0; $i < 10; $i++) {
    $protector = Protector::create([
        'company' => $faker->jobTitle,
        'short_bio' => $faker->paragraph
    ]);

    $protector->user()->create([          // Instance
        'name' => $faker->firstName, 'last_name' => $faker->lastName, 'cellphone' => $faker->phoneNu
        'address' => $faker->address, 'image' => $faker->imageUrl(400, 300, null, false),
        'date_of_birth' => $faker->date(), 'email' => $faker->email, 'password' => $password,
        'role' => 'ROLE_PROTECTOR'
    ]);

    $adopter = Adopter::create([
        'company' => $faker->jobTitle,
        'short_bio' => $faker->paragraph
    ]);

    $adopter->user()->create([          // Instance
        'name' => $faker->firstName, 'last_name' => $faker->lastName, 'cellphone' => $faker->phoneNu
        'address' => $faker->address, 'image' => $faker->imageUrl(400, 300, null, false),
        'date_of_birth' => $faker->date(), 'email' => $faker->email, 'password' => $password,
        'role' => 'ROLE_ADOPTER'
    ]);
```

Fig. 7: *Seeder* usuarios

Ya en este punto, el enfoque va dirigido al código referente al manejo de la autenticación desde el paquete JWT; desarrollado de manera manual y adoptada en el proyecto por medio de su respectiva implementación sencilla como se muestra en la **Fig. 8**.

```php
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Tymon\JWTAuth\Contracts\JWTSubject;

class User extends Authenticatable implements JWTSubject
{
```

Fig. 8: JWT usuarios

Las funciones entran en un ambiente de desarrollo más específico el cual presentan la lógica del funcionamiento de las rutas para consumir los datos y que puedan ser expuestos al usuario. Entonces, Laravel ofrece esta implementación en controladores que permiten la conexión entre la base de datos con el *API* de manera rápida y segura; validando cada uno de sus campos requeridos en el sistema como se visualiza en la **Fig. 9**.

```php
public function authenticate(Request $request)
{
    $credentials = $request->only('email', 'password');
    try {
        if (!$token = JWTAuth::attempt($credentials)) {
            return response()->json(['message' => 'invalid_credentials'], 400);
        }
    } catch (JWTException $e) {
        return response()->json(['message' => 'could_not_create_token'], 500);
    }
    $user = JWTAuth::user();

    return response()->json(compact('token', 'user'))
        ->withCookie(
            'token',
            $token,
            config('jwt.ttl'), // ttl => time to live
            '/', // path
            null, // domain
            config('app.env') !== 'local', // Secure
            true, // httpOnly
            false, //
            config('app.env') !== 'local' ? 'None' : 'Lax' // SameSite
        );
}
```

Fig. 9: Inicio de sesión usuarios controlador

El registro también mantiene la validación de los campos requeridos, de igual manera se encuentran implementados en el controlador como se muestra en la **Fig. 10**.

```php
public function register(Request $request)
{
    $request->validate([
        'name' => 'required|string|max:100',
        'last_name' => 'required|string|max:100',
        'cellphone' => 'required|string|max:10',
        'address' => 'required',
        'image' => 'required|image|dimensions:min_width=200,min_height=200',
        'date_of_birth' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:6|confirmed',
        'company' => 'required|string',
        'short_bio' => 'required|string',
        'role' => 'required'
    ]);

    if ($request->role == User::ROLE_PROTECTOR) {
        $userable = Protector::create([
            'company' => $request->get('company'),
            'short_bio' => $request->get('short_bio'),
        ]);
    } elseif ($request->role == User::ROLE_ADOPTER) {
        $userable = Adopter::create([
            'company' => $request->get('company'),
            'short_bio' => $request->get('short_bio'),
        ]);
    } else {
        $userable = Admin::create([
            'identity_card' => $request->get('identity_card'),
        ]);
```

Fig. 10: Registro usuarios controlador

El cierre de sesión mucho más simple, pero muy importante de implementar funciona para sesiones activas que desean inactivar la sesión hasta una actualización de autenticación como se visualiza en la **Fig. 11**.

```php
public function logout()
{
    try {
        JWTAuth::invalidate(JWTAuth::getToken());

          Cookie::queue(Cookie::forget('token'));
          $cookie = Cookie::forget('token');
          $cookie->withSameSite('None');
        return response()->json([
            "status" => "success",
            "message" => "User successfully logged out."
        ], 200)
            ->withCookie('token', null,
                config('jwt.ttl'),
                '/',
                null,
                config('app.env') !== 'local',
                true,
                false,
                config('app.env') !== 'local' ? 'None' : 'Lax'
            );
    } catch (JWTException $e) {
        // something went wrong whilst attempting to encode the token
        return response()->json(["message" => "No se pudo cerrar la sesión."], 500);
    }
}
```

Fig. 11: Cierre de sesión usuarios controlador

Para visualizar a los usuarios, se realiza una función extra que permite la obtención de los autenticados y exponerlos por medio de las rutas. A esto se le añade que la autenticación es de mucho valor para la respectiva presentación de los usuarios a validar; al existir errores de la misma forma es representado por mensajes de exhibición como se muestra en la **Fig. 12**.

```php
public function getAuthenticatedUser()
{
    try {
        if (!$user = JWTAuth::parseToken()->authenticate()) {
            return response()->json(['message' => 'user_not_found'], 404);
        }
    } catch (Tymon\JWTAuth\Exceptions\TokenExpiredException $e) {
        return response()->json(['message' => 'token_expired'], $e->getStatusCode());
    } catch (Tymon\JWTAuth\Exceptions\TokenInvalidException $e) {
        return response()->json(['message' => 'token_invalid'], $e->getStatusCode());
    } catch (Tymon\JWTAuth\Exceptions\JWTException $e) {
        return response()->json(['message' => 'token_absent'], $e->getStatusCode());
    }
    return response()->json(new UserResource($user), 200);
}
```

Fig. 12: Obtención usuarios controlador

Ejecución de rutas de registro, inicio, cierre de sesión y funciones de presentación de datos del usuario

Una de las mejores partes son las rutas, tanto para registrar como para iniciar sesión se ha implementado con correo electrónico del sistema mismo y con servicios de Google; debido a que, es la fuente para el funcionamiento del sistema donde interactúa y logra intercambiar información concurrente en segundo plano. Normalmente, las peticiones son varias y de distintos tipos como: *POST*, *GET*, *PUT*, *DELETE*; que son recibidas y manipuladas por los controladores como se visualiza en la **Fig. 13**.

```php
//Public routes
Route::post('register', [UserController::class, 'register']);
Route::post('login', [UserController::class, 'authenticate']);
Route::get('auth/google', [GoogleController::class, 'redirectToGoogle']);
Route::get('auth/google/callback', [GoogleController::class, 'handleGoogleCallback']);
//Private routes
Route::group(['middleware' => ['jwt.verify']], function() {
    Route::post('logout', [UserController::class, 'logout']);
```

Fig. 13: Rutas de autenticación usuarios

Para obtener una visualización de los datos de los usuarios, presenta una ruta también. Esta ruta es bastante especifica y nueva que usa el controlador como un aliado para

verificar la sesión y presentar los datos del usuario mediante la autenticación válida como indica en la **Fig. 14**.

```
//Private routes
Route::group(['middleware' => ['jwt.verify']], function() {
    Route::post('logout', [UserController::class, 'logout']);
    Route::get('user', [UserController::class, 'getAuthenticatedUser']);
});
```

Fig. 14: Ruta datos usuario

El administrador y el protector tienen el beneficio de observar todos los usuarios para manipularlos, y de tal manera potenciar los datos en modo de prueba. Incluso, tienen permisos para desvincular o contactar en el sistema como se muestra en la **Fig. 15**.

```
Route::get('users', [UserController::class, 'index']);
Route::get('users/{user}', [UserController::class, 'show']);
Route::get('users/{user}', [UserController::class, 'update']);
Route::get('users/{user}', [UserController::class, 'delete']);
```

Fig. 15: Rutas administrador y protectores

Sprint 2. Implementación de la entidad mascotas con detalle

La continuación del *Sprint Backlog* es en el cual se va a desarrollar el modelo de mascotas con detalle. Entonces, el funcionamiento y relación con las rutas es esencial en las tareas del *Sprint* 2 presentadas; la relación debe ser juntamente con cada usuario en su mascota respectivamente. Los resultados abarcados en el *Sprint* son:

- Ejecución migración, controlador, *seeder* y rutas para las funciones crear, mostrar, editar y eliminar del modelo mascotas.
- Implementación de los campos de detalle al modelo mascotas.
- Relación en la migración y el *seeder* para el campo de mascotas en el modelo usuario.

Ejecución migración, controlador, *seeder* y rutas para las funciones crear, mostrar, editar y eliminar del modelo mascotas

La migración como principal implementación presenta campos que identificarán a las mascotas respectivas al modelo. Bastante sencillo en la elaboración, mediante comandos rápidos por parte de Laravel la cual crea la entidad en la base de datos como se visualiza en la **Fig. 16**.

```php
public function up()
{
    Schema::create( table: 'pets', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'name',  length: 100);
        $table->text( column: 'description');
        $table->boolean( column: 'adopted')->default( value: 0);
        $table->timestamps();
    });
}
```

Fig. 16: Migración mascotas

A modo de simplificación, y de presentación funcional se realiza la parte del *seeder* en el cual mantiene datos falsos para ver el funcionamiento del modelo mascotas. Estos datos falsos pueden ser escogidos y desarrollados de manera automática como se muestra en la **Fig. 17**.

```php
for ($j = 0; $j < $num_pets; $j++) {
    Pet::create([
        'name' => $faker->name,
        'description' => $faker->paragraph,
        'image' => $faker->imageUrl( width: 400,  height: 300,  category: null,  randomize: false)
    ]);
}
```

Fig. 17: *Seeder* mascotas

Lo siguiente, es la creación del controlador referencial al modelo mascotas; que incluye funcionalidades para las rutas respectivas. En este caso, la importancia de la visualización es de gran nivel por el que se pueden obtener los datos y ser presentados al usuario desde una interfaz gráfica. Entonces, recibe la lista de mascotas o mascota como se muestra en la **Fig. 18**.

```php
public function index()
{
    $this->authorize( ability: 'viewAny',  arguments: Pet::class);
    return new PetCollection(Pet::paginate(5));
}
public function image(Pet $pet)
{
    return response()->download(public_path(Storage::url($pet->image)), $pet->name)
}
public function show(Pet $pet) // Relationship with an instance that corresponded f
{
    $this->authorize( ability: 'view', $pet);
    return response()->json(new PetResource($pet),  status: 200);
}
```

Fig. 18: Visualización mascotas controlador

Las siguientes funcionalidades para implementar son: crear y editar. Cada funcionalidad implementa la validación de los datos, y que puedan ser correctos; de tal manera, que son enviados para ser almacenados con respuestas del estado de guardado como se muestra en la **Fig. 19**.

```php
public function store(Request $request)
{
    $this->authorize( ability: 'create', arguments: Pet::class);
    $rules = ['name' => 'required|string|unique:pets|max:100', 'description' => 'require
    $request->validate($rules, self::$messages);
    $pet = new Pet($request->all()); // New instance with data
    $pet->save();
    return response()->json(new PetResource($pet), status: 201); // New instance
}

public function update(Request $request, Pet $pet)
{
    $this->authorize( ability: 'update', $pet); // instance of that article
    $rules = ['name' => 'required|string|unique:pets,name,'.$pet->id.'|max:100',]; // Al
    $request->validate($rules, self::$messages);
    $pet->update($request->all()); // It's updating  directly without looking for in DB
```

Fig. 19: Creación y actualización de mascotas controlador

Para finalizar el funcionamiento del modelo de mascotas, es de mucha utilidad la eliminación de una mascota o varias dentro de la base de datos como se visualiza en la **Fig. 20**.

```php
public function delete(Pet $pet)
{
    $this->authorize( ability: 'delete', $pet);
    $pet->delete(); // It's deleting
    return response()->json( data: null, status: 204); // Empty content or nothing, al
}
```

Fig. 20: Eliminación mascota controlador

Lo siguiente, es concluir con las rutas para que sean manejadas desde fuentes externas. La lógica de las rutas ya está en el controlador que funciona para permitir al usuario hacer uso y guardar los datos por peticiones al servidor como se muestra en la **Fig. 21**.

```php
// Pets
Route::get( uri: 'pets', [PetController::class, 'index']);
Route::get( uri: 'pets/{pet}', [PetController::class, 'show']);
Route::get( uri: 'pets/{pet}/image', [PetController::class, 'image']);
Route::post( uri: 'pets', [PetController::class, 'store']);
Route::put( uri: 'pets/{pet}',  [PetController::class, 'update']);
Route::delete( uri: 'pets/{pet}', [PetController::class, 'delete']);
```

Fig. 21: Rutas mascotas controlador

Implementación de los campos de detalle al modelo mascotas

Para el proyecto es esencial presentar más información detallada principalmente de las mascotas. Por tal motivo, se incentivó a añadir campos extras al modelo mascotas que se logren identificar y mostrar al usuario. Es decir, cada mascota pueda ser mejor presentada por cada usuario con características fundamentales como se muestra en la **Fig. 22**.

```php
public function up()
{
    Schema::create( table: 'pets', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'name',  length: 100);
        $table->enum( column: 'gender', ['Macho', 'Hembra']);
        $table->enum( column: 'type', ['Perro', 'Gato', 'Otros']);
        $table->enum( column: 'size', ['Pequeño', 'Mediano', 'Grande']);
        $table->text( column: 'description');
        $table->date( column: 'date_of_birth');
        $table->boolean( column: 'adopted')->default( value: 0);
        $table->timestamps();
    });
}
```

Fig. 22: Migración completa detalle de mascotas

Lo siguiente, es modificar el *seeder* acorde a los campos añadidos por detalle de las mascotas presentados anteriormente. La base de datos hace uso de más datos para ser añadidos con datos ficticios correspondientes a ser almacenados en modo de prueba como se muestra en la **Fig. 23**.

```php
for ($j = 0; $j < $num_pets; $j++) {
    Pet::create([
        'name' => $faker->name,
        'gender' => $faker->randomElement(['Macho', 'Hembra']),
        'type' => $faker->randomElement(['Perro', 'Gato', 'Otros']),
        'size' => $faker->randomElement(['Pequeño', 'Mediano', 'Grande']),
        'description' => $faker->paragraph,
        'date_of_birth' => $faker->date(),
        'image' => $faker->imageUrl( width: 400, height: 300, category: null, randomize: false)
    ]);
}
```

Fig. 23: *Seeder* completo detalle de mascotas

El funcionamiento vario en la función crear el cual depende aún más de los campos añadidos al detalle de las mascotas, que deberán ser validados también como se visualiza en la **Fig. 24**.

```php
public function store(Request $request)
{
    $this->authorize( ability: 'create', arguments: Pet::class);
    $rules = [
        'name' => 'required|string|unique:pets|max:100', 'gender' => 'in:Macho,Hembra',
        'type' => 'in:Perro,Gato,Otros', 'size' => 'in:Pequeño,Mediano,Grande',
        'description' => 'required', 'date_of_birth' => 'required|string|max:255',
        'image' => 'required|image|dimensions:min_width=200,min_height=200'
    ];
    $request->validate($rules, self::$messages);
    $pet = new Pet($request->all()); // New instance with data
    $path = $request->image->store('public/pets'); // upload file to server | route ->
    $pet->image = $path; // Field image
    $pet->save();
    return response()->json(new PetResource($pet), status: 201); // New instance
}
```

Fig. 24: Creación con detalle de mascotas

El funcionamiento vario en la función editar el cual depende aún más de los campos añadidos al detalle de las mascotas, que deberán ser validados también como se visualiza en la **Fig. 25**.

```php
public function update(Request $request, Pet $pet)
{
    $this->authorize( ability: 'update', $pet); // instance of that pet
    $rules = [
        'name' => 'required|string|unique:pets,name,'.$pet->id.'|max:100', // Allow t
        'gender' => 'in:Macho,Hembra', 'type' => 'in:Perro,Gato,Otros',
        'size' => 'in:Pequeño,Mediano,Grande', 'description' => 'required',
        'date_of_birth' => 'required|string|max:255',
    ];
    $request->validate($rules, self::$messages);
    $pet->update($request->all()); // It's updating, directly without looking for in
    return response()->json($pet,  status: 200);
}
```

Fig. 25: Actualización con detalle de mascotas

Relación en la migración y el *seeder* para el campo de mascotas en el modelo usuario

La relación de la mascota funciona desde atributos referenciales a las tablas, que se incrusta en el modelo usuario con su migración respectiva. La modificación va específicamente en la migración de usuarios que relaciona los campos como se muestra en la **Fig. 26**.

```php
public function up()
{
    Schema::table( table: 'users', function (Blueprint $table) {
        $table->integer( column: 'userable_id');
        $table->string( column: 'userable_type');
    });
}
```

Fig. 26: Relación campos en el modelo usuarios

Sprint 3. Implementación de la entidad comentarios

Nuevamente el *Sprint Backlog* presenta tareas en relación con el *Sprint* 3, que se enfoca en la elaboración de los modelos comentarios y formularios para el desarrollo del proyecto. Los resultados abarcados en este *Sprint* son:

- Verificación de validación de datos en el controlador usuario.
- Codificación roles en modelo usuarios.
- Ejecución migración, controlador, *seeder* y rutas para las funciones crear, mostrar, editar y eliminar del modelo comentarios.

Verificación de validación de datos en el controlador usuario

En esta fase se comprueba que la validación es correcta, y presenta mensajes válidos del funcionamiento. De tal forma, exige un control de los datos del usuario al ser validados para poder ser almacenados en la base de datos por medio de la interfaz de usuario como se indica una breve revisión con el rol correspondiente en la **Fig. 27**.

```php
$request->validate([
    'name' => 'required|string|max:100',
    'last_name' => 'required|string|max:100',
    'cellphone' => 'required|string|max:10',
    'address' => 'required',
    'image' => 'required|image|dimensions:min_width=200,min_height=200',
    'date_of_birth' => 'required|string|max:255',
    'email' => 'required|string|email|max:255|unique:users',
    'password' => 'required|string|min:6|confirmed',
    'company' => 'required|string',
    'short_bio' => 'required|string',
    'role' => 'required'
]);
```

Fig. 27: Validación campos modelo usuarios

Codificación roles en modelo usuarios

Esta parte es la más interesante e importante del proyecto; debido a que, el sistema está basado para presentar tres perfiles. Cada perfil, permite sus actividades que serán relacionadas a lo que se pueda hacer o no; por parte del usuario. Como bien se presenta, el perfil administrador tiene todos los privilegios, en cuanto al perfil protector y adoptante tienen permisos no tan desarrollados con una limitación clara.

Los roles a continuación presentados, se basan específicamente en el desarrollo del sistema; por lo cual se encuentran dentro del modelo usuarios como se visualiza en la **Fig. 28**.

```php
// With auth
const ROLE_SUPERADMIN = 'ROLE_SUPERADMIN';
const ROLE_ADMIN = 'ROLE_ADMIN';
const ROLE_PROTECTOR = 'ROLE_PROTECTOR';
const ROLE_ADOPTER = 'ROLE_ADOPTER';

private const ROLES_HIERARCHY = [
    self::ROLE_SUPERADMIN => [self::ROLE_ADMIN],
    self::ROLE_ADMIN => [self::ROLE_PROTECTOR, self::ROLE_ADOPTER],
    self::ROLE_PROTECTOR => [],
    self::ROLE_ADOPTER => []
];
```

Fig. 28: Roles modelo usuarios

El funcionamiento del rol por usuario mantiene la lógica en las funciones que identifica los roles con los respectivos permisos en orden jerárquico como se muestra en la **Fig. 29**.

```php
public function isGranted($role)
{
    if ($role === $this->role) {
        return true;
    }
    return self::isRoleInHierarchy($role, self::ROLES_HIERARCHY[$this->role]);
}

private static function isRoleInHierarchy($role, $role_hierarchy)
{
    if (in_array($role, $role_hierarchy)) {
        return true;
    }
    foreach ($role_hierarchy as $role_included) {
        if(self::isRoleInHierarchy($role,self::ROLES_HIERARCHY[$role_included]))
        {
            return true;
        }
    }
    return false;
}
```

Fig. 29: Funciones jerarquía roles

A causa de la implementación de roles, se incrusta un nuevo atributo al modelo usuarios para identificar el rol directamente y almacenar en la base de datos a que rol pertenece en numeración en el registro del usuario al sistema como se muestra en la **Fig. 30**.

```php
public function up()
{
    Schema::table( table: 'users', function (Blueprint $table) {
        $table->enum( column: 'role', [User::ROLE_ADMIN, User::ROLE_PROTECTOR, User::ROLE_ADOPTER]);
    });
}
```

Fig. 30: Campo rol en usuarios

Ejecución migración, controlador, *seeder* y rutas para las funciones crear, mostrar, editar y eliminar del modelo comentarios

La migración como principal implementación presenta campos que identificarán a los comentarios respectivos al modelo. Bastante sencillo en la elaboración, mediante

comandos rápidos por parte de Laravel la cual crea la entidad en la base de datos como se visualiza en la **Fig. 31**.

```php
public function up()
{
    Schema::create( table: 'comments', function (Blueprint $table) {
        $table->id();
        $table->text( column: 'text');
        $table->foreignId( column: 'pet_id')->constrained( table: "pets")->onDelete( action: 'cascade');
        $table->foreignId( column: 'user_id')->constrained( table: "users")->onDelete( action: 'cascade');
        $table->timestamps();
    });
}
```

Fig. 31: Migración comentarios

A modo de simplificación, y de presentación funcional se realiza la parte del *seeder* en el cual mantiene datos falsos para ver el funcionamiento del modelo comentarios. Estos datos falsos pueden ser escogidos y desarrollados de manera automática como se muestra en la **Fig. 32**.

```php
public function run()
{
    //
    // Vaciamos la tabla comments
    Comment::truncate();
    $faker = \Faker\Factory::create(); // Instance for faker
    // Obtenemos todos los artículos de la bdd
    $pets = Pet::all();
    // Obtenemos todos los usuarios
    $users = User::all();
    // Iterar
    foreach ($users as $user) {
        // iniciamos sesión con cada uno
        JWTAuth::attempt(['email' => $user->email, 'password' => '12312312']);
        // Creamos un comentario para cada mascota con este usuario
        foreach ($pets as $pet) {
            Comment::create([
                'text' => $faker->paragraph, // Text of the comment (Paragraph)
                'pet_id' => $pet->id, // Create the "id" to the owner of the comm
            ]);
        }
    }
}
```

Fig. 32: *Seeder* comentarios

Lo siguiente, es la creación del controlador referencial al modelo comentarios; que incluye funcionalidades para las rutas respectivas. En este caso, la importancia de la visualización es de gran nivel por el que se pueden obtener los datos y ser presentados al usuario desde una interfaz gráfica. Entonces, recibe la lista de comentarios o comentario como se muestra en la **Fig. 33**.

```php
public function index(Pet $pet)
{
    $comment = $pet->comments;
    return response()->json(CommentResource::collection($comment), status: 200);
}

/**
 * @param Pet $pet
 * @param Comment $comment
 * @return \Illuminate\Http\JsonResponse
 */
public function show(Pet $pet, Comment $comment)
{
    $comments = $pet->comments()->where( column: 'id', $comment->id)->firstOrFail();
    return response()->json(new CommentResource($comments), status: 200);
}
```

Fig. 33: Visualización comentarios controlador

Las siguientes funcionalidades para implementar son: crear y editar. Cada funcionalidad de los comentarios implementa la validación de los datos, y que puedan ser correctos; de tal manera, que son enviados para ser almacenados con respuestas del estado de guardado como se muestra en la **Fig. 34**.

```php
public function store(Request $request, Pet $pet)
{
    $request->validate([
        'text' => 'required|string',
    ]);
    $comments = $pet->comments()->save(new Comment($request->all())); //
    $pet->user->notify(new NewCommentNotification($pet));
    return response()->json(new CommentResource($comments), status: 201);
}

public function update(Request $request, Pet $pet)
{
    $pet->comments()->update($request->all()); // It's updating, directl
    return response()->json($pet, status: 200);
}
```

Fig. 34: Creación y actualización de comentarios controlador

Para finalizar el funcionamiento del modelo de comentarios, es de mucha utilidad la eliminación de un comentario o varias dentro de la base de datos como se visualiza en la **Fig. 35**.

```php
public function delete(Pet $pet)
{
    $pet->comments()->delete(); // It's deleting
    return response()->json( data: null,  status: 204); // Empty content or nothing, all okay.
}
```

Fig. 35: Eliminación comentario controlador

Lo siguiente, es concluir con las rutas de los comentarios para que sean manejadas desde fuentes externas. La lógica de las rutas ya está en el controlador que funciona para permitir al usuario hacer uso y guardar los datos por peticiones al servidor como se muestra en la **Fig. 36**.

```php
// Comments
Route::get( uri: 'pets/{pet}/comments', [CommentController::class, 'index']);
Route::get( uri: 'pets/{pet}/comments/{comment}', [CommentController::class, 'show']);
Route::post( uri: 'pets/{pet}/comments', [CommentController::class, 'store']);
Route::put( uri: 'pets/{pet}/comments/{comment}',  [CommentController::class, 'update']);
Route::delete( uri: 'pets/{pet}/comments/{comment}', [CommentController::class, 'delete']);
```

Fig. 36: Rutas comentarios controlador

Sprint 4. Implementación de las entidades formularios, notificaciones y políticas

A manera de continuar con la parte ya casi final del *Sprint Backlog*, se ejercen las tareas del *Sprint* 4. Fundamentalmente existe nueva lógica corta de rutas, y tratar de cumplir con las políticas o permisos del sistema. Los resultados abarcados en este *Sprint* son:

- Ejecución migración, controlador, *seeder* y rutas para las funciones crear, mostrar, editar y eliminar del modelo formularios.
- Ejecución migración notificaciones.
- Realización ruta para las funciones notificar y leer del modelo notificaciones.
- Codificación de la base de datos relacional.
- Codificación para políticas del sistema.

Ejecución migración, controlador, *seeder* y rutas para las funciones crear, mostrar, editar y eliminar del modelo formularios

La migración como principal implementación presenta campos que identificarán a los formularios respectivos al modelo. Bastante sencillo en la elaboración, mediante comandos rápidos por parte de Laravel la cual crea la entidad en la base de datos como se visualiza en la **Fig. 37**.

Fig. 37: Migración formularios

A modo de simplificación, y de presentación funcional se realiza la parte del *seeder* a partir de las mascotas y las categorías en el cual mantiene datos falsos para ver el funcionamiento del modelo formularios. Estos datos falsos pueden ser escogidos y desarrollados de manera automática como se muestra en la **Fig. 38**.

```php
foreach ($pets as $pet) {
    Form::create([
        'responsible' => $faker->name,
        'reason' => $faker->paragraph,
        'home' => $faker->word,
        'description' => $faker->paragraph,
        'diseases' => $faker->boolean,
        'children' => $faker->boolean,
        'time' => $faker->boolean,
        'trip' => $faker->word,
        'new' => $faker->boolean,
        'animals' => $faker->boolean,
        'pet_id' => $pet->id,
        'category_id' => $faker->numberBetween( int1: 1, int2: 3),
    ]);
}
```

Fig. 38: *Seeder* formularios

Lo siguiente, es la creación del controlador referencial al modelo formularios; que incluye funcionalidades para las rutas respectivas. En este caso, la importancia de la visualización es de gran alcance por el que se pueden obtener los datos y ser presentados al usuario desde una interfaz gráfica. Entonces, recibe la lista de formularios o formulario como se muestra en la **Fig. 39**.

```php
public function index(Pet $pet)
{
    $form = $pet->forms;
    return response()->json(FormResource::collection($form), status: 200);
}

/**
 * Display the specified resource.
 *
 * @param  \App\Models\Form  $form
 * @return \Illuminate\Http\Response
 */
public function show(Pet $pet, Form $form)
{
    $this->authorize( ability: 'view', $form);
    $forms = $pet->forms()->where( column: 'id', $form->id)->firstOrFail();
    return response()->json(new FormResource($forms), status: 200);
}
```

Fig. 39: Visualización formularios controlador

Las siguientes funcionalidades para implementar son: crear y editar. Cada funcionalidad del formulario implementa la validación de los datos, y que puedan ser correctos; de tal manera, que son enviados para ser almacenados con respuestas del estado de guardado como se muestra en la **Fig. 40**.

```php
public function store(Request $request, Pet $pet)
{
    $this->authorize( ability: 'create',  arguments: Form::class);
    $rules = [
        'responsible' => 'required|string|max:100', 'reason' => 'required', 'home'
    ];
    $request->validate($rules, self::$messages);
    $forms = $pet->forms()->save(new Form($request->all())); // New instance with
    auth()->user()->notify(new FormNotification($forms));
    return response()->json(new FormResource($forms), status: 201); // New instance
}
public function update(Request $request, Pet $pet)
{
    $this->authorize( ability: 'update',  arguments: Form::class); // instance of that ar
    $rules = [
        'responsible' => 'required|string|max:100', 'reason' => 'required', 'home'
    ];
    $request->validate($rules, self::$messages);
    $forms = $pet->forms()->update($request->all());
    return response()->json($forms,  status: 200);
}
```

Fig. 40: Creación y actualización de formularios controlador

Para finalizar el funcionamiento del modelo de formularios, es de mucha utilidad la eliminación de un formulario o varios dentro de la base de datos como se visualiza en la **Fig. 41**.

```
public function delete(Pet $pet)
{
    $this->authorize( ability: 'delete',  arguments: Form::class);
    //$form->delete(); // It's deleting
    $pet->forms()->delete();
    return response()->json( data: null,  status: 204); // Empty
}
```

Fig. 41: Eliminación formulario controlador

Lo siguiente, es concluir con las rutas de formularios para que sean manejadas desde fuentes externas. La lógica de las rutas ya está en el controlador que funciona para permitir al usuario hacer uso y guardar los datos por peticiones al servidor como se muestra en la **Fig. 42**.

```
// Forms
Route::get( uri: 'pets/{pet}/forms', [FormController::class, 'index']);
Route::get( uri: 'pets/{pet}/forms/{form}', [FormController::class, 'show']);
Route::post( uri: 'pets/{pet}/forms', [FormController::class, 'store']);
Route::put( uri: 'pets/{pet}/forms/{form}',  [FormController::class, 'update']);
Route::delete( uri: 'pets/{pet}/forms/{form}', [FormController::class, 'delete']);
```

Fig. 42: Rutas formularios controlador

Ejecución migración notificaciones

La migración como principal implementación presenta campos que identificarán a las notificaciones respectivas. Bastante sencillo en la elaboración, mediante comandos rápidos por parte de Laravel la cual crea la entidad en la base de datos como se visualiza en la **Fig. 43**.

```php
public function up()
{
    Schema::create( table: 'notifications', function (Blueprint $table) {
        $table->uuid( column: 'id')->primary();
        $table->string( column: 'type');
        $table->morphs( name: 'notifiable');
        $table->text( column: 'data');
        $table->timestamp( column: 'read_at')->nullable();
        $table->timestamps();
    });
}
```

Fig. 43: Migración notificaciones

Realización ruta para las funciones notificar y leer del modelo notificaciones

Lo siguiente, es concluir con la ruta de notificaciones leídas para que sean manejadas desde fuentes externas. La lógica de las rutas ya está implementada internamente por parte de Laravel para notificaciones automáticas que funcionan para permitir al usuario hacer uso de los datos por peticiones al servidor como se muestra en la **Fig. 44**.

```php
// Notifications
Route::get( uri: 'markAsRead', function() {
    auth()->user()->unreadNotifications->markAsRead();
});
```

Fig. 44: Ruta notificaciones

Codificación de la base de datos relacional

En la implementación de la relación más esencial en el proyecto ha sido entre la tabla categorías y usuarios, que rompe la relación para lo cual se crea inicialmente la migración como principal elemento que presenta campos que identificará la relación a representar. Bastante sencillo en la elaboración, mediante comandos rápidos por parte de Laravel la cual crea la entidad en la base de datos como se visualiza en la **Fig. 45**.

```php
Schema::create( table: 'category_user', function (Blueprint $table) {
    $table->foreignId( column: 'category_id')->constrained( table: 'categories')->onDelete( action: 'cascade');
    $table->foreignId( column: 'user_id')->constrained( table: 'users')->onDelete( action: 'cascade');
    $table->timestamps();
});
```

Fig. 45: Migración categoría-usuario

Luego, añadimos la importante función para ser accedida desde la ruta mediante el controlador que representa en Laravel como una relación de muchos a muchos totalmente rota que debe tener una asociación como se muestra en la **Fig. 46**.

```php
public function categoriesByUser()
{
    $user = Auth::user();
    return new CategoryCollection($user->categories);
}
```

Fig. 46: Relación categoría-usuario

Para concluir con la relación, se espera que desde la ruta pueda ejercer ese control obteniendo los datos correctos por medio de las especificaciones en las peticiones al servidor como se indica en la **Fig. 47**.

```php
Route::get( uri: '/user/categories', [CategoryController::class, 'categoriesByUser']);
```

Fig. 47: Ruta relacional

Codificación para políticas del sistema

Las políticas son muy bien ejecutadas que se presentan ante rutas en riesgo. Entonces, el *backend* sostiene la seguridad, integridad de los datos a través de privilegios; es decir, cuando el funcionamiento se vea amenazado es posible que no deba mostrarse. Por lo tanto, al existir roles que puedan dar los permisos oportunos mantienen el inicio de sesión correspondiente. En la **Fig. 48** se presenta una de las mayores herramientas ejercidas por Laravel, siendo convocado el *middleware* que actúa como filtro de evaluación de un inicio de sesión para la petición de datos [26].

```php
//Private routes
Route::group(['middleware' => ['jwt.verify']], function() {
```

Fig. 48: Herramienta *middleware*

Para la respectiva creación de las políticas, Laravel proporciona comandos sencillos para su implementación lo cual funcionan por medio de funciones que identifican las sesiones de los usuarios validando con los permisos que obtienen cada uno. A continuación, se presenta un ejemplo del modelo mascotas que prioriza la acción de acuerdo con el controlador; este ejerce un inicio de sesión de usuario adoptante para crear o actualizar mascotas en el sistema como se visualiza en la **Fig. 49**.

```php
public function create(User $user)
{
    return $user->isGranted( role: User::ROLE_ADOPTER);
}

/**
 * Determine whether the user can update the model.
 *
 * @param  \App\Models\User  $user
 * @param  \App\Models\Form  $form
 * @return mixed
 */
public function update(User $user, Form $form)
{
    return $user->isGranted( role: User::ROLE_ADOPTER) && $user->id === $form->user_id;
}
```

Fig. 49: Políticas mascotas

En el controlador, las funciones crear y actualizar deben presentar la referencia a la política planteada para que de tal manera se pueda realizar la petición desde el usuario activo como se muestra en la **Fig. 50**.

```php
$this->authorize( ability: 'create',  arguments: Form::class);
$this->authorize( ability: 'update',  arguments: Form::class);
```

Fig. 50: Enlace a políticas mascotas

Sprint 5. Despliegue en Heroku y pruebas de aceptación

Para concluir con el *Sprint Backlog*, se procede a realizar las tareas del *Sprint* 5. En este apartado ya se empieza a construir el sistema en un servidor público lo cual hacemos uso de la plataforma de Heroku para desplegar y poder probar el funcionamiento. Los resultados abarcados en el último *Sprint* son:

- Despliegue en Heroku por variables públicas con proyección en migraciones y *seeders* en Heroku.
- Realización pruebas de aceptación.

Despliegue en Heroku por variables públicas con proyección en migraciones y *seeders* en Heroku

Inicialmente, la configuración en los archivos del proyecto es fundamental añadir un nuevo documento *Procfile* que va a permitir la conexión con Heroku desde el sistema llamado

Heroku CLI. No obstante, en el documento se especificará la ruta de la construcción pública del sistema; usando como servidor web de por medio Apache para el manejo de las peticiones como se describe en la **Fig. 51**.

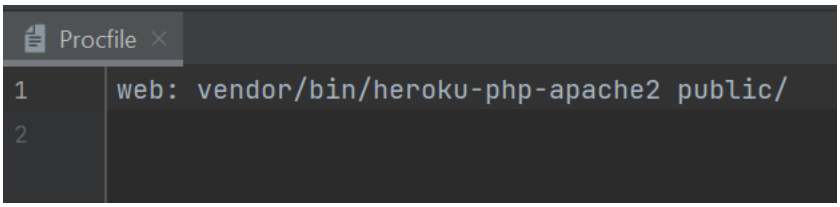

Fig. 51: Documento *Procfile*

Para la conexión con Heroku, es específicamente desde la raíz de GitHub el cual se va a conectar y desplegar. Entonces, se ha dado acceso al repositorio para que pueda presentar ya de forma pública; obviamente, con los últimos cambios realizados desde la rama principal. Nombre de dicho proyecto representa al sistema que es poli-huellas, permite ya la creación de la aplicación con su sistema gestor de base de datos incorporado respectivamente como se visualiza en la **Fig. 52**.

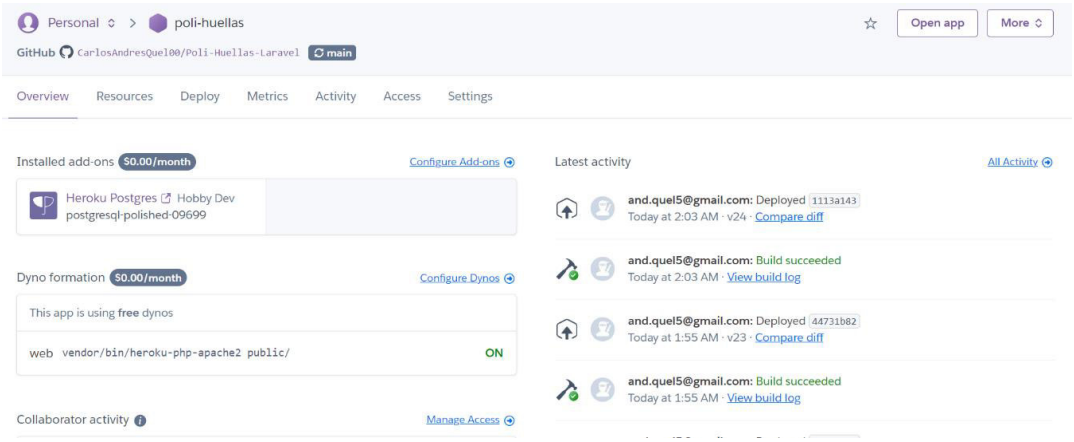

Fig. 52: Despliegue aplicación Heroku

En el repositorio no es posible subir las variables de entorno con las que se va a trabajar de manera pública en el proyecto. Por tal motivo, se procede de forma manual o desde líneas de comandos a agregar cada una de las variables públicas con las que se desarrolla el proyecto como se muestra en la **Fig. 53**; representan tanto a la base de datos que se está usando como los servidores y almacenamiento de imágenes desde una fuente externas. Entonces, para la base de datos hacemos uso del sistema gestor PostgreSQL; permitiendo almacenar todos los datos correspondientes lo cual tiene una licencia libre con limitación de ingreso de información, y ofreciendo las variables al ser instalado. Heroku como el servidor web donde funciona la aplicación externamente, permitiendo tener un

dominio público también específicamente para consumir la información por parte del cliente. Para el almacenamiento de imágenes, ya fue muy importante hacer uso de otra plataforma externa Cloudinary que ofrece la libertad de almacenar galería del sistema por el hecho del respectivo formato [27].

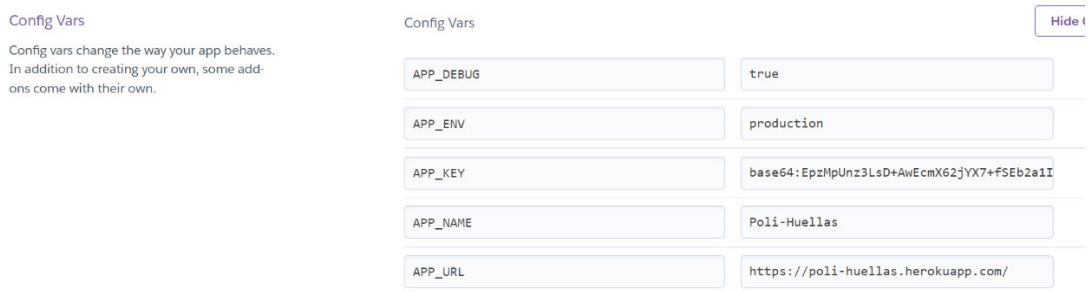

Fig. 53: Heroku variables de entorno

Por último, ya en el despliegue de la aplicación se compila las migraciones y *seeders* en producción desde la consola de la plataforma de Heroku con el objetivo de crear las tablas y los datos ficticios en la base de datos como se muestra en la **Fig. 54**.

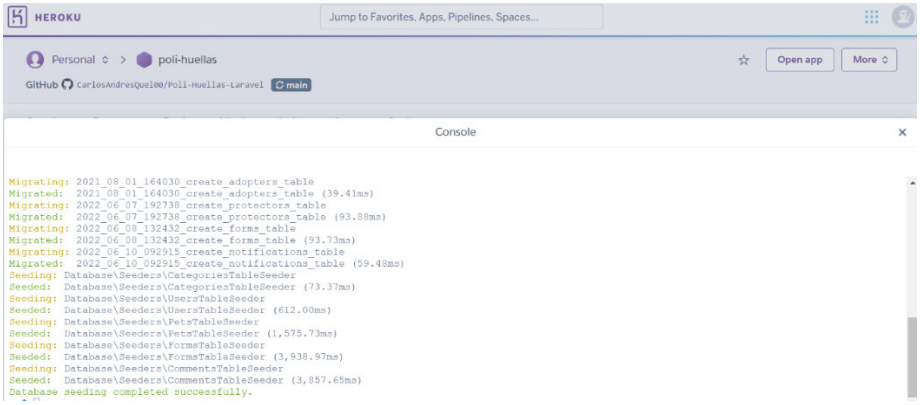

Fig. 54: Heroku migraciones y *seeders*

Realización pruebas de aceptación

En esta sección, se permite la implementación del funcionamiento del proyecto para verificar que este ejerciendo resultados positivos con respecto a pruebas realizadas desde Postman. El cliente puede usar el sistema de acuerdo con los requerimientos planteados que beneficiaron el desarrollo del proyecto, principalmente la autenticación que ha permitido mostrar el ejemplo inicial de la prueba de aceptación ya en producción para iniciar sesión. Las pruebas de aceptación completas se presentan en el **ANEXO II** Manual técnico.

La autenticación es el inicio de sesión que se proporciona a cualquier usuario a ingresar al sistema en el cual debe indicar su correo electrónico como base para este proyecto junto

con la contraseña con la cual se registró en el sistema, y fue almacenado en la base de datos con respuesta de los demás datos almacenados correctamente como se visualiza en la **Fig. 55**.

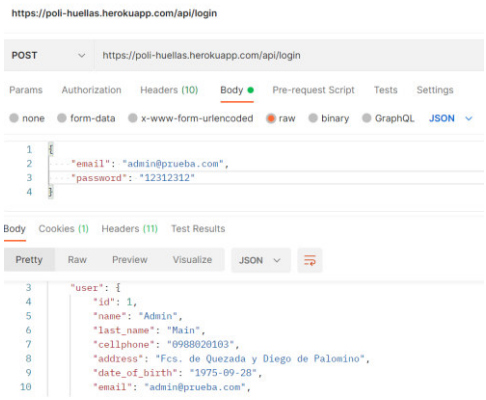

Fig. 55: Postman iniciar sesión

Por el contrario, si la autenticación no es correcta; también presenta mensajes de error lo cual permite al cliente identificar y proyectar el error para que el usuario pueda corregir, así tener un inicio de sesión con credenciales funcionales. Del ejemplo anterior, se intenta iniciar sesión con credenciales incorrectas lo cual muestra el mensaje correspondiente como se visualiza en la **Fig. 56**.

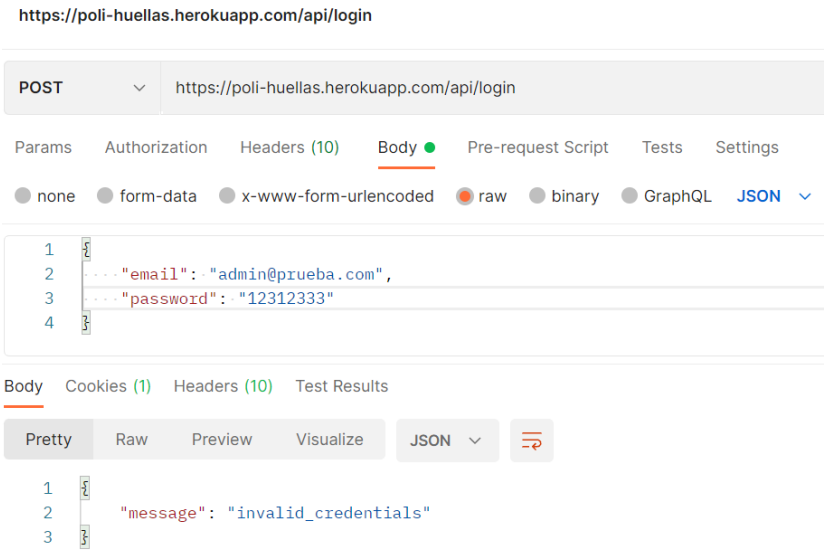

Fig. 56: Postman validación credenciales

En la siguiente **TABLA IV** se visualiza la descripción de la prueba de aceptación respectivamente al inicio de sesión ejecutada por los usuarios. La descripción de las demás pruebas de aceptación se presenta en el **ANEXO II** Manual técnico.

TABLA IV Prueba de Aceptación PA002

Prueba de Aceptación	
Identificador: PA002	**Identificador de Historia de Usuario:** HU002
Título de la prueba de aceptación: Iniciar sesión	
Descripción: El usuario por rol podrá iniciar sesión por medio del formulario corto de campos obligatorios de llenar para acceder al sistema. Añadiendo, el usuario puede autenticarse desde los servicios de Google.	
Entrada/Pasos de ejecución: Ingresar a la URL de la aplicación web Completar los campos: - Correo electrónico - Contraseña Seleccionar botón iniciar sesión	
Resultado deseado: El *backend* nuevamente evalúa los datos enviados sean correctamente colocados, y los procesa verificando que en la base de datos exista el usuario. Los servicios de Google proporcionan los datos, de tal manera pueden ser almacenados también.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba información correcta lo cual permite iniciar sesión. Acepta uso del cliente.	

4 CONCLUSIONES

- El funcionamiento del proyecto en cada una de sus etapas se ha logrado con éxito, así manteniendo los requerimientos para el sistema. Durante cada etapa se han presentado resultados buenos con el objetivo de ir formando y uniendo los cambios desarrollados al *API REST*.
- La metodología me fue de mucha utilidad porque hubo cambios casi al final del proyecto que se lograron solucionar e implementar, añadir al sistema de manera inmediata. Una metodología bastante ágil que la adaptación a los cambios fue de gran alcance, interfiriendo en destrucciones del propio documento como tal.
- Laravel ha sido una de las mayores experiencias útiles a nivel de *backend* que es en donde se produce este proyecto. El uso y desarrollo fue excelente con un gran manejo accesible a todas las herramientas personalizadas por parte del código, muy interactivo, facilitando las implementaciones concretas.
- En la parte de las notificaciones falto concretar y profundizar un poco más, pero funciona acorde al alcance proporcionado por el sistema.
- Finalmente, la construcción para poner en producción esta parte del sistema fue correcta. Es decir, el despliegue se lo realizo en la plataforma de Heroku con todas las consideraciones previstas en el desarrollo con fácil acceso. Las actualizaciones del proyecto estarán en constante relación con el despliegue; debido a que, la estructura entre el repositorio del código y la plataforma enlazan un mantenimiento automático.

5 RECOMENDACIONES

- Una de las mayores recomendaciones si es analizar cómo, con qué y dónde se van a hacer los formularios. El hecho de usar formularios exige gran variedad de la información por lo que manejarlo desde una base de datos es pesado y delicado por lo que pensarlo bien ayudaría mucho a un mejor desarrollo del sistema.
- Usar una buena metodología facilitaría y ahorraría tiempo, recursos del proyecto en camino. Uno de los factores importantes es mantener el contacto con el cliente y el producto para adaptar cambios necesarios.
- Fundamentalmente, la investigación previa y la asociación a las herramientas en uso desde sus fuentes precisas sirve de mucho para corregir o manipular el código fuente de un proyecto.
- La realización de una *API REST* desde el inicio es una buena forma de aprender el funcionamiento en segundo plano de un sistema distribuido, y también de seguir una correcta ejecución con un control seguro.
- A nivel de pruebas de aceptación, es recomendable ir probando el sistema para verificar o convalidar los errores que se van produciendo y poder solucionarlos a tiempo con investigaciones continuas.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] D. S. Ramírez, E. d. l. M. Zurita y F. J. Galora, «Analizando Internet de las Cosas y la nube informática,» 10 Febrero 2022. [En línea]. Available: https://revista.uisrael.edu.ec/index.php/ro/article/view/535/544. [Último acceso: 21 Mayo 2022].

[2] E. Robles, «Cultura y Era Tecnológica,» RAZÓN Y PALABRA, Octubre - Noviembre 2003. [En línea]. Available: http://www.razonypalabra.org.mx/anteriores/n35/erobles.html. [Último acceso: 21 Mayo 2022].

[3] A. López, «Estudio comparativo sobre la tenencia de mascotas entre sectores rurales y urbanos del DMQ,» Quito, 2020.

[4] J. A. Millán, «Breve Historia de la Internet,» 21 Enero 2006. [En línea]. Available: http://cv.udl.cat/cursos/elsmitjans/t1/docs/internet2.pdf. [Último acceso: 21 Mayo 2022].

[5] H. Montero y M. Fernández, «La Experiencia del Usuario,» nsu, 7 Septiembre 2005. [En línea]. Available: https://www.nosolousabilidad.com/articulos/experiencia_del_usuario.htm. [Último acceso: 21 Mayo 2022].

[6] E. Fritz, G. A. Montejano, P. Garcia y S. G. Bast, «Integridad de datos en sistemas de gestión de aprendizaje,» 2 Junio 2015. [En línea]. Available: http://sedici.unlp.edu.ar/handle/10915/46010. [Último acceso: 13 Febrero 2022].

[7] G. Velásquez, «METODOLOGÍA DE DESARROLLO DE SOFTWARE,» Universidad Católica de los Ángeles Chimbote, 19 Diciembre 2017. [En línea]. Available: https://www.uladech.edu.pe/images/stories/universidad/documentos/2018/metodologia-desarrollo-software-v001.pdf. [Último acceso: 21 Mayo 2022].

[8] F. Imbernón, «1918-2018. Cien años de la metodología de proyectos,» 4 Abril 2018. [En línea]. Available: http://hdl.handle.net/2445/166033. [Último acceso: 21 Mayo 2022].

[9] M. Mejia, «¿Qué es el Backend y cómo usarlo?,» Crehana, 16 Febrero 2021. [En línea]. Available: https://www.crehana.com/ec/blog/desarrollo-web/que-es-el-backend-y-como-usarlo/. [Último acceso: 21 Mayo 2022].

[10] I. García, «Definición e implementación de una API REST para sistemas de recomendación,» UAM. Departamento de Ingeniería Informática, Julio 2018. [En línea]. Available: http://hdl.handle.net/10486/688292. [Último acceso: 22 Mayo 2022].

[11] C. Álvarez y J. L. Maroto, «La elección del estudio de caso en investigación educativa,» Gazeta de Antropología, Junio 2012. [En línea]. Available: http://hdl.handle.net/10481/20644. [Último acceso: 22 Mayo 2022].

[12] J. F. Pareja Quinaluisa, «Evaluación de procesos de software utilizando EvalProSoft Aplicado a un caso de estudio,» 08 02 2012. [En línea]. Available: http://bibdigital.epn.edu.ec/handle/15000/4491 .

[13] K. Schwaber y J. Sutherland, «Las Reglas del Juego,» La Guía de Scrum, Julio 2013. [En línea]. Available: https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf. [Último acceso: 22 Mayo 2022].

[14] C. A. Guerra, «Obtención de Requerimientos. Técnicas y Estrategia,» SG, [En línea]. Available: https://sg.com.mx/revista/17/obtencion-requerimientos-tecnicas-y-estrategia. [Último acceso: 23 Mayo 2022].

[15] A. Álvarez, «Historias de Usuario: qué son, reglas y consejos.,» netmind, 06 Julio 2020. [En línea]. Available: https://netmind.net/es/historias-de-usuario-reglas/. [Último acceso: 23 Mayo 2022].

[16] T. Otwell, «Laravel 8,» [En línea]. Available: https://laravel.com/docs/8.x. [Último acceso: 23 Mayo 2022].

[17] E. Niño, «Modelo cliente servidor,» 09 Marzo 2012. [En línea]. Available: http://hdl.handle.net/10584/2205. [Último acceso: 24 Mayo 2022].

[18] J. M. Aguilar, «¿Qué es el patrón MVC en programación y por qué es útil?,» campusMVP, 15 Octubre 2019. [En línea]. Available: https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx. [Último acceso: 24 Mayo 2022].

[19] «Windows Preguntas frecuentes,» Apache Friends, [En línea]. Available: https://www.apachefriends.org/es/faq_windows.html. [Último acceso: 24 Mayo 2022].

[20] «PostgreSQL 14.1 Documentation,» PostgreSQL, 11 Noviembre 2021. [En línea]. Available: https://www.postgresql.org/docs/current/index.html. [Último acceso: 24 Mayo 2022].

[21] «Create API Documentation with Postman,» API PLATFORM, [En línea]. Available: https://learning.postman.com/docs/getting-started/introduction/. [Último acceso: 24 Mayo 2022].

[22] «Getting Started on Heroku with PHP,» Heroku Dev Center, [En línea]. Available: https://devcenter.heroku.com/articles/getting-started-with-php. [Último acceso: 24 Mayo 2022].

[23] Y. Fernández, «Qué es Github y qué es lo que le ofrece a los desarrolladores,» xataka, 30 Octubre 2019. [En línea]. Available: https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores. [Último acceso: 24 Mayo 2022].

[24] «Cómo empezar,» Jetbrains, 03 Agosto 2021. [En línea]. Available: https://www.jetbrains.com/es-es/phpstorm/documentation/. [Último acceso: 06 Junio 2022].

[25] «Introduction to JSON Web Tokens,» Auth0, [En línea]. Available: https://jwt.io/introduction/. [Último acceso: 13 Junio 2022].

[26] D. Bakken, «Middleware,» 04 Enero 2003. [En línea]. Available: https://webhost.laas.fr/TSF/IFIPWG/Workshops&Meetings/43/01-Bakken.pdf. [Último acceso: 02 Julio 2022].

[27] «Developer get started guide,» Cloudinary, 2012. [En línea]. Available: https://cloudinary.com/documentation/how_to_integrate_cloudinary. [Último acceso: 02 Julio 2022].

7 ANEXOS

ANEXO I. Certificado Turnitin

ANEXO II. Manual técnico

ANEXO III. Manual de usuario

ANEXO IV. Manual de instalación

ANEXO I Certificado Turnitin

ESCUELA POLITÉCNICA NACIONAL
ESCUELA DE FORMACIÓN DE TECNÓLOGOS
CAMPUS POLITÉCNICO "ING. JOSÉ RUBÉN ORELLANA"

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 11 de septiembre de 2022

De mi consideración:

Yo, Richard Paúl Rivera Guevara, en calidad de Director del Trabajo de Integración Curricular titulado DESARROLLO DEL BACKEND asociado al DESARROLLO DE UN SISTEMA WEB Y UNA APLICACIÓN MÓVIL PARA GESTIONAR LA ADOPCIÓN DE MASCOTAS EN LA CIUDAD DE QUITO elaborado por el estudiante CARLOS ANDRES QUEL REDROBÁN de la carrera en TECNOLOGÍA SUPERIOR EN DESARROLLO DE SOFTWARE, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 9%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el informe generado por la herramienta Turnitin.

Atentamente,

RICHARD PAUL RIVERA GUEVARA

Firmado digitalmente por RICHARD PAUL RIVERA GUEVARA
Fecha: 2022.09.11 22:03:35 -05'00'

Richard Rivera
Profesor Ocasional a Tiempo Completo
ESFOT

ANEXO II Manual técnico

A continuación, se describen con mayor entendimiento todos los componentes faltantes en el informe técnico del proyecto.

1. RECOPILACIÓN DE REQUERIMIENTOS

En la **TABLA V** se muestra la recopilación de requerimientos completo del proyecto.

TABLA V Recopilación de Requerimientos

RECOPILACIÓN DE REQUERIMIENTOS	
ID - RR	**ENUNCIADO DEL ÍTEM**
RR001	Como *API*, necesito exponer la ruta de registro.
RR002	Como *API*, necesito exponer la ruta de inicio sesión.
RR003	Como *API*, necesito exponer la ruta de los usuarios.
RR004	Como *API*, necesito exponer la ruta de actualización de perfil.
RR005	Como *API*, necesito exponer la ruta de eliminación en cascada de los usuarios.
RR006	Como *API*, necesito exponer ruta de almacenamiento y actualización de una mascota.
RR007	Como *API*, necesito exponer ruta de visualización de las mascotas.
RR008	Como *API*, necesito exponer la ruta de eliminación en cascada de las mascotas.
RR009	Como *API*, necesito exponer la ruta de almacenamiento y actualización de comentarios.
RR010	Como *API*, necesito exponer rutas de visualización y eliminación de los comentarios.
RR011	Como *API*, necesito exponer rutas de almacenamiento, actualización y eliminación de formularios.
RR012	Como *API*, necesito exponer la ruta de visualización del formulario de adopción.
RR013	Como *API*, necesito exponer la ruta de notificaciones.

2. HISTORIAS DE USUARIO

En las tablas consiguientes de la **TABLA VI** hasta la **TABLA XVII** se muestran la lista de historias de usuario completas del proyecto.

TABLA VI Historia de Usuario HU002

HISTORIA DE USUARIO	
Identificador (ID): HU002	**Usuario:** Administrador, Protector y Adoptante
Nombre Historia: Iniciar sesión	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 3	

Responsable: Carlos Quel
Descripción: El administrador, protector y adoptante deberán permitir la funcionalidad de inicio de sesión normal o con Google en el *backend*, para que los usuarios puedan iniciar sesión en el sistema.
Observación: El inicio de sesión deberá contener los siguientes campos: email, contraseña.

TABLA VII Historia de Usuario HU003

	HISTORIA DE USUARIO
Identificador (ID): HU003	**Usuario:** Administrador y Protector
Nombre Historia: Mostrar usuarios	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El administrador deberá tener la funcionalidad de visualizar a los potenciales usuarios finales desde el *backend*.	
Observación: Los potenciales usuarios deben mostrarse por sus campos: nombre, email, dirección, fecha de nacimiento, estado, ocupación.	

TABLA VIII Historia de Usuario HU004

	HISTORIA DE USUARIO
Identificador (ID): HU004	**Usuario:** Protector y Adoptante
Nombre Historia: Editar perfil	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Iteración Asignada: 3	
Responsable: Carlos Quel	
Descripción: El protector y adoptante deberán ofrecer la funcionalidad de actualización de los campos correspondientes a sus datos personales.	
Observación: Las actualizaciones del perfil del usuario deben ser realizadas en cualquier momento y sin restricción por los mismos dueños del perfil.	

TABLA IX Historia de Usuario HU005

	HISTORIA DE USUARIO
Identificador (ID): HU005	**Usuario:** Administrador
Nombre Historia: Eliminar usuarios	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	

Responsable: Carlos Quel
Descripción: El administrador deberá tener la funcionalidad de eliminar a los usuarios registrados en el *backend*, que el usuario final este intentando realizar un procedimiento ilegal.
Observación: El administrador debe contar con el privilegio de poder eliminar a los usuarios mal intencionados que quieran dañar el sistema.

TABLA X Historia de Usuario HU006

HISTORIA DE USUARIO	
Identificador (ID): HU006	**Usuario:** Protector
Nombre Historia: Crear y editar mascotas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El protector deberá realizar por parte del *backend* la funcionalidad de almacenar y modificar mascotas propias en el sistema.	
Observación: La mascota debe ser especificada en el campo estado: sin adopción o con adopción.	

TABLA XI Historia de Usuario HU007

HISTORIA DE USUARIO	
Identificador (ID): HU007	**Usuario:** Administrador y Adoptante
Nombre Historia: Mostrar mascotas	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El adoptante debe proporcionar en el *backend* la funcionalidad de visualizar las características de las mascotas almacenadas por parte del *backend*.	
Observación: Las mascotas deben mostrarse por sus campos: imagen, nombre, descripción, fecha de nacimiento, vacunado, tipo de mascota, tamaño, sexo.	

TABLA XII Historia de Usuario HU008

HISTORIA DE USUARIO	
Identificador (ID): HU008	**Usuario:** Administrador
Nombre Historia: Eliminar mascotas	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	

Responsable: Carlos Quel
Descripción: El administrador deberá tener la funcionalidad de eliminar a las mascotas almacenadas en el *backend*, que el usuario final este publicando contenido no exclusivo.
Observación: El administrador debe contar con el privilegio de poder eliminar a las mascotas que no sean del reino animal, y que en su campo foto sea exclusivamente un animal.

TABLA XIII Historia de Usuario HU009

HISTORIA DE USUARIO	
Identificador (ID): HU009	**Usuario:** Administrador, Protector y Adoptante
Nombre Historia: Crear y editar comentarios	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El administrador, protector y adoptante deberán realizar por parte del *backend* la funcionalidad de almacenar y modificar comentarios propios en el sistema.	
Observación: Los comentarios deben ser especificados por los campos: correo, mensaje, fecha de publicación.	

TABLA XIV Historia de Usuario HU010

HISTORIA DE USUARIO	
Identificador (ID): HU010	**Usuario:** Administrador, Protector y Adoptante
Nombre Historia: Mostrar y eliminar comentarios	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El administrador, protector y adoptante debe efectuar la funcionalidad de visualizar y eliminar los comentarios que se vayan haciendo en el transcurso de la interacción con el sistema.	
Observación: Cada comentario debe mostrarse con la identificación del usuario que posteo, ya sea con el campo de nombre o de correo. De igual manera, cada usuario puede eliminar su comentario si así lo desea; si ya es motivos de amenazas, el administrador puede eliminarlos.	

TABLA XV Historia de Usuario HU011

HISTORIA DE USUARIO	
Identificador (ID): HU011	**Usuario:** Administrador y Adoptante
Nombre Historia: Crear, editar y eliminar formularios	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alta
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El administrador deberá crear, editar y eliminar un apartado de formularios donde van a ser llenados por los usuarios adoptantes interesados en las mascotas.	
Observación: Los formularios obligatoriamente deben ser llenados por el campo: teléfono o celular del usuario adoptante. Si se necesita algún campo extra debe poder hacerlo. Incluso, si existen formularios con mal interés pueden ser eliminados.	

TABLA XVI Historia de Usuario HU012

HISTORIA DE USUARIO	
Identificador (ID): HU012	**Usuario:** Administrador y Protector
Nombre Historia: Mostrar formularios de adopción	
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El adoptante deberá ofrecer la funcionalidad de visualización del formulario completo correspondiente a la mascota de interés que quiere adoptar.	
Observación: La visualización del formulario de adopción debe ser únicamente revisada por el administrador y el adoptante del sistema.	

TABLA XVII Historia de Usuario HU013

HISTORIA DE USUARIO	
Identificador (ID): HU013	**Usuario:** Protector y Adoptante
Nombre Historia: Mostrar notificaciones	
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Media
Iteración Asignada: 2	
Responsable: Carlos Quel	
Descripción: El protector deberá ofrecer la funcionalidad de visualizar o recibir las notificaciones correspondientes a los formularios llenados por los potenciales usuarios adoptantes interesados en las mascotas.	

Observación: Las notificaciones deben ser inmediatas, tratando en lo más posible una recepción rápida.

3. PRODUCT BACKLOG

En la **TABLA XVIII** se muestra el *Product Backlog* completo del proyecto.

TABLA XVIII *Product Backlog*

ELABORACIÓN DEL *PRODUCT BACKLOG*				
ID –HU	**HISTORIA DE USUARIO**	**ITERACIÓN**	**ESTADO**	**PRIORIDAD**
HU001	Registrar usuario	1	Finalizado	Alta
HU002	Iniciar sesión	1	Finalizado	Alta
HU003	Mostrar usuarios	2	Finalizado	Media
HU004	Editar perfil	2	Finalizado	Alta
HU005	Eliminar usuarios	2	Finalizado	Media
HU006	Crear y editar mascotas	3	Finalizado	Alta
HU007	Mostrar mascotas	3	Finalizado	Alta
HU008	Eliminar mascotas	3	Finalizado	Media
HU009	Crear y editar comentarios	4	Finalizado	Media
HU010	Mostrar y eliminar comentarios	4	Finalizado	Media
HU011	Crear, editar y eliminar formularios	4	Finalizado	Alta
HU012	Mostrar formularios de adopción	4	Finalizado	Media
HU013	Mostrar notificaciones	4	Finalizado	Alta

4. SPRINT BACKLOG

En la **TABLA XIX** se muestra el detalle completo del *Sprint Backlog* del proyecto.

TABLA XIX *Sprint Backlog*

ELABORACIÓN DEL *SPRINT BACKLOG*						
ID-SB	**NOMBRE**	**MÓDULO**	**ID-HU**	**HISTORIA DE USUARIO**	**TAREAS**	**TIEMPO ESTIMADO**
SB00	Acondicionamiento del ambiente de trabajo				• Realizar la recopilación de requerimientos. • Analizar los requerimientos obtenidos. • Realizar las historias de usuario basándose en los requerimientos. • Instalación y actualización de herramientas para el desarrollo del proyecto. • Realización del modelo entidad relación.	50 H
SB01	REGISTRAR USUARIOS		HU001	Registrar usuario	• Elaboración de migración y modelo de Usuario por rol protector o adoptante. • Elaboración de *seeder* para modelo de Usuario por rol protector o adoptante. • Implementación función de registro en controlador para modelo de Usuario por rol protector o adoptante. • Validación de campos para registro de usuario por rol protector o adoptante. • Elaboración de ruta de registro de usuario por rol protector o adoptante. • Elaboración de funciones para visualización de datos del modelo de Usuario por rol protector o adoptante.	60 H

					• Elaboración de rutas para la visualización de datos del modelo de Usuario por rol protector o adoptante.	
	AUTENTICAR USUARIOS		HU002	Iniciar sesión	• Elaboración de función de autenticación por rol en controlador para modelo de Usuario. • Validación de campos para la autenticación por rol de usuarios. • Elaboración de ruta de inicio de sesión de usuarios.	
SB02	Implementación de CRUD del modelo de Usuarios.	MOSTRAR USUARIOS	HU003	Mostrar usuarios	• Elaboración de migraciones para el modelo de Usuarios. • Implementación de función para visualizar potenciales usuarios desde el controlador. • Validación de autenticación para acceder a los potenciales usuarios. • Elaboración de ruta de mostrar potenciales usuarios.	50 H
		EDITAR PERFIL	HU004	Editar perfil	• Validación de nuevo campo en función de actualización del controlador de modelo de Usuario. • Elaboración de ruta para actualizar usuario por rol en el modelo de Usuario.	
		ELIMINAR USUARIOS	HU005	Eliminar usuarios	• Elaboración de migraciones para el modelo de Usuarios. • Implementación de función para eliminar usuarios desde el controlador. • Validación de autenticación para eliminar a los usuarios. • Elaboración de ruta de eliminar usuarios.	
SB03	Implementación de CRUD del modelo	CREAR Y EDITAR MASCOTAS	HU006	Crear y editar mascotas	• Elaboración de migraciones para el modelo de Mascotas. • Elaboración de *seeder* para modelo de Mascotas.	40 H

	de Detalle de Mascotas.				• Implementación de función para la creación de mascotas en el controlador de modelo de Mascotas. • Validación de campos para creación de mascotas en modelo de Mascotas. • Validación de nuevo campo en función de actualización del controlador de modelo de Mascotas. • Elaboración de ruta para creación de mascotas en el modelo de Mascotas.	
		MOSTRAR MASCOTAS	HU007	Mostrar mascotas	• Elaboración de migraciones para el modelo de Mascotas. • Implementación de función para visualizar el detalle de las mascotas desde el controlador. • Validación de autenticación para acceder al detalle de las mascotas. • Elaboración de ruta de mostrar detalle de las mascotas.	
		ELIMINAR MASCOTAS	HU008	Eliminar mascotas	• Elaboración de migraciones para el modelo de Mascotas. • Implementación de función para eliminar mascotas desde el controlador. • Validación de autenticación para eliminar a las mascotas. • Elaboración de ruta de eliminar mascotas.	
SB04	Implementación de CRUD del modelo de Comentarios.	CREAR Y EDITAR COMENTARIOS	HU009	Crear y editar comentarios	• Elaboración de migraciones para el modelo de Comentarios. • Elaboración de *seeder* para modelo de Comentarios. • Implementación de función para la creación de comentarios en el controlador de modelo de Comentarios.	60 H

					• Validación de campos para creación de comentarios en modelo de Comentarios. • Validación de nuevo campo en función de actualización del controlador de modelo de Comentarios. • Elaboración de ruta para creación y actualización de comentarios en el modelo de Comentarios.	
		MOSTRAR Y ELIMINAR COMENTARIOS	HU010	Mostrar y eliminar comentarios	• Elaboración de migraciones para el modelo de Comentarios. • Implementación de función para visualizar y eliminar comentarios desde el controlador. • Validación de autenticación para acceder y eliminar comentarios. • Elaboración de ruta de mostrar y eliminar comentarios.	
	Implementación de CRUD del modelo de Formularios.	CREAR, EDITAR Y ELIMINAR FORMULARIOS	HU011	Crear, editar y eliminar formularios	• Elaboración de migraciones para el modelo de Formularios. • Elaboración de *seeder* para modelo de Formularios. • Implementación de función para la creación de formularios en el controlador de modelo de Formularios. • Validación de campos para creación de formularios en modelo de Formularios. • Elaboración de ruta para creación, actualización y eliminación de formularios en el modelo de Formularios.	
	Implementación de CRUD del modelo de Visualización de Formulario.	MOSTRAR FORMULARIOS DE ADOPCIÓN	HU012	Mostrar formularios de adopción	• Elaboración de migraciones para el modelo de Formularios. • Implementación de función para visualizar formularios desde el controlador. • Validación de autenticación para acceder a los formularios.	

					• Elaboración de ruta de mostrar formularios.	
	Implementación de CRUD del modelo de Notificaciones.	MOSTRAR NOTIFICACIONES	HU013	Mostrar notificaciones	• Elaboración de migraciones para el modelo de Notificaciones. • Implementación de función para visualizar notificaciones desde el controlador. • Validación de autenticación para acceder a las notificaciones. • Elaboración de ruta de mostrar notificaciones.	
	Implementación de relaciones entre modelos de la base de datos.				• Elaboración de migraciones para modelo categoría y usuario. • Implementación de funcionalidades CRUD de los modelos en sus respectivos controladores. • Validación de campos los modelos. • Elaboración de rutas CRUD para los modelos.	
	Implementación de políticas en modelos de la base de datos.				• Implementación de políticas en todos los modelos de la base de datos con sus respectivos permisos. • Implementación de uso de las políticas creadas en los controladores de cada modelo de la base de datos.	
SB05	Pruebas y despliegue.				• Realizar pruebas de aceptación. • Desplegar la *API* en Heroku.	20 H

5. PRUEBAS DE ACEPTACIÓN

En las tablas consiguientes de la **TABLA XX** hasta la **TABLA XXXI** se describen las pruebas de aceptación del sistema.

TABLA XX Prueba de Aceptación PA001

Prueba de Aceptación	
Identificador: PA001	**Identificador de Historia de Usuario:** HU001
Título de la prueba de aceptación: Registrar usuario	
Descripción: El usuario por rol tendrá la posibilidad de registrarse a través del formulario que presenta los datos. Añadiendo, el usuario puede autenticarse desde los servicios de Google.	
Entrada/Pasos de ejecución: Ingresar a la URL de la aplicación web Completar los campos: - Nombre - Apellido - Número de teléfono - Dirección - Imagen - Fecha de nacimiento - Correo electrónico - Contraseña - Ocupación - Biografía corta Seleccionar botón registrarse	
Resultado esperado: El *backend* procesa la información y la analiza; para acorde a los datos pasados sean validados, de tal forma si son correctos puedan ser almacenados. Los servicios de Google proporcionan los datos, de tal manera pueden ser almacenados también.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba datos correctos y almacena en la base de datos. Acepta uso del cliente.	

TABLA XXI Prueba de Aceptación PA003

Prueba de Aceptación	
Identificador: PA003	**Identificador de Historia de Usuario:** HU003
Título de la prueba de aceptación: Mostrar usuarios	
Descripción: El usuario con rol correspondiente al administrador y protector podrán visualizar los usuarios correspondientes a la base de datos.	
Entrada/Pasos de ejecución: Iniciar sesión ya sea administrador o protector Proyectar un JSON y sus respectivos datos: - Nombre - Apellido - Número de teléfono	

- Dirección - Imagen - Fecha de nacimiento - Correo electrónico
Resultado deseado: El *backend* válida que realmente sean los usuarios correctos que están solicitando la información y los autoriza para la muestra de los usuarios.
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba los usuarios y muestra la información. Acepta uso del cliente.

TABLA XXII Prueba de Aceptación PA004

Prueba de Aceptación	
Identificador: PA004	**Identificador de Historia de Usuario:** HU004
Título de la prueba de aceptación: Editar perfil	
Descripción: Cada usuario podrá modificar sus datos con una sesión autenticada.	
Entrada/Pasos de ejecución: Iniciar sesión usuario propio Completar campos que se desee actualizar en formato JSON Transmitir los datos al *API*	
Resultado deseado: El *backend* socializa la sesión y verifica que sea correcta, autenticada; luego evalúa y analiza los datos con los requerimientos para poder almacenarlos.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba la sesión y guarda los cambios en la base de datos. Acepta uso del cliente.	

TABLA XXIII Prueba de Aceptación PA005

Prueba de Aceptación	
Identificador: PA005	**Identificador de Historia de Usuario:** HU005
Título de la prueba de aceptación: Eliminar usuarios	
Descripción: El usuario con rol administrador podrá quitar usuarios que intentan dañar el sistema por medio de identidades extrañas.	
Entrada/Pasos de ejecución: Iniciar sesión del administrador Usar URL de la aplicación web Seleccionar botón eliminar	
Resultado deseado: El *backend* verifica la sesión activa y como aceptador de peticiones lo que realiza es la eliminación del usuario/s del sistema, borrándolos de la base de datos.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba la sesión y procede a eliminar de la base de datos. Acepta uso del cliente.	

TABLA XXIV Prueba de Aceptación PA006

Prueba de Aceptación	
Identificador: PA006	**Identificador de Historia de Usuario:** HU006
Título de la prueba de aceptación: Crear y editar mascotas	
Descripción: Cada usuario con rol protector tendrá la oportunidad de crear y editar sus publicaciones de mascotas siempre y cuando se tenga la sesión activa.	
Entrada/Pasos de ejecución: Iniciar sesión del protector Usar URL de la aplicación web Completar campos que se desee actualizar en formato JSON Llenar campos para crear en formato JSON: - Nombre - Sexo - Tipo - Tamaño - Descripción - Fecha de nacimiento - Imagen Transmitir los datos al *API*	
Resultado deseado: El *backend* evalúa y analiza la sesión autenticada para después verificar que los datos que se estén enviando sean los correctos, y de allí almacenarlos en la base de datos.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba la sesión y los datos para almacenar en la base de datos. Acepta uso del cliente.	

TABLA XXV Prueba de Aceptación PA007

Prueba de Aceptación	
Identificador: PA007	**Identificador de Historia de Usuario:** HU007
Título de la prueba de aceptación: Mostrar mascotas	
Descripción: El usuario con rol correspondiente al administrador y adoptante podrán visualizar las mascotas correspondientes a la base de datos.	
Entrada/Pasos de ejecución: Iniciar sesión ya sea administrador o adoptante Proyectar un JSON y sus respectivos datos: - Nombre - Sexo - Tipo - Tamaño - Descripción - Fecha de nacimiento - Adoptado - Imagen	
Resultado deseado: El *backend* válida que realmente sean los usuarios correctos que están solicitando la información y los autoriza para la muestra de las mascotas.	
Evaluación de prueba: Resultado completado satisfactoriamente.	

El *backend* comprueba los usuarios y muestra la información. Acepta uso del cliente.

TABLA XXVI Prueba de Aceptación PA008

Prueba de Aceptación	
Identificador: PA008	**Identificador de Historia de Usuario:** HU008
Título de la prueba de aceptación: Eliminar mascotas	
Descripción: El usuario con rol administrador podrá quitar mascotas que intentan dañar el sistema por medio de publicaciones extrañas.	
Entrada/Pasos de ejecución: Iniciar sesión del administrador Usar URL de la aplicación web Seleccionar botón eliminar	
Resultado deseado: El *backend* verifica la sesión activa y como aceptador de peticiones lo que realiza es la eliminación de la mascota/s del sistema, borrándolos de la base de datos.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba la sesión y procede a eliminar de la base de datos. Acepta uso del cliente.	

TABLA XXVII Prueba de Aceptación PA009

Prueba de Aceptación	
Identificador: PA009	**Identificador de Historia de Usuario:** HU009
Título de la prueba de aceptación: Crear y editar comentarios	
Descripción: Cada usuario tendrá la oportunidad de crear y editar sus comentarios de las mascotas publicadas siempre y cuando se tenga la sesión activa.	
Entrada/Pasos de ejecución: Iniciar sesión del usuario Usar URL de la aplicación web Completar campos que se desee actualizar en formato JSON Llenar campos para crear en formato JSON: - Texto Transmitir los datos al *API*	
Resultado deseado: El *backend* evalúa y analiza la sesión autenticada para después verificar que los datos que se estén enviando sean los correctos, y de allí almacenarlos en la base de datos.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba la sesión y los datos para almacenar en la base de datos. Acepta uso del cliente.	

TABLA XXVIII Prueba de Aceptación PA010

Prueba de Aceptación

Identificador: PA010	**Identificador de Historia de Usuario:** HU010
Título de la prueba de aceptación: Mostrar y eliminar comentarios	
Descripción: El usuario podrán visualizar y quitar los comentarios propios correspondientes a la base de datos.	
Entrada/Pasos de ejecución: Iniciar sesión del usuario Usar URL de la aplicación web Proyectar un JSON y sus respectivos datos: - Texto Seleccionar botón eliminar	
Resultado deseado: El *backend* válida que realmente sean los usuarios correctos que están solicitando la información y los autoriza para la muestra de los comentarios, y como aceptador de peticiones lo que realiza es la eliminación del comentario/s del sistema, borrándolos de la base de datos.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba los usuarios, muestra la información y procede a eliminar de la base de datos. Acepta uso del cliente.	

TABLA XXIX Prueba de Aceptación PA011

	Prueba de Aceptación
Identificador: PA011	**Identificador de Historia de Usuario:** HU011
Título de la prueba de aceptación: Crear, editar y eliminar formularios	
Descripción: El usuario con rol administrador o adoptante tendrán la oportunidad de crear, editar y quitar los formularios propios correspondientes a la base de datos de las mascotas publicadas siempre y cuando se tenga la sesión activa.	
Entrada/Pasos de ejecución: Iniciar sesión del administrador o adoptante Usar URL de la aplicación web Completar campos que se desee actualizar en formato JSON Llenar campos para crear en formato JSON: - Responsable - Razón - Vivienda - Descripción - Enfermedades - Hijos - Tiempo libre - Viaja - Nuevo - Animales Transmitir los datos al *API* Seleccionar botón eliminar	
Resultado deseado: El *backend* evalúa y analiza la sesión autenticada para después verificar que los datos que se estén enviando sean los correctos, y de allí almacenarlos en la base de datos; como aceptador de peticiones lo que realiza es la eliminación del formulario/s del sistema, borrándolos de la base de datos.	
Evaluación de prueba:	

Resultado completado satisfactoriamente. El *backend* comprueba la sesión, los datos para almacenar en la base de datos y procede a eliminar de la base de datos. Acepta uso del cliente.

TABLA XXX Prueba de Aceptación PA012

	Prueba de Aceptación
Identificador: PA012	**Identificador de Historia de Usuario:** HU012
Título de la prueba de aceptación: Mostrar formularios de adopción	
Descripción: El usuario con rol correspondiente al administrador y protector podrán visualizar los formularios correspondientes a la base de datos.	
Entrada/Pasos de ejecución: Iniciar sesión ya sea administrador o protector Proyectar un JSON y sus respectivos datos: - Responsable - Razón - Vivienda - Descripción - Enfermedades - Hijos - Tiempo libre, etc.	
Resultado deseado: El *backend* válida que realmente sean los usuarios correctos que están solicitando la información y los autoriza para la muestra de los formularios.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba los usuarios y muestra la información. Acepta uso del cliente.	

TABLA XXXI Prueba de Aceptación PA013

	Prueba de Aceptación
Identificador: PA013	**Identificador de Historia de Usuario:** HU013
Título de la prueba de aceptación: Mostrar notificaciones	
Descripción: El usuario con rol correspondiente al protector y adoptante podrán visualizar las notificaciones correspondientes a la base de datos.	
Entrada/Pasos de ejecución: Iniciar sesión del protector o adoptante Proyectar un JSON y sus respectivos datos: - Tipo	
Resultado deseado: El *backend* válida que realmente sean los usuarios correctos que están solicitando la información y los autoriza para la muestra de las notificaciones.	
Evaluación de prueba: Resultado completado satisfactoriamente. El *backend* comprueba los usuarios y muestra la información. Acepta uso del cliente.	

ANEXO III Manual de usuario

El funcionamiento del sistema se demuestra por medio de un video como parte del manual de usuario, y que se encuentra en el siguiente enlace de YouTube: https://youtu.be/sq5hW9cW3qE

ANEXO IV Manual de instalación

En el repositorio de GitHub del siguiente enlace se presenta el manual de como instalar y usar por medio del código fuente expuesto, que describe el completo desarrollo del proyecto: https://github.com/CarlosAndresQuel00/Poli-Huellas-Laravel