ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE APLICACIÓN WEB PARA COMPRAVENTA DE BIENES INMUEBLES
COMPONENTE: DESARROLLO DE UN BACKEND

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

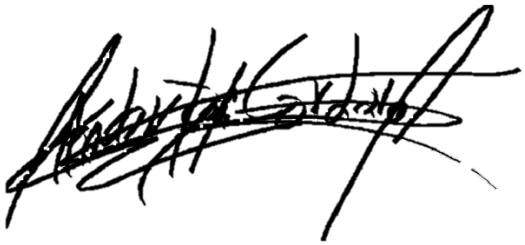
ESTUDIANTE: ANDERSON IVÁN CÓRDOVA CALVOPIÑA

DIRECTOR: ING. JUAN PABLO ZALDUMBIDE PROAÑO

DMQ, septiembre 2022

CERTIFICACIONES

Yo, ANDERSON IVÁN CÓRDOVA CALVOPIÑA 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.



ANDERSON IVÁN CÓRDOVA CALVOPIÑA

anderson.cordova@epn.edu.ec

andersoncordova277@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por ANDERSONIVÁN CÓRDOVA CALVOPIÑA, bajo mi supervisión.

Firmado electrónicamente por:
JUAN PABLO
ZALDUMBIDE
PROANO

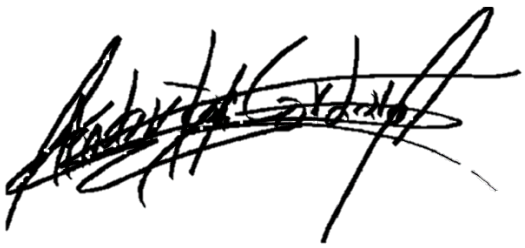
DIRECTOR

juan.zaldumbide@epn.edu.ec

DECLARACIÓN DE AUTORÍA

Yo, ANDERSON IVÁN CÓRDOVA CALVOPIÑA, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Sin perjuicio de los derechos reconocidos en el primer párrafo del artículo 114 del Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación -COESC-, soy titular de la obra en mención y otorgo una licencia gratuita, intransferible y no exclusiva de uso con fines académicos a la Escuela Politécnica Nacional. Entrego toda la información técnica pertinente. En el caso de que hubiese una explotación comercial de la obra por parte de la EPN, se negociará los porcentajes de los beneficios conforme lo establece la normativa nacional vigente.



ANDERSON IVÁN CÓRDOVA CALVOPIÑA

DEDICATORIA

El presente proyecto y el esfuerzo invertido se lo dedico en primera instancia a mi abuelita, quien supo ser pilar para mi crecimiento como ejemplo de perseverancia, honestidad y trabajo duro. Sin su amor, esto no hubiera sido posible.

A mi hermano mayor, Paúl, que es y sigue siendo mi mayor tutor, supo dejar en mi la confianza de mis capacidades, que hoy en día se ven reflejadas en mis proyectos.

A mis padres, por su apoyo incondicional y confianza.

Finalmente, quiero dedicar este proyecto a mis amigos y colegas del vecindario, quienes, en colectivo, han sido un apoyo y fuentes de inspiración como recordatorio de que los objetivos que se propongan en la vida son posibles a pesar de las adversidades.

-Anderson-

AGRADECIMIENTO

Para agradecer se obtiene una lista larga, desde el apoyo de mi hogar con mi abuelita y mi hermano, por su confianza y amor en todos los momentos de mi vida. Sin duda, la familia es lo primero y agradezco a Dios por esta, no tan común, pero maravillosa familia.

A mis padres, les agradezco por darme la confianza, los recursos y el apoyo necesario para mi educación.

A todos los docentes y tutores que tuve desde pequeño que supieron inculcar en mí esa curiosidad para buscar nuevas respuestas. Como adicional, deseo agradecer en esta etapa a Juan Pablo Zaldumbide, por dar su aporte profesional a este proyecto e incentivarme en mi desarrollo profesional.

Finalmente, gracias a todos mi seres queridos y amigos que creyeron en mí.

-Anderson-

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Un desarrollo backend que controle, almacene y gestione la información generada por un sitio Web encargado de la publicación de bienes inmuebles en una base de datos y se conecte con servicios externos para almacenamiento de imágenes y gestor de búsquedas. De manera imperceptible para el usuario final del sitio, facilitando la obtención de información específica y agilizando la respuesta de los recursos.

La metodología usada es *Extreme Programming* (XP) ya que se basa en principios de simplicidad, comunicación, respeto y la realimentación o "*feedback"* por parte del equipo dedesarrollo; todo esto proporciona la flexibilidad y control sobre los requerimientos cambiantes del proyecto a la vez que garantiza un software de calidad **[1]**

El desarrollo backend se realizó a través de la construcción de una *Aplication programming interface* (API) regido bajo un esquema *REST* (*Representational State Transfer*) por lo que se permite la interacción con los recursos del servidor mediante los métodos *HTTP: POST* (crear), *GET* (consultar), *PUT* (editar), *DELETE* (eliminar) **[2]**.

Dicha APIREST está implementada en el *framework de* Laravel y se encarga de la lógica interna de la aplicación, así mismo ofrece escalabilidad y portabilidad al proyecto.

PALABRAS CLAVE: XP, APIREST, Laravel, inmueble, gestor de búsquedas

ABSTRACT

A backend *development that controls, stores and manages the information generated by a website in charge of publishing real estate in a database and connects with external services for image storage and search engine; in a hidden way for the final user of the site, facilitating the obtaining of specific information and speeding up the response of the resources.*

The methodology used is Extreme Programming (XP) since it is based on principles of simplicity, communication, respect and feedback from the development team; all of this provides flexibility and control over changing project requirements while ensuring quality software ***[1]***

The backend development was carried out through the construction of an Application programming interface (API) governed under a REST scheme (Representational State Transfer) so that interaction with the server resources is allowed through the HTTP methods: POST (create), GET (query), PUT (edit), DELETE (delete) ***[2].***

This APIREST is implemented in the Laravel framework and is in charge of the internal logic of the application, as well as offering scalability and portability to the project.

KEYWORDS: XP, APIREST, Laravel, real estate, search manager

1. INTRODUCCIÓN

En un mundo digitalizado en el que el bien más preciado son los datos, surgen las API's para la gestión de aplicaciones, dejando atrás los sistemas aislados y monolíticos. Con ellas, se consigue ahorro de tiempo y recursos al reutilizar infraestructuras, funciones y software existentes en nuestros proyectos **[3].**

Actualmente el negocio de bienes raíces hace uso de tecnologías para poder ofertar sus bienes y servicios, y se genera una pequeña relación de amor-odio con loa agentes inmobiliarios y los sitios Web, ya que se puede establecer como que estos sitios puedan sustituir la labor de agentes, aunque si bien, estos, son los encargados siempre o casi siempre de alimentar el sitio Web proporcionando publicaciones de bienes y servicios **[4]**.

1.1 Planteamiento del problema

Se desea desarrollar un sitio web que beneficie al sector de agentes inmobiliarios, por ello, se plantea un sitio sencillo de publicación y búsqueda de bienes inmuebles, y se requiere de una interfaz entre aplicaciones que gestione las solicitudes hacia los servidores y devuelva los recursos de manera rápida, ágil y segura.

La información de cada publicación de inmueble se debe indexar en un motor de búsqueda que permita realizar consultas establecida por el usuario en base a los parámetros que éste desee.

Finalmente, se requiere almacenar las imágenes del inmueble en un servidor en la nube para poder acceder a ellas de manera pública en cualquier momento.

1.2 Objetivo general

Desarrollar el backend para aplicación web de compraventa de bienes inmuebles.

1.3 Objetivos específicos

- **Objetivo 1:** Determinar los requerimientos de la APIREST a través de reuniones con el desarrollador frontend con un enfoque orientado a comprender las necesidades del sitio Web.
- **Objetivo 2:** Diseñar el modelo de base de datos en función de la información obtenida de las reuniones del objetivo 1.
- **Objetivo 3:** Implementar la APIREST en función al modelo establecido en el objetivo 2.
- **Objetivo 4:** Desplegar la APIREST con servicios en la nube.
- **Objetivo 5:** Realizar pruebas de la APIREST

1.4 Alcance

Para la implementación de la APIREST se plantea el uso del *framework* de Laravel, el cual nos proporciona, una integridad a los datos a través de reglas y pautas inmersas en la base de datos;confidencialidad, haciendo distinciones de rutas públicas y privadas, para el caso de estas últimas, se hará uso del estándar abierto RFC-7579 *JSON Web Token* (JWT) y llevar a cabo la autenticación de los usuarios **[2]**.
También, se va a brindar disponibilidad con el despliegue en una plataforma de servicio en la nube como Heroku, en la que se generan políticas de acceso para gestionar las autorizaciones de los usuarios dentro de la APIREST.

La propuesta proporcionará la lógica interna de una interfaz en la que se beneficia el desarrollador frontend para que pueda diseñar de manera ágil un sitio Web en la que agentes inmobiliarios puedan ofertar sus servicios, al igual que a laspersonas naturales que quieren vender o adquirir un inmueble, ya que, contará con una herramienta de búsqueda y publicación según sus necesidades.

1.1 Marco Teórico

- **API**

Aplication programming interface (API) es una interfaz encargada de la comunicación entre el backend y el frontend. Esta, recibe peticiones de los recursos de un servidor y los administra entregando respuestas en base a políticas de seguridad y configuración extra que pueda ser incluidas sobre el recurso solicitado **[2] [5].**
La empresa "*MarketsandMarkets"* ha dicho en base a un estudio estadístico que las API's estarían manejando más de 5 millones de dólares para el 2023, debido a la importancia que le está dando la industria a utilizar tecnologías en la nube que administren los servicios, y recalca el uso de la tendencia a la inteligencia artificial e internet de las cosas **[6] [7]**.
Cabe mencionar la importancia en esta tendencia a la tecnología como un ambiente favorable a las API's, pues en una industria en la que se priorice el dato, estas nos ofrecen ahorro de tiempo y de recursos mediante librerías, funciones, infraestructuras y plataformas sin la necesidad de diseñarlo desde un principio **[7]**.
Como se dijo en párrafos anteriores una APIREST en el puente de comunicación, mas no corresponde al servidor o a la base de datos, sino que es explícitamente el código que existe como punto de acceso que a la final nos brinda una mayor escalabilidad y portabilidad en los proyectos **[2] [8].**

- **JSON**

JavaScript Object Notation (JSON) son formatos simples para el intercambio de datos basadas en un esquema de clave: valor **[9]**.

Es ligero y ayuda a su portabilidad, ya que en la actualidad cualquier lenguaje es capaz de leerlo. Y permite datos de arreglas anidados, objetos, booleanos, cadenas de texto y números **[2] [9].**

- **JWT**

JSON Web Token (JWT) es un estándar abierto (RFC-7519) que se basa en el formato de JSON para obtener un token el cual sirve como punto de autorización entre aplicaciones o servicios con la garantía de que sean válidos y seguros **[2] [10]**.

Para este proyecto se centró en los tokens firmados. Estos, nos permiten verificar la integridad de los datos contiene, mientras que los tokens encriptados mantienen el anonimato de estas otras partes. Cuando los tokens se firman utilizando pares de claves públicas y privadas **[2] [10].**

Se tiene la estructura definida de un JWT a través de los siguientes parámetros

| *header.payload.signature* |

- *header*: este contiene dos campos: algo y typ. El primero nos indica el algoritmo de firma que se utiliza, mientras que el segundo define el tipo de token, en el caso de nuestro proyecto es JWT **[2] [10]**
- *payload*: puede contener cualquier tipo de propiedad, pero se recomienda mantener el estándar para JWT **[2] [10].**
- *signature*: cumple la función de una firma secreta que solo conozca el servidor encargado de generar JWT, con la finalidad de que el mensaje no se ve alterado en el trascurso, en el caso de los tokens firmados con una clave privada, también puede verificar que el remitente del JWT es el correspondiente a dicho token **[2] [10].**

Los campos de *header* y *payload* se codifican en Base64 antes de enviarse junto la *secret signature* **[10] .**

- **Algolia**

Algolia, es un motor de búsqueda con resultados a tiempo real a través de una API, que permite búsquedas de texto, numérico o por facetas con resultados en menos de 100ms para cualquier parte del mundo **[11]** .

Su uso reduce el tiempo de desarrollo en la implementación de motores de búsqueda.

- Heroku

Heroku es un servicio de plataforma en la nube *Plataform as a Service* (PaaS), el cual nos ofrece una infraestructura lista para el despliegue en distintos lenguajes de programación como: Node, Ruby, Java, Clojure, Scala, Go, Python, PHP **[12] [13].**

Se ha convertido en en uno de los PaaS más empleado por empresas por sus arquitecturas orientadas a servicios que ahorran tiempo en su levantamiento, permitiendo enfocarse en el desarrollo **[12] [13]**.

2. METODOLOGÍA

Las metodologías ágiles involucran al cliente en el proceso y otorgan la capacidad de responder de manera inmediata a los cambios que puedan darse siendo flexible sin que afecten de forma negativa con los objetivos planteados en los proyectos de software **[14].**

A continuación, definimos la metodología ágil en el que está basado este proyecto.

2.1 Metodología de Desarrollo XP

La metodología *Extreme Programming* (XP) se considera ágil, pues permite poner prioridades en el desarrollo de tal modo que se pueda adaptar los procesos al equipo de desarrollo **[15]**. Esta metodología fue formulada por primera vez por Kent Beck en el libro *Extreme Programming Explained: Embrace Change*, publicado en 1999.

En uso de XP ha facilitado la realización del proyecto, con una comunicación clara entre el desarrollador frontend*,* como cliente directo, con el desarrollador backend*;* permitiendo que cada requerimiento se adapte a los con entregables funcionales manteniendo un esquema de solución simple a las necesidades del cliente [16]. Para comprender mejor esta gestión se muestra en la **TABLA I**: Asignación y roles del equipo de trabajo.**TABLA *I*:** Asignación y roles del equipo de trabajo

TABLA I: Asignación y roles del equipo de trabajo

Nombre	Rol
Ronny Cajas	*Customer* (cliente)
Ing. Juan Pablo Zaldumbide	*Tracker* (encargado de seguimiento)
Anderson Córdova	*Programmer* (desarrollador)

La metodología XP, se basa en 5 valores que deben llevarsea cabo por equipos de trabajo para que puedan realizar proyectos con éxito [**15**]. Se detalla a continuación:

- **Comunicación**

Para XP la comunicación es fundamental, por lo que el cliente debe estar dispuesto a participar con el equipo de desarrollo de forma activa con el proyecto, brindando la disponibilidad necesaria para interactuar en todas las fases del desarrollo **[16] [17].**

- **Simplicidad**

XP sigue el principio “mantenlo sencillo”, con el cual se debe procurar hacer un diseño de de la base de datos lo menos complicado posible, y con el menor número de relaciones

[16]. Se recomienda segmentar muy bien las funcionalidades del proyecto para hacerlo más eficaz y preciso **[16] [17]** .

- **Retroalimentación**

En XP las pruebas deben realizar de manera periódica y transversal al ciclo de desarrollo, con el objetivo de asegurar el funcionamiento del código implementado, de ahí, que a lo largo del proyectose hayan efectuado pruebas unitarias que han permitido testear el código de manera individual cada que se agrega alguna funcionalidad, así como también pruebas de aceptación, un tipo de pruebas centradas en las funcionalidades, y basadas en criterios deaceptación del cliente al verificar como responde el sistema y pueda colaborar con una serie de ideas con el fin de solventar problemas o futuros errores **[15] [16] [17] .**

- **Valentía**

Los miembros de XP deben afrontar con valentía de pequeñas funcionalidades en cada iteración, realizando entregas en corto tiempo, además de lidiar con el estrés, deben ser valientes al encontrarse con errores y darlos a conocer al equipo de desarrollo para resolverlos en la brevedad posible **[15] [18].**

- **Respeto**

El respeto en XP se lo manifiesta de diversas formas para que el equipo trabaje de forma eficiente, que van desde el trato en las distintas reuniones, como las modificaciones en el desarrollo que puedan generar algún impacto negativo en el compañero **[15] [18].**

2.2 Análisis y Levantamiento de Requerimientos

Para todo desarrollo de software es necesario abstraer las necesidades de nuestro cliente, diferenciado sus requerimientos funcionales y no funcionales **[19]**.

En esta etapa se ha realizado reuniones con el desarrollador frontend que será el encargado de consumir nuestra APIREST

- **Historias de Usuario**

Las historias de usuario resultan un resumen breve con la descripción de las necesidades del cliente y se utilizancomo herramienta para dar a conocer los requerimientos del sistema a los desarrolladores **[20]**.

En la metodología XP existe constante comunicación con los usuarios involucrados, lo que implica que los clientes se involucren en la creación de estas historias **[17]**.

Para este proyecto se conversó con el desarrollador frontend y se acordaron 10 historias de Historias resumidas en la **TABLA II:** Resumen de Historias de Usuario para la APIREST . También se muestra un ejemplo en

TABLA *III*: Historia de Usuario HU007: Buscador de inmueble, las restantes historias, se encuentran en el Manual Técnico –Sección - Historias de Usuarios (Pág. 3 - 7).

TABLA II: Resumen de Historias de Usuario para la APIREST

Identificador	Nombre
H001	Inicio de sesión
H002	Registro
H003	Información del usuario
H004	Actualizar perfil usuario
H005	Eliminar cuenta usuario
H005	Publicar inmueble
H006	Actualizar inmueble
H007	Buscador de inmueble
H008	Información de inmueble
H009	Lista de inmuebles
H010	Eliminar inmueble

TABLA III: Historia de Usuario HU007: Buscador de inmueble

HISTORIA DE USUARIO	
Identificador (ID): HU007	**Usuario:** Desarrollador frontend
Nombre Historia: Buscar Inmueble	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Iteración Asignada: 3	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la APIREST debe poder filtrar los resultados de las publicaciones de inmuebles a través de una búsqueda en la que el usuario proporcione palabras clave y valores específicos a los campos del inmueble	
Observación: • El resultado debe ser paginado a razón de 10 publicaciones por página • El usuario establece los parámetros de la búsqueda, estos campos no son obligatorios • Para los filtros numéricos el resultado será igual o mayor al que proporcione el usuario Para los filtros a través de palabras claves se utilizan los campos de título y	

descripción

2.3 Diseño de la Arquitectura

La arquitectura de un proyecto de software define la forma en la que se estructura del proyecto a través de los componentes que debemos construir y el modo en el que se deben de juntar y trabajar entre ellos **[21]**.

- **Arquitectura API REST**

REpresentational State Transfer (REST) es una arquitectura dentro del desarrollo web que apoya al estándar HTTP. Menciona como tal por primera vez por Roy Fielding en el año 2000 en su tesis doctoral **[2] [22]** Y este mismo define una RESTFULL cuando se restringe a:

1. **Cliente-servidor:** el cliente no necesita conocer los detalles de implementación los datos que envía al cliente con el que se mantiene un débil acoplamiento **[2].**
2. **Sin estado:** no se mantiene un estado en la comunicación entre el cliente y el servidor **[2].**
3. **Cacheable:** debe admitir una caché de varios niveles, para recuperar un mismo recurso **[2]**.
4. **Interfaz uniforme:** define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única **[2].**
5. **Sistema de capas:** el servidor se puede levantar en un numero de capas, es decir que cada capa contiene componentes que no puede interactuar de manera directa con las capas inmediatamente inferior o superior. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad [2]**.**
6. **Código a demanda:** Permite añadir funcionalidad mediante descarga de scripts o plugins. Esto simplifica al cliente y mejora la extensibilidad.**[2]**.

Las inmueblees en los métodos de HTTP pueden ser:

- **Método seguro:** no modifica datos en el servidor y es cacheado.
- **Método idempotente:** el resultado es independiente del número de invocaciones.

A continuación, se ve en la **TABLA IV:** Propiedades de los métodos HTTP se detallan los métodos de HTTP.

TABLA IV: Propiedades de los métodos HTTP

Método	Seguro	Idempotente	Definición
GET	Si	Si	Obtener
POST	No	No	Crear
PUT	No	Si	Modificar // Actualizar
DELETE	No	Si	Borrar

Con estas consideraciones en la **Fig 1:** Arquitectura API REST. se muestra la arquitectura que guía a este proyecto.

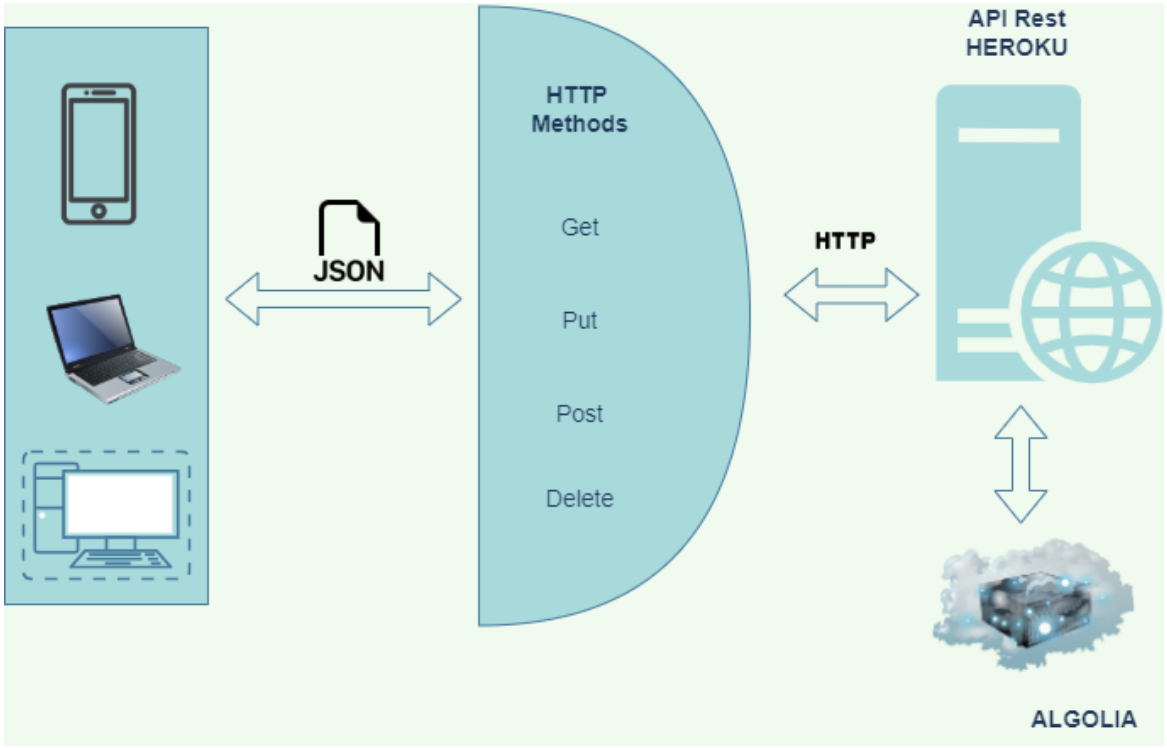


Fig 1: Arquitectura API REST

2.4 Diseño de Base de Datos

Para gestionar los datos que se generen desde el frontend se ha establecido el siguiente modelo entidad relación para definir a nuestra base de datos como vemos en la **Fig 2:** Modelo entidad-relación.

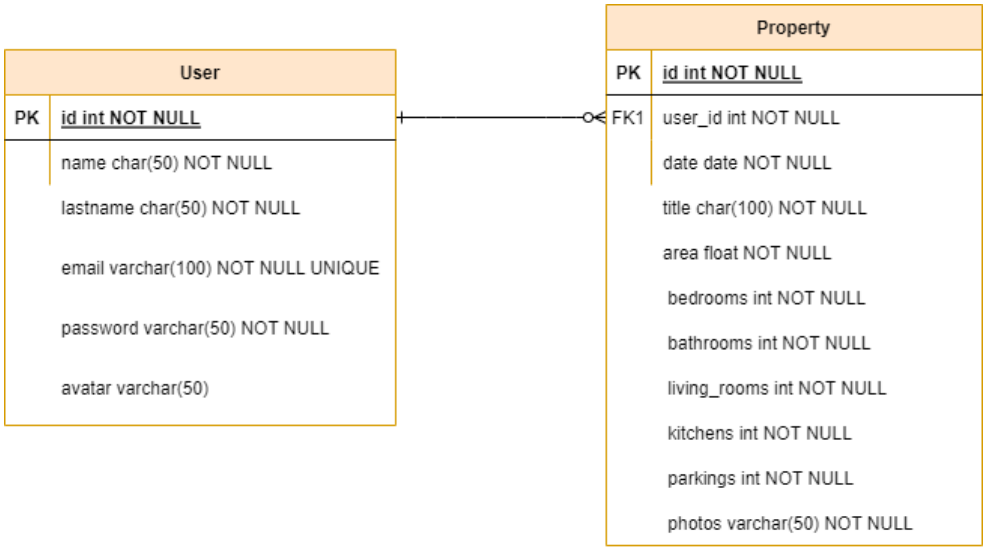


Fig 2: Modelo entidad-relación

2.5 Herramientas de desarrollo

Las herramientas utilizadas para el desarrollo de este proyecto se han resumido en la siguiente **TABLA V**.

TABLA V: Herramientas para el desarrollo de la API REST

Herramienta	Justificación
Laravel	Este *framework* permite el desarrollo ágil y portabilidad del proyecto. A su vez que ahorra tiempo en el desarrollo gracias a sus librerías orientadas a convenciones sobre configuraciones **[23].**
Composer	Permite gestionar las dependencias y librerías de PHP. Además, esta herramienta admite la descarga de componentes o instalar las librerías necesarias para el buen funcionamiento y desarrollo de la APIREST **[24]**.
XAMPP	Permite generar un entorno de desarrollo, para gestionar nuestra base de datos basada en MySQL y configurar cada componente por separado, haciendo que su administración sea sencilla **[25].**

Amazon S3	Servicio de *storage* ofrecido por Amazon para almacenar objetos y recuperar cualquier volumen de datos desde distintas ubicaciones **[26].**

3. RESULTADOS Y DISCUSIÓN

En este apartado se detalla el proceso de creación abarcando desde su análisis de requerimientos, desarrollo y el despliegue a producción de la APIREST con sus respectivas pruebas.

3.1 Iteración 0

Durante la iteración 0 se ha creado el ambiente de desarrollo sobre el cual trabajaremos sobre este proyecto.

Como primer punto se ha descargado del sitio oficial de XAMPP su paquete de software libre para la gestión de nuestra base de datos basada en MySQL de manera local y Composer como gestor de dependencias. Una vez instalados estas herramientas podemos descargar e instalar el *framework* de PHP utilizado para el desarrollo, Laravel, a través del comando '*composer global require laravel/installer',* como se aprecia en la **Fig 3**

```
ander@DESKTOP-VOBAN2E MINGW64 ~/Desktop
$ composer global require laravel/installer
Changed current directory to C:/Users/ander/AppData/Roaming/Composer
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^4.2 for laravel/installer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Info from https://repo.packagist.org: #StandWithUkraine
Updating dependencies
Lock file operations: 0 installs, 1 update, 0 removals
  - Upgrading laravel/installer (v4.2.6 => v4.2.11)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 0 installs, 1 update, 0 removals
  - Downloading laravel/installer (v4.2.11)
  - Upgrading laravel/installer (v4.2.6 => v4.2.11): Extracting archive
Generating autoload files
11 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

Fig 3: Comando para la instalación de Laravel

Utilizaremos el IDE de desarrollo de Visual Studio Code (VSC), en el cual se han instalado extensiones con el lenguaje de PHP para ayudarnos al momento de codificar nuestro proyecto, como se muestra en la **Fig 4.**

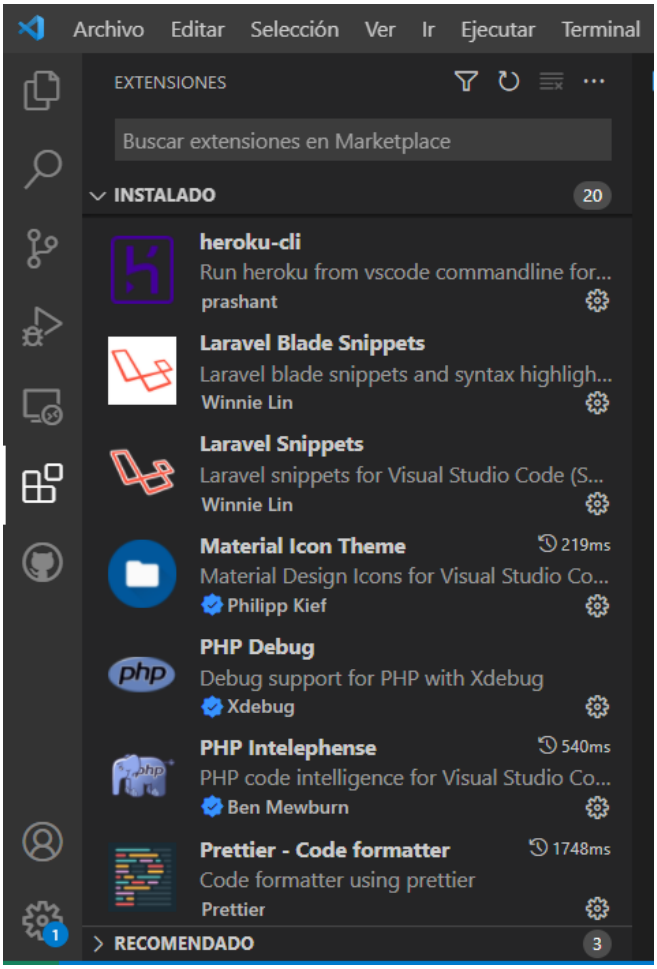


Fig 4: Extensiones en VSC para el desarrollo de la API REST

Una vez generado nuestro ambiente de desarrollo procedemos mediante la línea de comandos, a escribir el comando '*laravel api_feria_inmuebles'* para que se dé la instalación del proyecto en el cual obtendremos un directorio de archivo como se ve en la **Fig 5.** A continuación, se detalla la importancia y uso de las carpetas de nuestro directorio raíz.

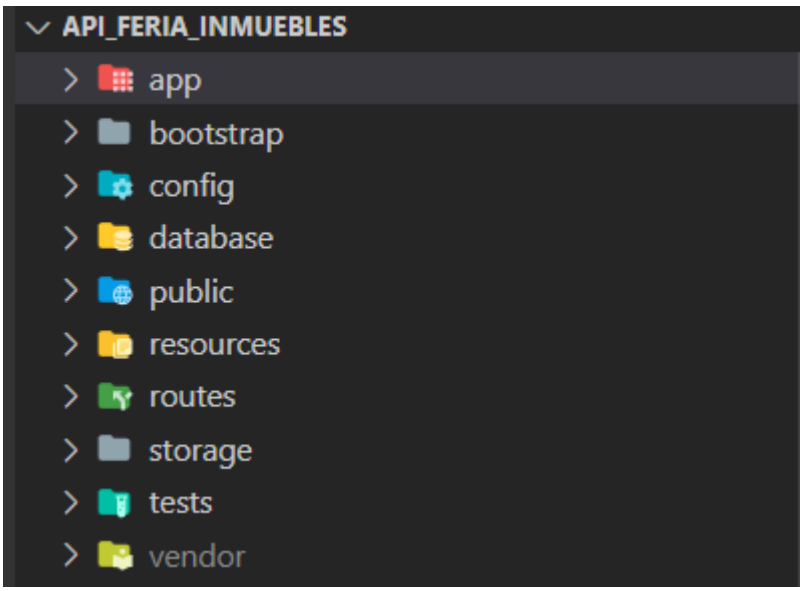


Fig 5: Directorios del proyecto Laravel

- **app:** Este directorio abarca el código principal de la aplicación **[2]**.
- **bootstrap:** Este directorio contiene el archivo de app.php para maquetar el *framework* y archivos generados por el mismo para su optimo rendimiento como los archivos de caché **[2]**.
- **config:** Este directorio contiene los archivos de configuración para toda la aplicación **[2]**.
- **database:** Este directorio de encarga de abarcar la gestión de base de datos desde sus migraciones, modelos, *factories* y *seeders* **[2]**.
- **public:** Este directorio contiene los archivos de acceso público de toda la aplicación a través de su archivo index.php como punto de acceso **[2]**.
- **resources:** Este directorio contiene archivos sin compilar y *templates* de vistas, también puede contener archivos de idiomas o recursos extras para toda la aplicación **[2].**
- **routes:** Este directorio contiene las definiciones de rutas para toda la aplicación A través de los siguientes archivos: web.php, api.php, console.php y channels.php **[2].**
- **storage:** Este directorio contiene cualquier archivo generado por la APIREST Para la comunicación se debe crear generar un enlace simbólico en que apunte a este directorio con nuestro directorio *public*. Para ello hacemos uso del comando '*php artisan storage:link'* **[2].**
- **tests:** Este directorio contiene los archivos que ejecutan las pruebas automatizadas **[2].**
- **vendor:** Este directorio contiene las dependencias de Composer añadidas en su instalación **[2]**.

Con esto podemos levantar nuestro servidor como observamos en la **Fig 6**.

Fig 6: Ejecución del comando 'php artisan serve' para levantar el servicio de Laravel

Al utilizar un *framework* como Laravel, nos facilita creando el proyecto con un repositorio git local, por lo cual su vinculación hacia un repositorio en la nube es sencilla, para este proyecto se va a utilizar un repositorio en GitHub.

Con esto realizamos nuestro *initial commit*, en el cual se envía la estructura inicial de los directorios, como vemos en la **Fig 7.** Y damos por finalizada la iteración 0.

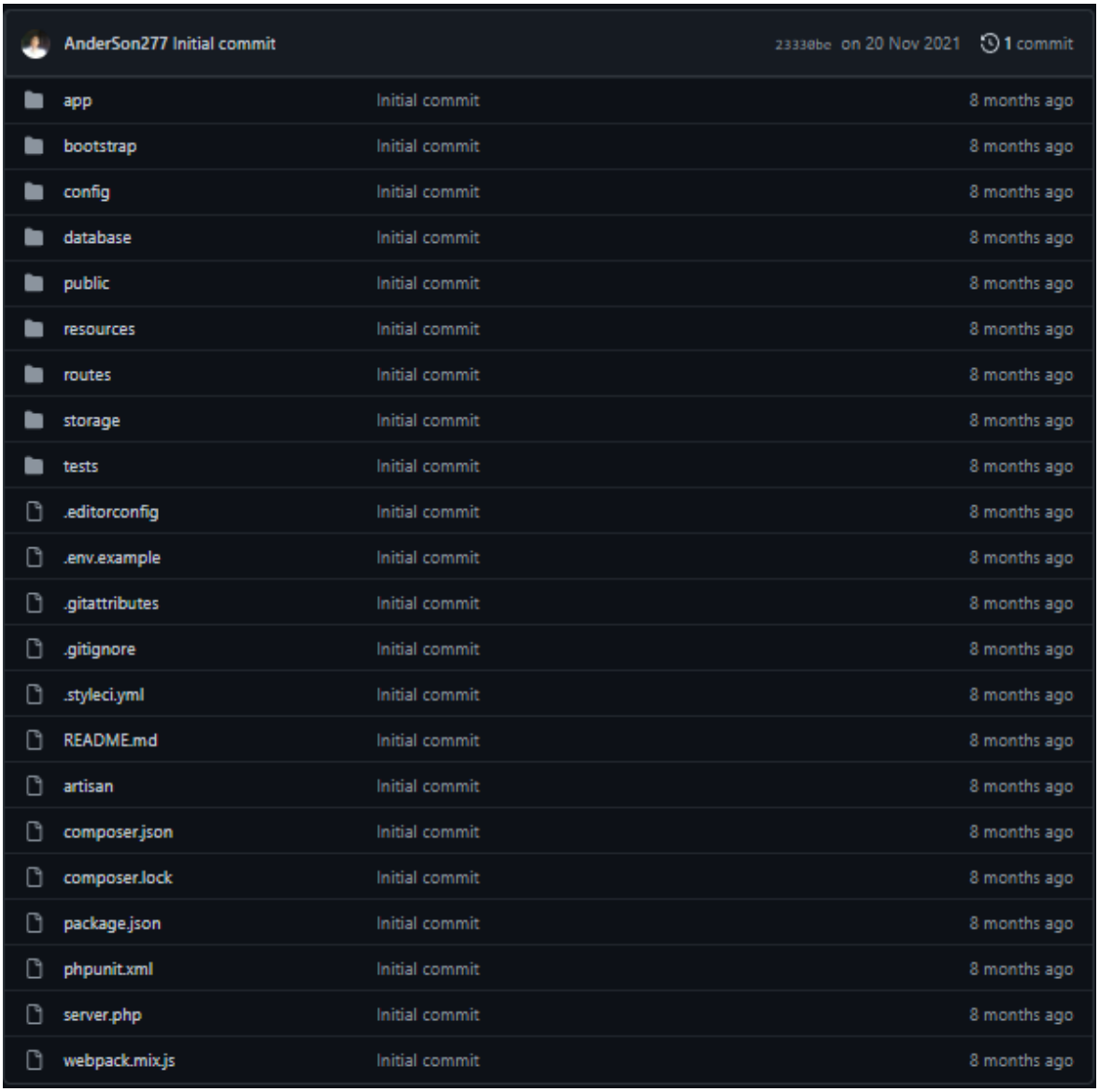


Fig 7: *Initial commit* en GitHub

3.2 Iteración 1

Para la Iteración 1, se procede a realizar las configuraciones básicas de la APIREST en el archivo config/app.php, como por ejemplo para establecer la zona horaria de Ecuador modificado el 'timezone' => 'America/Guayaquil'.

- **Configuración de JWT**

Laravel nos permite trabajar con JWT, estándar explicado en la sección 1, a través de los siguientes pasos: comenzando por instalar como dependencia de nuestro proyecto con el comando *'composer require tymon/jwt-auth:dev-develop --prefer-source'*. Una vez instalado dentro de nuestro archivo config/app.php se añade los *providers* y los *facades* como se aprecia en la **Fig 8**.7

```
'JWTAuth' => Tymon\JWTAuth\Facades\JWTAuth::class,
'JWTFactory' => Tymon\JWTAuth\Facades\JWTFactory::class,

Tymon\JWTAuth\Providers\LaravelServiceProvider::class,
```

Fig 8: Providers y Facades para la configuración JWT en el archivo config/app.php

Luego, es necesario publicar el archivo de configuración de JWT en el directorio vendor a través del siguiente comando *'php artisan vendor:publish --*'

Por último, ejecutamos el comando '*php artisan jwt:secret*' para generar nuestra *jwt-auth secret* de nuestra aplicación, la cual veremos como variable de entorno en nuestro archivo .env.

- **Inicio de sesión y registro**

Lista la configuración para utilizar JWT en nuestra APIREST, se procede con la creación del controlador para usuarios con el comando *'php artisan make:controller UserController'* en el cual se ha implementado las funciones de *authenticate, register* y *getAuthenticadetUser* como se aprecia en las **Fig 9**, **Fig 10** y **Fig 11**.

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

Fig 9: Código fuente de la función *authenticate* en el *UserController.*

```php
public function register(Request $request)
{
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:50',
        'last_name' => 'required|string|max:50',
        'email' => 'required|string|email|max:100|unique:users',
        'password' => 'required|string|min:6|confirmed',
        'avatar' => 'required|string',
    ]);
    if ($validator->fails()) {
        return response()->json($validator->errors()->toJson(), 400);
    }

    $user = User::create([
        'name' => $request->get('name'),
        'last_name' => $request->get('last_name'),
        'email' => $request->get('email'),
        'avatar' => $request->get('avatar'),
        'password' => Hash::make($request->get('password')),
    ]);
    $token = JWTAuth::fromUser($user);
    return response()->json(compact('user', 'token'), 201);
}
```

Fig 10: Código fuente de la función *register* en el *UserController.*

```php
public function getAuthenticatedUser()
{
    try {
        if (!$user = JWTAuth::parseToken()->authenticate()) {
            return response()->json(['user_not_found'], 404);
        }
    } catch (TokenExpiredException $e) {
        return response()->json(['token_expired'], 401);
    } catch (TokenInvalidException $e) {
        return response()->json(['token_invalid'], 401);
    } catch (JWTException $e) {
        return response()->json(['token_absent'], 401);
    }
    return response()->json(compact('user'), 200);
}
```

Fig 11: Código fuente de la función *getAuthenticatedUser* en el *UserController.*

Para este punto es necesario utilizar la plataforma de Postman para verificar las rutas disponibles, así que una vez añadimos nuestras funciones a una ruta en el archivo routes/api.php y tras habilitar hacemos las pruebas unitarias correspondientes, las cuales se encuentran en el Manual Técnico –Sección – Pruebas Unitarias (Pág. 7 - 15).

3.3 Iteración 2

Durante esta iteración se procede a implementar las funciones del CRUD (*create, read, update, delete*) para cada entidad de nuestro APIREST. Al igual procedemos a modelar los modelos mediante una relación de uno a muchos.

- **Relación uno a muchos**

Para esta aplicación se ha establecido una relación de uno a muchos entre la entidad de *user* y *property*; en vista que las inmuebles serán proporcionadas por los usuarios y a su vez cada uno de estas va a pertenecer a un único usuario.

Esto lo podemos establecer en Laravel a través de sus modelos. Hacemos uso del modelo *User* por defecto que se genera en nuestro APIREST, y se crea un nuevo modelo *Property* a través del comando *'php artisan make:model Property -m'* con la opción –m para crear el modelo junto con su migración.

Tras esto, se establece la relación haciendo uso de los métodos proporcionados por el Eloquent de Laravel: *hasMany()* y *belognsTo()* en cada modelo según corresponda como vemos en las **Fig *12* y Fig 13.**

```php
public function properties()
{
    return $this->hasMany('App\Models\Property');
}
```

Fig 12: Código fuente de la función *properties* con método *hasMany*() en el modelo *User*

```php
public function user()
{
    return $this->belongsTo('App\Models\User');
}
```

Fig 13: Código fuente de la función *user* con método *belongsTo()* en el modelo *Property*

- **CRUD User**

En la Iteración 1 se implementó el controlador para el usuario con las sus primeras funciones correspondientes al CRUD, como fueron las funciones de *autenticación* y *register*, por lo que dentro del mismo archivo se ha implementado las funciones de *update, delete* y *logout,* como se aprecia en las **Fig 14**, **Fig 15** y **Fig 16**.

```php
public function update(Request $request)
{
    try {
        if (!$user = JWTAuth::parseToken()->authenticate()) {
            return response()->json(['user_not_found'], 404);
        }
    } catch (TokenExpiredException $e) {
        return response()->json(['token_expired'], 401);
    } catch (TokenInvalidException $e) {
        return response()->json(['token_invalid'], 401);
    } catch (JWTException $e) {
        return response()->json(['token_absent'], 401);
    }
    //validacion de campos
    $validator = Validator::make($request->all(), [
        'name' => 'nullable|string|max:50',
        'last_name' => 'nullable|string|max:50',
        'email' => 'nullable|string|email|max:100|unique:users',
        'password' => 'nullable|string|min:6|confirmed',
        'avatar' => 'nullable|string',
    ]);
    if ($validator->fails()) {
        return response()->json($validator->errors()->toJson(), 400);
    }
    $user->update($request->all());
    return response()->json(compact('user'), 200);
}
```

Fig 14:Código fuente de la función *update* en el *UserController*

```php
public function delete()
{
    try {
        if (!$user = JWTAuth::parseToken()->authenticate()) {
            return response()->json(['user_not_found'], 404);
        }
    } catch (TokenExpiredException $e) {
        return response()->json(['token_expired'], 401);
    } catch (TokenInvalidException $e) {
        return response()->json(['token_invalid'], 401);
    } catch (JWTException $e) {
        return response()->json(['token_absent'], 401);
    }
    DB::statement('SET FOREIGN_KEY_CHECKS=0;');
    $user->delete();
    DB::statement('SET FOREIGN_KEY_CHECKS=1;');
    return response()->json(null, 204);
}
```

Fig 15: Código fuente de la función *delete* en el *UserController*

```php
public function logout(){
    auth()->logout(true);
    response()->json(["message" => "logged_out"], 200);
}
```

Fig 16: Código fuente de la función *logout* en el *UserController.*

Procedemos a realizar las pruebas en Postman como se puede encontrar en el Manual Técnico –Sección – Pruebas Unitarias (Pág. 7 - 15).

Tras realizar las pruebas enviar los cambios al repositorio en GitHub.

- **CRUD Property**

Creamos un controlador para la entidad de inmuebles a través del comando *'php artisan make:controller PropertyController'* y dentro de este archivo se ha implementado las funciones de *index, show, store, update* y *delete* las cuales se detalla en la **TABLA VI** y se visualizan en las **Fig 17 Fig 18 Fig 19 Fig 20 y Fig 21**

TABLA VI: Resumen funciones CRUD en *PopertyController*

Nombre	Función
index()	Regresa todos los documentos de la entidad
show(Property $property)	Regresa un documento en específico, el cual se establece es sus parámetros
store(Request $request,)	Crea un documento con información que establece en sus parámetros
update(Request $request, Property $property)	Actualiza un documento en específico, el cual se establece en sus parámetros al igual que la nueva información
delete(Property $property)	Elimina un documento en específico, el cual se establece es sus parámetros

```php
public function index()
{
    return Property::all();
}
```

Fig 17: Código fuente de la función *index* en *PropertyController*

```php
public function show(Property $property)
{
    return response()->json($property, 200);
}
```

Fig 18: Código fuente de la función *show* en *PropertyController*

```php
public function store(Request $request)
{
    $validator = Validator::make(
        $request->all(),
        [
            'title' => ['required', 'string', 'max:255'],
            'area' => ['required', 'integer'],
            'bedrooms' => ['required', 'integer'],
            'bathrooms' => ['required', 'integer'],
            'livingrooms' => ['required', 'integer'],
            'kitchens' => ['required', 'integer'],
            'parkings' => ['required', 'integer'],
            'photos' => ['required', 'string'],
            'description' => ['required', 'string'],
            'address' => ['required', 'string'],
            'price' => ['required', 'integer'],
            'type' => ['required', 'string']
        ]
    );
    if ($validator->fails()) {
        return response()->json($validator->errors()->toJson(), 400);
    }
    $property = new Property($request->all());
    $property->user_id = Auth::id();
    $property->save();

    return response()->json($property, 201);
}
```

Fig 19: Código fuente de la función *store* en *PropertyController*

```php
public function update(Request $request, Property $property)
{
    $validator = Validator::make(
        $request->all(),
        [
            'title' => ['nullable', 'string', 'max:255'],
            'area' => ['nullable', 'integer'],
            'bedrooms' => ['nullable', 'integer'],
            'bathrooms' => ['nullable', 'integer'],
            'livingrooms' => ['nullable', 'integer'],
            'kitchens' => ['nullable', 'integer'],
            'parkings' => ['nullable', 'integer'],
            'photos' => ['nullable', 'string'],
            'description' => ['nullable', 'string'],
            'address' => ['nullable', 'string'],
            'price' => ['nullable', 'integer'],
            'type' => ['nullable', 'string']
        ]
    );
    if ($validator->fails()) {
        return response()->json($validator->errors()->toJson(), 400);
    };
    $property->update($request->all());
    return response()->json($property, 200);
}
```

Fig 20: Código fuente de la función *update* en *PropertyController*

```php
public function delete(Property $property)
{
    $property->delete();
    return response()->json(null, 204);
}
```

Fig 21: Código fuente de la función *delete* en *PropertyController*

- **Migraciones y Seeders**

Dentro del directorio *database* de nuestro proyecto Laravel podremos encontrar los directorios de *seeders* y *migrations*. Las migraciones se añadieron al utilizar la opción de –m en la creación de modelos, solo es necesario definir los tipos de campos de cada entidad y ejecutar el comando *'php artisan migrate*' para migrar a nuestra base de datos **[2]**.

También, se ha creado *seeders* a través del comando '*php artisan make:seeder*' para las tablas de *user* y *property*. Esto con el objetivo de popular las tablas de la base de datos con datos ficticios haciendo uso de la librería *Faker*.

Finalmente, en estos archivos se han escrito las funciones para crear un usuario de prueba y 10 usuarios ficticios, a su vez que por cada usuario se ha creado un inmueble, manteniendo la relación de los modelos como vemos en la **Fig 22** y **Fig 23**.

```php
    public function run()
    {
        // Vaciar la tabla
        User::truncate();

        //LLamado a libreria faker
        $faker = \Faker\Factory::create();

        /* Crear la misma clave para todos los usuarios
        conviene hacerlo antes del for para que el seeder
        no se vuelva lento.*/
        $password = Hash::make('123123');

        // Usuario de prueba
        User::create([
            'name' => 'Anderson',
            'last_name' => 'Cordova',
            'email' => 'ander@prueba.com',
            'password' => $password,
            'avatar' => 'avatar'
        ]);

        //Usuarios con faker

        for ($i = 0; $i < 10; $i++) {
            User::create([
                'name' => $faker->name,
                'last_name' => $faker->lastName,
                'email' => $faker->email,
                'password' => $password,
                'avatar' => $faker->sentence
            ]);
        }
    }
}
```

Fig 22: Código fuente de *seeders* para *User*

```php
public function run()
{
    // Vaciar la tabla
    Property::truncate();

    //LLamado a libreria faker
    $faker = \Faker\Factory::create();

    //Obtenemos todos los usuarios registrados
    $users = User::all();

    //Creamos 1 propiedad faker porcada usuario registrado
    foreach ($users as $user) {
        // Iniciar sesión con cada Teacher
        JWTAuth::attempt(['email' => $user->email, 'password' => '123123']);
        // Crear 1 Course por cada Teacher que tengamos en la BD
        Property::create([
            'title' => $faker->sentence,
            "area" => $faker->numberBetween(100, 1000),
            "bathrooms" => $faker->numberBetween(1, 5),
            "bedrooms" => $faker->numberBetween(1, 5),
            "kitchens" => $faker->numberBetween(1, 5),
            "livingrooms" => $faker->numberBetween(1, 5),
            "parkings" => $faker->numberBetween(1, 5),
            "photos" => [
                '/storage/properties/' . $faker->image('public/storage/properties', 400, 300, null, false),
                '/storage/properties/' . $faker->image('public/storage/properties', 400, 300, null, false),
                '/storage/properties/' . $faker->image('public/storage/properties', 400, 300, null, false)
            ],
            "description" => $faker->paragraph,
            "address" => $faker->sentence,
            "price" => $faker->numberBetween(1000, 5000),
            "type" => "VENTA",
            "user_id" => $user->id,
        ]);
    }
}
```

Fig 23: Código fuente de *seeders* para *Property*

3.4 Iteración 3

Esta iteración cubre el desarrollo del motor de búsqueda de las inmuebles y la vinculación de almacenamiento externo para las imágenes, utilizando servicios externos

- **Implementación de motor de búsqueda con Algolia**

Debemos crear una cuenta en el sitio oficial de Algolia https://www.algolia.com con ello, obtener las API Key's como vemos en la **Fig 24,** que posteriormente se añaden a nuestro archivo .env

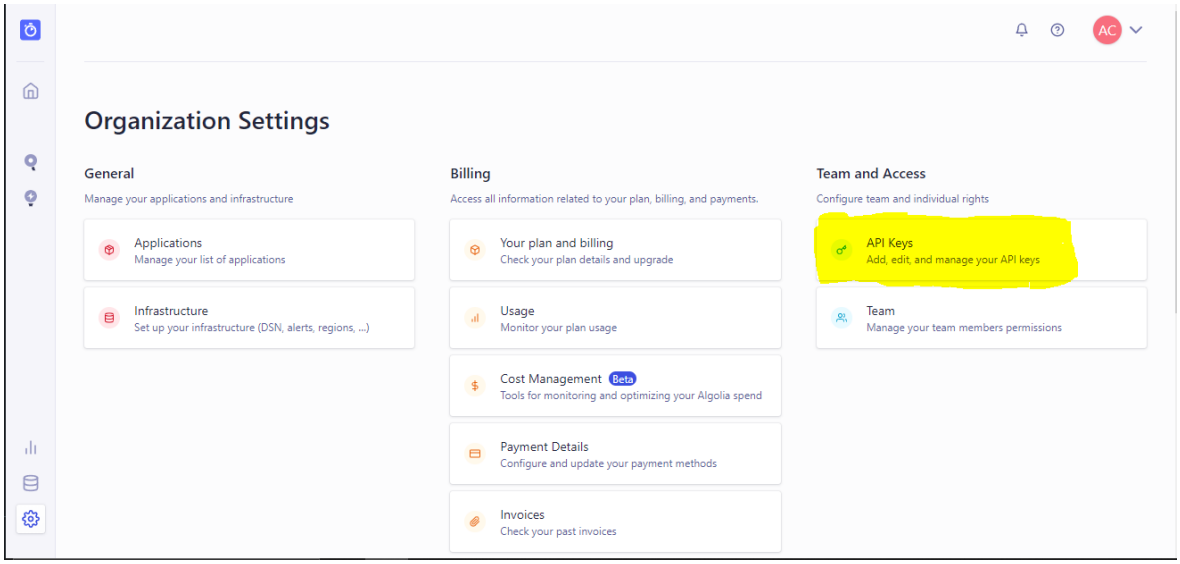


Fig 24: Obtención de API Key's en la plataforma de Algolia

Se procede con la instalación del paquete Scout en nuestro proyecto con el comando '*composer require laravel/scout* ' y se añade el *provider* en el mismo archivo de configuración. Por último, se añade la instancia de *Serchable* y los métodos establecidos de *toSearchableArray()* y *searchableAs()* en el modelo que vayamos a indexar a Algolia, que en nuestra caso es el modelo de *Property* como se aprecia en la **Fig 25**

```php
public function searchableAs()
{
    return 'properties';
}

public function toSearchableArray()
{
    $array = $this->toArray();

    $array['title'] = $this->title;
    $array['description'] = $this->description;
    $array['area'] = $this->area;
    $array['bedrooms'] = $this->bedrooms;
    $array['bathrooms'] = $this->bathrooms;
    $array['livingrooms'] = $this->livingrooms;
    $array['kitchens'] = $this->kitchens;
    $array['parkings'] = $this->parkings;

    return $array;
}
```

Fig 25: Código fuente de los métodos *toSearchableArray*() y *searchableAs*() en el modelo *Property*

Lista la instalación y configuración de Scopus en nuestro proyecto, se indexa nuestro modelo a de *Property* a Algolia mediante el comando *'php artisan scout:import App\Models\Property'*

Y finalmente, dentro del controlador de *Property* se crea la función que establece nuestro búsqueda en Algolia utilizando una cláusula *where* para filtrar la información a través de los parámetros que se nos brinde; teniendo en cuenta que es posible que algunos parámetros brindados contengan el valor de *null,* por lo que se establece un condicional *if()* en nuestro ciclo *forEach()*, extenuando también los valores con campo de texto, pues estos se buscan de manera más sencilla a través del método establecido por Scopus *search()*. Todo este código lo encontramos en la **Fig 26.**

```php
public function searchEngine(Request $request)
{

    $clauses = [];

    $params = $request->all();

    foreach ($params as $key => $value)
        if ($key === 'title' || $key === 'description' || $key === 'address') {

            $clauses[] = [$key, 'LIKE', '%' . $value . '%'];
        } else {

            if ($key === 'page') continue;
            $clauses[] = [$key, '>=', $value];
        }

    return Property::where($clauses)->paginate(5);
}
```

Fig 26: Código fuente de la función *searchEngine* en *PropertyController*

- **Implementación AWS S3 para imágenes**

Se registra una cuenta en la página oficial de AmazonS3 disponible en **https://aws.amazon.com/es/s3/.** Y después se crea un *bucket* de acceso público para almacenar las imágenes generadas de nuestra APIREST **[26]**.
Obtenemos las credenciales de seguridad del recipiente para añadirlas al archivo .env de nuestro proyecto y comenzar a trabajar con S3 en nuestra APIREST. Con ello, se implementa dentro de las funciones de *register, store* y *update* el código para guardar las imágenes a nuestro servidor y almacenar una URL de acceso de la imagen en nuestra base de datos.

Cabe mencionar, que para el *store* y *update* del *PropertyController* se puede generar un arreglo de hasta 10 imágenes y para el *UserControler* tenemos permitida 1 foto de perfil, como podemos apreciar en las ***Fig 27*** **y** ***Fig 28***.

```php
//Upload File to s3
Storage::disk('s3')->setVisibility($path, 'public');
$user->avatar = Storage::disk('s3')->url($path);
```

Fig 27: Código fuente para almacenar la foto de perfil en AmazonS3

```php
//Subiendo imagenes al servidor y entregarndo el url
if ($request->hasFile('photos')) {
    $ListPhotos = array();
    foreach ($request->file('photos') as $photo) {
        $path = $photo->store('properties', 's3');
        Storage::disk('s3')->setVisibility($path, 'public');
        array_push($ListPhotos, Storage::disk('s3')->url($path));
    }
    $property->photos = $ListPhotos;
}
```

Fig 28: Código fuente para almacenar un arreglo de imagenes de un inmueble en AmazonS3

3.5 Pruebas del Sistema

Para esta fase antes del despliegue, es necesario realizar pruebas para comprobar el correcto funcionamiento de la APIREST. Las pruebas son necesarias para que un sistema pueda ser sostenible con el tiempo.

- **Pruebas Unitarias**

Las pruebas unitarias consisten en centrase en una única funcionalidad o método y se ejecutan de manera rápida **[27]**.

Estas pruebas se han venido ejecutando con la herramienta de testeo de Postman durante las iteraciones del proyecto para comprobar con las distintas funciones y rutas de manera individual. Obteniendo como resultado la colección de rutas de nuestra aplicación como vemos en la ***Fig*** **29**.

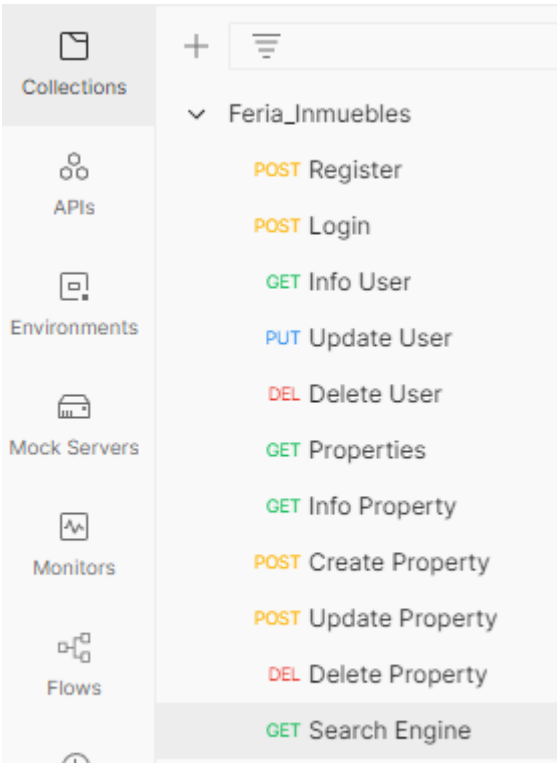


Fig 29: Colección de rutas en Postman

Postman nos permite crear un ambiente de pruebas nivel local y de producción tras el despliegue, esto junto con las respuestas de cada ruta se encuentran en el Manual Técnico –Sección – Pruebas Unitarias (Pág. 12 - 20).

También, se han ejecutado test de prueba, para ello, Laravel proporciona el comando *'vendor/bin/phpunit'* con el cual se obtuvo el resultado de la **Fig 30**

```
ander@DESKTOP-VOBAN2E MINGW64 ~/Desktop/Ander/2021-B/Tesis/BackEnd/api_feria_inmuebles (master)
$ vendor/bin/phpunit
PHPUnit 9.5.13 by Sebastian Bergmann and contributors.

..                                                                  2 / 2 (100%)

Time: 00:19.813, Memory: 22.00 MB

OK (2 tests, 2 assertions)
```

Fig 30: Ejecución de pruebas unitarias con el comando 'vendor/bin/phpunit"

- **Pruebas de Rendimiento**

Para comprobar el rendimiento de nuestra APIREST en Laravel hacemos uso de la herramienta de prueba de carga LoadView, tras exportar la colección de Postman obtenida con las Pruebas Unitarias, se establece los paramentros para ejecutar una prueba de carga en Load View.

Obtendremos una gráfica que representa los niveles de carga que puede llegar a tener nuestra APIREST con una cantidad aceptable de usuarios como se aprecia en la **Fig 31**, el

resto de graficas obtenidas con estas pruebas se encuentran en el Manual Técnico –Sección – Pruebas Rendimiento (Pág. 12 - 20).

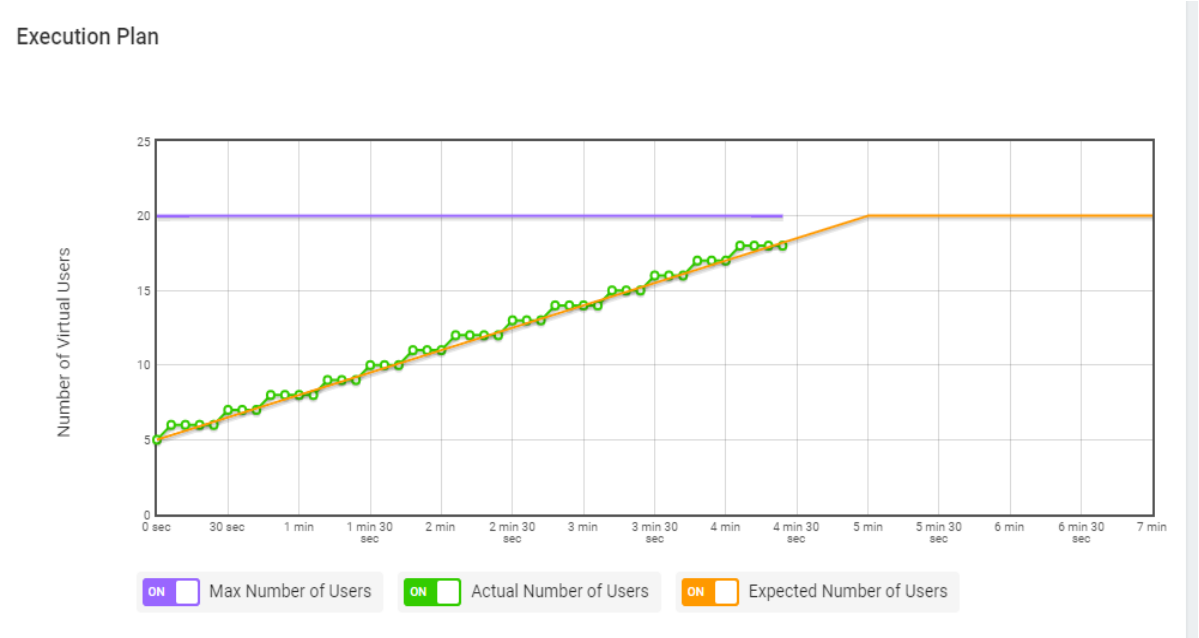


Fig 31: Gráfica de pruebas de carga para el número de usuarios virtuales en la API REST

- **Pruebas de Aceptación**

En este tipo de prueba la metodología XP propone, con mayor grado mayor de implicación con el cliente para corroborar su estado de satisfacción **[16] [27]**.

Para nuestra APIREST el cliente directo es el desarrollador del frontend con el cual se mantuvo otro principio de XP como lo es la comunicación, para establecer las pruebas de aceptación de este proyecto.

A continuación, la **TABLA VII** presenta un ejemplo de una las pruebas de aceptación, las restantes, se encuentran en el Manual Técnico – Sección - Pruebas de Aceptación (Pág. 15 - 22).

TABLA VII: Prueba de Aceptación: PA001 Inicio de Sesión

	PRUEBA DE ACEPTACIÓN
Identificador de prueba (ID): PA001	**Identificador historia de usuario:** HU001
Nombre prueba de aceptación: Inicio de sesión	
Descripción: Los usuarios registrados en la base de datos deben poder obtener su token al momento de iniciar sesión en la APIREST	
Condiciones de prueba: • Tener conexión a internet • Ingresar a la URL donde se aloja el proyecto	

Pasos de ejecución: • Abrir nuestra plataforma de testeo, en nuestro caso Postaman. • En la barra de búsqueda ingresar a la url: https://api-feria-inmueble s.herokuapp.com/api/login • Ingresar las credenciales en el cuerpo de la petición
Resultado deseado: Recibe el token de acceso o un mensaje con el error detectado
Evaluación de la prueba: Se comprobó el resultado esperado. Aprobación del cliente 100%

3.6 Despliegue

Para el despliegue del proyecto se planifico en la plataforma de servicio de Herkou, por lo que en este punto hemos creado registrado una cuenta en https://www.heroku.com y vincular nuestro repositorio en GitHub **[12].**

Se añade un archivo *Procfile* a nuestro proyecto cambiado el apuntador de apache2 a public/, ya que por defecto va a web/ con la línea de código vista en la **Fig 32**.

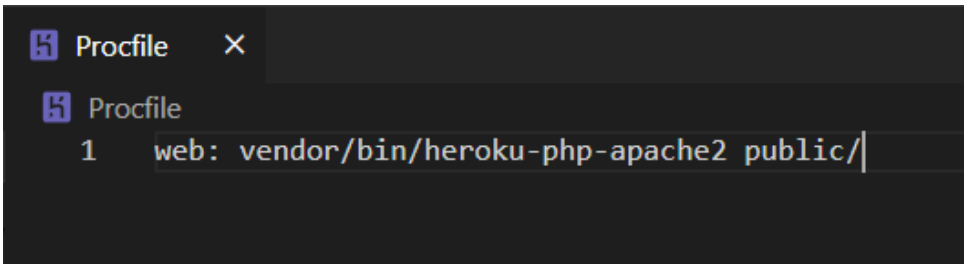


Fig 32: Archivo Procfile del proyecto

Es necesario instalar HerokuCLI, en nuestro caso al estar un sistema operativo en Windows solo es necesario ejecutar el instalador el cual obtenemos de su sitio oficial disponible en https://devcenter.heroku.com/articles/heroku-cli **[12]** .

Luego, en la carpeta raíz de nuestro proyecto ejecutamos el comando *'herouku create'* y en su primera ejecución nos solicita iniciar sesión en Heroku.

Una vez ingresamos con nuestra credenciales, es necesario configurar las variables de entorno en Heroku con las que utilizamos en nuestro archivo .env. Este punto es necesario para mantener en el despliegue los servicios externos de nuestro APIREST y se puede realizar a través de la terminal con el comando '*heroku config:set [name enviroment]*'

Por último, ejecutamos el comando *'git push heroku master'* para desplegar los cambios de nuestro repositorio al servicio en la nube de Heroku como apreciamos en la **Fig 33.**

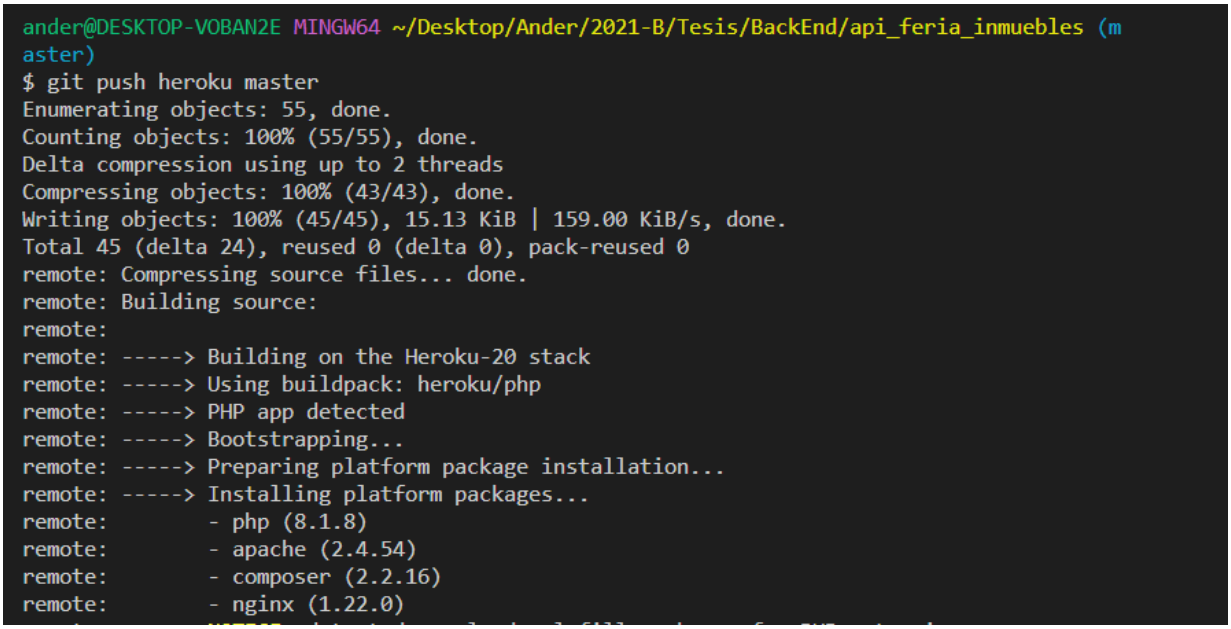


Fig 33: Ejecución del comando *'git push heroku master' para el despliegue de la API REST en Heroku*

4. CONCLUSIONES Y RECOMENDACIONES

En este apartado se enumera las conclusiones y posibles recomendaciones tras la finalización del proyecto.

4.1 Conclusiones

- Se ha desarrollado una APIREST que permite la publicación y búsqueda de bienes inmuebles.
- Se ha logrado solventar los requerimientos del desarrollador de frontend para proporcionar las rutas en base a las historias de usuario.
- La arquitectura REST ha permitido un desarrollo estandarizado para manejar la interacción con los recursos mediante los métodos HTTP.
- Laravel es un *framework* muy completo que nos ha facilitado el desarrollo de este proyecto gracias a sus herramienta y métodos preestablecidos tanto para las pruebas como para su implementación.
- Las pruebas han servido para corroborar el correcto funcionamiento y aceptación del desarrollador frontend.
- La APIREST diseñada es portable y factible para cualquier aplicativo web o móvil.

4.2 Recomendaciones

- El mantenimiento de la APIREST se debe hacer, conforme los nuevos requerimientos del frontend.
- Se debe hacer un mantenimiento a la base para depurar información, para que contenga datos verídicos acerca de los bienes inmuebles
- Tener en consideración la capacidad limitada del *bucket* en AWS S3, para conservar, si así se desea, su instancia gratuita
- Cuidar la cantidad de peticiones en el buscador de Algolia. para conservar, si así se desea, su instancia gratuita

5. REFERENCIAS BIBLIOGRÁFICAS

[1] maldeadora, «Que es Frontend y Backend: diferencias y características,» 2018. [En línea]. Available: https://platzi.com/blog/que-es-frontend-y-backend/.

[2] S. Chalo, «CREACIÓN DE UNA API REST CON LARAVEL,» Quito, Ecuador, 2020.

[3] C. Simoes, «La importancia de las APIs y Web APIs,» CSimoes, 16 Marzo 2016. [En línea]. Available: https://www.chiyanasimoes.com/blog/la-importancia-de-las-apis-y-web-apis#:~:text=Las%20APIs%20ayudan%20a%20dar,confidencialidad%20del%20código%20es%20elemental..

[4] E. S. P. CARRASCO, «LA UTILIZACIÓN DE LAS TICS EN LA GESTIÓN COMERCIAL DE LAS PYMES PARA SERVICIOS DE BIENES INMUEBLES,» Guayaquil/Ecuador, 2013.

[5] I. G. Rodríguez, «DEFINICIÓN E IMPLEMENTACIÓN DE UNA API REST PARA SISTEMAS DE RECOMENDACIÓN,» Escuela Politecnica Superior, Madrid, 2018.

[6] A. T. A. J. C. Z. L. Marleny Loaiza Soto, «EL MARKETING RELACIONAL FRENTE A LOS AVANCES TECNOLÓGICOS EN EL SECTOR INMOBILIARIO EN LA CIUDAD DE MEDELLÍN,» Medellín, Colombia, 2019.

[7] S. Mira, «DIGITALIZACIÓN Y TENDENCIAS DEL SECTOR INMOBILIARIO,» Cataluña,España, 2019.

[8] R. Arjonillo, «Que es el back-end y porque es importante para tu web,» 2016. [En línea]. Available: https://rafarjonilla.com/que-es/backend/.

[9] JSON, «Introducing JSON,» 2022. [En línea]. Available: https://www.json.org/json-en.html.

[10] JWT, «Introduction to JSON Web Tokens,» Auth0, 2022. [En línea]. Available: https://jwt.io/introduction/.

[11] Algolia, «Documentation Algolia,» Algolia, 2022. [En línea]. Available: https://www.algolia.com/doc/.

[12] Heroku, «Documentation Heroku,» Heroku Dev Center, 2022. [En línea]. Available: https://devcenter.heroku.com/categories/reference.

[13] R. Celis, «¿Qué es Heroku? Cómo funciona la plataforma y para qué sirve,» Platzi, 2018. [En línea]. Available: https://platzi.com/blog/que-es-heroku/.

[14] M. M. R. C. M. Ailin Orjuela Duarte, «Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo,» Avances en Sistemas e Informática, Medellin, Colombia, 2018.

[15] K. B. w. C. Andres, Extreme Programming Explained, Boston: Addison-Wesley, 2004.

[16] B. Kent y A. Cynthia, Extreme Programming Explained, Boston: Addison-Wesley, 2012.

[17] B. Buegree, «Agile Methodologies XP and Scrum,» Technische Universitaet, Muenchen, 2009.

[18] G. Sambasivam, «EXTREME PROGRAMMING(XP)».

[19] G. O., «Recopilación de requisitos,» proyectum, 1 Mayo 2013. [En línea]. Available: https://www.proyectum.com/sistema/blog/recopilacion-de-requisitos/.

[20] A. Álvarez, «Historias de Usuario: qué son, reglas y consejos.,» netmind, 6 Julio 2020. [En línea]. Available: https://netmind.net/es/historias-de-usuario-reglas/#:~:text=Las%20Historias%20de%20Usuario%20son,la%20hora%20de%20definir%20requisitos..

[21] H. Cervantes, «Arquitectura de Software,» SG, 2020. [En línea]. Available: https://sg.com.mx/revista/27/arquitectura-software.

[22] SendPulse, «Que es Interfaz de programación de aplicaciones (API) RESTful - Significado,» SendPulse, 26 Diciembre 2018. [En línea]. Available: https://sendpulse.com/latam/support/glossary/restful-api.

[23] Laravel, «Documentation Laravel,» Laravel, 2022. [En línea]. Available: https://laravel.com/docs/8.x.

[24] Composer, «Composer,» getcomposer, 2022. [En línea]. Available: https://getcomposer.org/doc/00-intro.md.

[25] A. Friends, «XAMPP,» ApacheFriends, 2022. [En línea]. Available: https://www.apachefriends.org/es/index.html.

[26] A. S3, «AmazonS3,» AWS, 2022. [En línea]. Available: https://aws.amazon.com/es/s3/.

[27] D. I. M. R. M. J. d. P. A. Z. y M. R. C. S. C. A. , «PRUEBAS DE SOFTWARE,» Juliana del Pilar Alva Zapata, Bagua, 2021.

[28] Atlassian, «¿En qué consiste la integración continua?,» Atlassian, 2020. [En línea]. Available: https://www.atlassian.com/es/continuous-delivery/continuous-integration.

[29] Cognodata, «Arquitectura de datos: la base de una estrategia diferenciadora,» 25 Abril 2019. [En línea]. Available: https://www.cognodata.com/blog/arquitectura-datos-estrategia-diferenciadora/.

6. ANEXOS

A continuación, se enlistan todos los documentos que se generaron a partir del desarrollo de la API REST.

6.1 Porcentaje de Plagio

6.1 Manual Técnico

1. Historias de Usuarios
2. Diseño de Base de Datos
3. Pruebas Funcionales
    a. Pruebas Unitarias
    b. Pruebas de Aceptación
    c. Pruebas de Rendimiento

6.2 Manual de Usuario

El manual de usuario se encuentra disponible en el siguiente enlace → https://youtu.be/ngLvkzeIrLo

El repositorio se encuentra disponible en el siguiente enlace → https://github.com/AnderSon277/API_Feria_Inmuebles

ANEXO I. CERTIFICADO DE ORIGINALIDAD

ANEXO II. MANUAL TÉCNICO

DESARROLLO BACKEND PARA APLICACIÓN WEB DE COMPRAVENTA DE BIENES INMUEBLES

CÓRDOVA CALVOPINA ANDERSON IVÁN

anderson.cordova@epn.edu.ec
andersoncordova277@gmail.com

ÍNDICE DE CONTENIDO

1. Historias de Usuario

A continuación, se muestran las 10 Historias de Usuario que guían la implementación de la API.

Tabla I: Historia de Usuario 1- Inicio de sesión

HISTORIA DE USUARIO	
Identificador (ID): HU001	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Inicio de sesión	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Iteración Asignada: 1	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe identificarse en el sistema a través de credenciales: usuario y contraseña, tras su comprobación retornar un token de validación o caso contario retornar el error correspondiente.	
Observación(es): • El sistema debe validar las credenciales si y sólo si estos usuarios se han registrado con anterioridad. • Cada usuario tiene la opción de cerrar su sesión	

Tabla II: Historia de Usuario 2 – Registro

HISTORIA DE USUARIO	
Identificador (ID): HU002	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Registro	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Iteración Asignada: 1	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe llenar un formulario de registro por cada usuario nuevo en el sistema, en el cual se contengan los siguientes campos: nombre, apellido, correo electrónico, celular, contraseña y avatar.	

Observación(es): • El campo nombre es un texto. • El campo apellido es un texto. • El campo celular es un texto. • El campo correo electrónico debe validar que sea un '*email*' y debe ser único en el sistema. • El campo de contraseña debe ser confirmado y tener un mínimo de 6 caracteres. • El campo de avatar es una imagen de máximo 2GB. • Todos los campos del registro son obligatorios. • Cada usuario registrado tiene un identificador único para reconocerlo en el sistema.

Tabla III: Historia de Usuario 3 – Información del Usuario

HISTORIA DE USUARIO	
Identificador (ID): HU003	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Información del Usuario	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Baja
Iteración Asignada: 1	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe obtener la información individual de cada usuario registrado una vez que este se encuentre con una sesión activa en el sistema.	
Observación: • Se debe tener una única URL, y mostrar la información de usuario en base a su JWT.	

Tabla IV: Historia de Usuario 4 – Actualizar Perfil de Usuario

HISTORIA DE USUARIO	
Identificador (ID): HU004	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Actualizar Perfil de Usuario	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Medio
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe ser capaz de actualizar la información de cualquier campo del registro (nombre, apellido, correo electrónico, celular, contraseña y avatar) para cualquier usuario previamente registrado en el sistema.	
Observación: • Sin importar el número de actualizaciones de un perfil, se mantiene un identificador único por usuario desde el momento de su registro. • Se mantienen las validaciones de los campos tal como en su registro.	

Tabla V: Historia de Usuario 5 – Eliminar Cuenta de Usuario

HISTORIA DE USUARIO	
Identificador (ID): HU005	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Eliminar Cuenta de Usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe ser capaz de eliminar cuentas de usuarios	
Observación: • Se borra el registro de la base de datos • Un usuario solo puede eliminar su cuenta si se encuentra con una sesión actica en el sistema.	

Tabla VI: Historia de Usuario 6 – Publicar Inmueble

HISTORIA DE USUARIO	
Identificador (ID): HU006	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Publicar inmueble	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API con un usuario con sesión activa tiene la posibilidad de publicar un inmueble en el sistema a través de un formulario en el que se indique: título, área, número de dormitorios, número de cocinas, número de baños, número de patios, descripción y precio.	
Observación: • El usuario debe brindar la información de todos los campos de manera obligatoria, caso contrario no se debe publicar el inmueble. • Cada publicación debe tener un identificador único.	

Tabla VII: Historia de Usuario 7 – Actualizar inmueble

HISTORIA DE USUARIO	
Identificador (ID): HU007	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Actualizar Inmueble	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	
Descripción: El equipo de desarrollo que consuma de la API con un usuario que ha realizado una publicación de un inmueble, tiene la opción de actualizar la información de esta publicación.	
Observación: • Los campos de la publicación pueden ser actualizados siempre y cuando son se reciba un valor en blanco (null).	

• No importa el número de actualizaciones, cada publicación conserva su identificador único desde el momento de su creación. • Un usuario no puede actualizar publicaciones que no sean suyas.

Tabla VIII: Historia de Usuario 8 – Información de Inmueble

HISTORIA DE USUARIO	
Identificador (ID): HU008	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Información del Inmueble	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe obtener la información individual de cada publicación de una inmueble	
Observación: • Se entrega la información de la inmueble en base a su identificador único creado en su publicación	

Tabla IX: Historia de Usuario 9 – Lista de Inmuebles

HISTORIA DE USUARIO	
Identificador (ID): HU009	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Lista de Inmuebles	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	
Descripción: El desarrollador que consuma de la API debe obtener un listado paginado de todas las publicaciones de inmueble que ha publicado un usuario en la base de datos	
Observación: • La paginación del listado se dada en un rango de 5 publicaciones por página.	

Tabla X: Historia de Usuario 10 – Eliminar Inmueble

HISTORIA DE USUARIO	
Identificador (ID): HU0010	**Usuario:** Desarrollador *Front-End*
Nombre Historia: Eliminar inmueble	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Iteración Asignada: 2	
Responsable (es): Anderson Córdova	

Descripción: El desarrollador que consuma de la API debe ser capaz de eliminar publicaciones de inmueble s hechas por los usuarios en el sistema.
Observación: • Se debe eliminar el registro en la base de datos. • Un usuario solo puede eliminar publicaciones suyas.

2. Diseño de la Base de Datos

A continuación, se define la entidad relación que servirá de guía para implementación de la base de datos de la API.

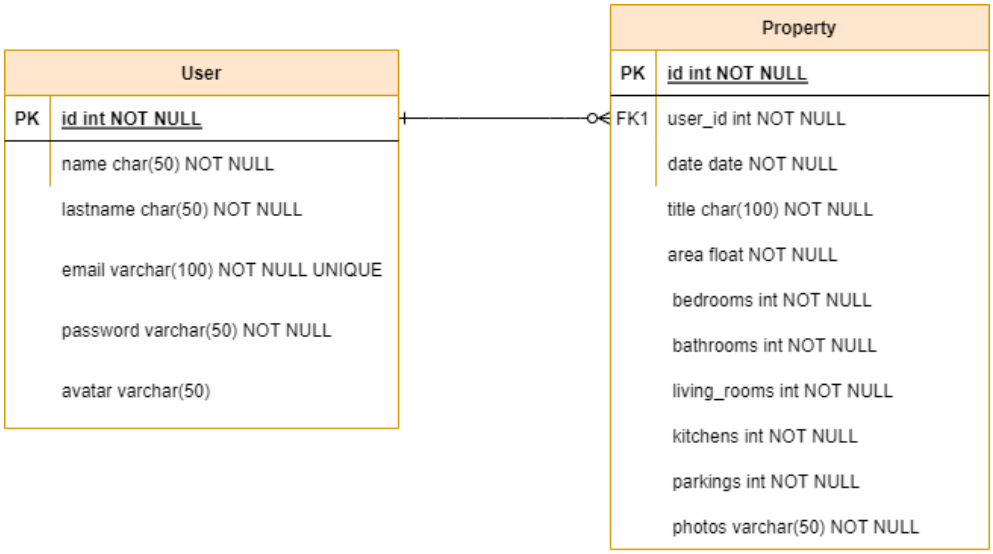


Fig 1: *Modelo Entidad-Relación*

3. Pruebas Funcionales

A continuación, se muestran las pruebas funcionales realizadas sobre la API.

3.1. Pruebas Unitarias

Haciendo uso del software de testeo, Postman, se ha generado un ambiente de pruebas unitarias con los resultados enlistado a continuación.

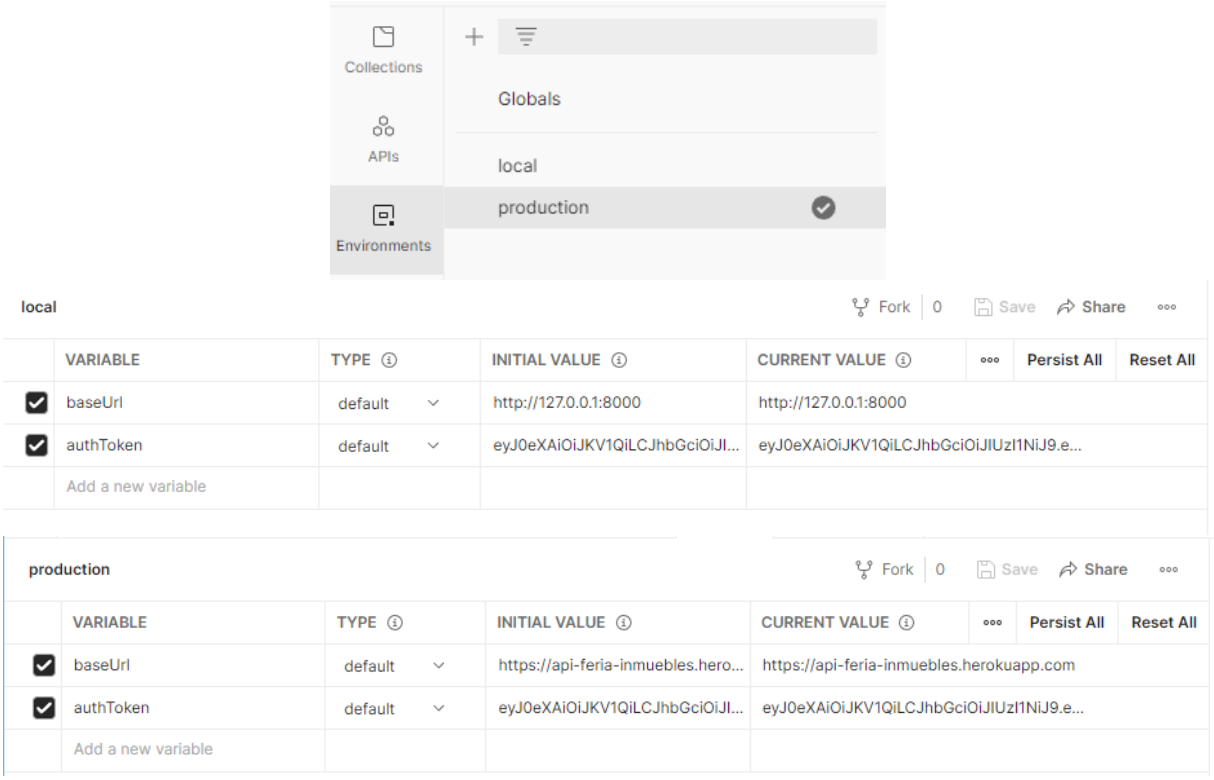


Fig 2: Ambiente de pruebas local y producción en Postman

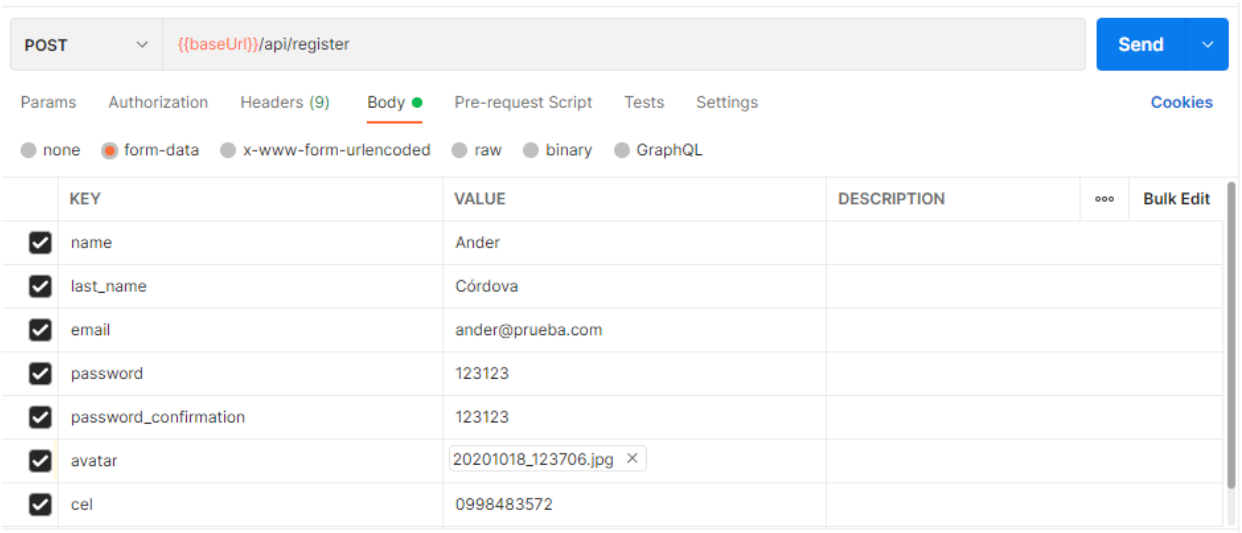


Fig 3: Cuerpo de petición para el Registro

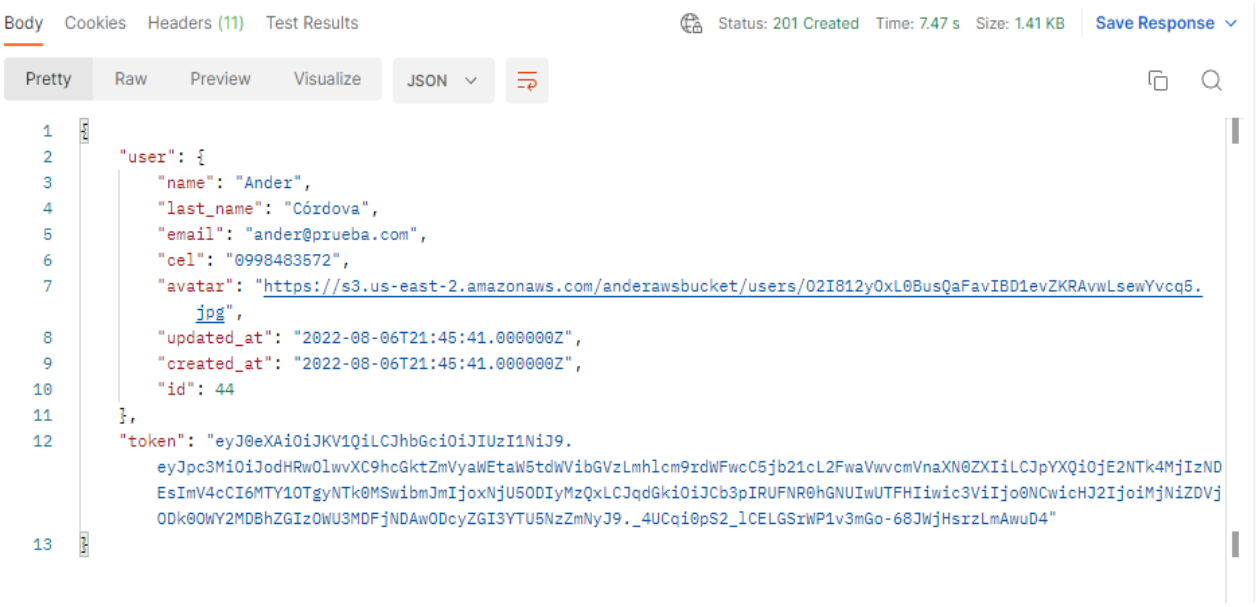


Fig 4: Respuesta a la petición de Registro

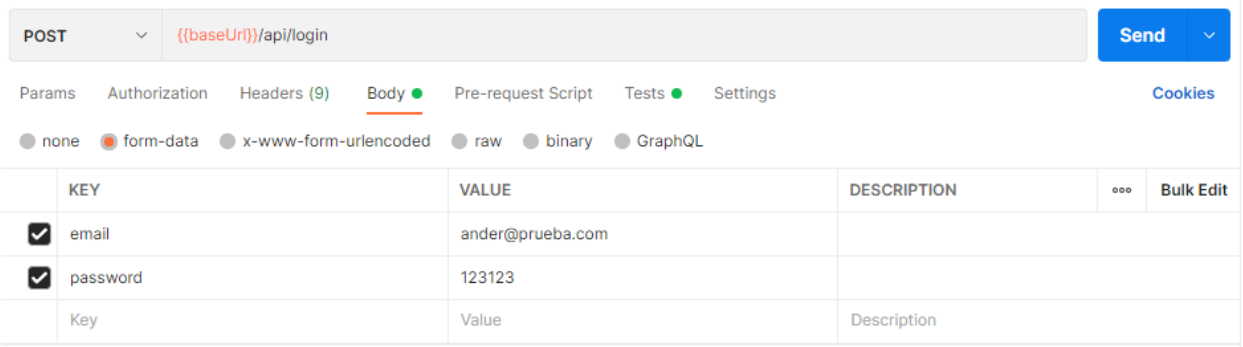


Fig 5:Cuerpo de petición para el Inicio de Sesión

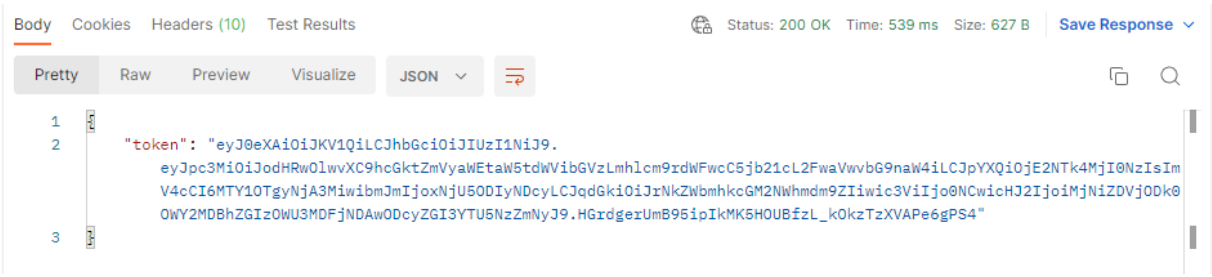


Fig 6: Respuesta a la petición del Inicio de Sesión

```
1  var jsonData = JSON.parse(responseBody);
2
3  postman.setEnvironmentVariable("authToken",jsonData.token)
```

Fig 7:Almacenar token variable de entorno “authToken”

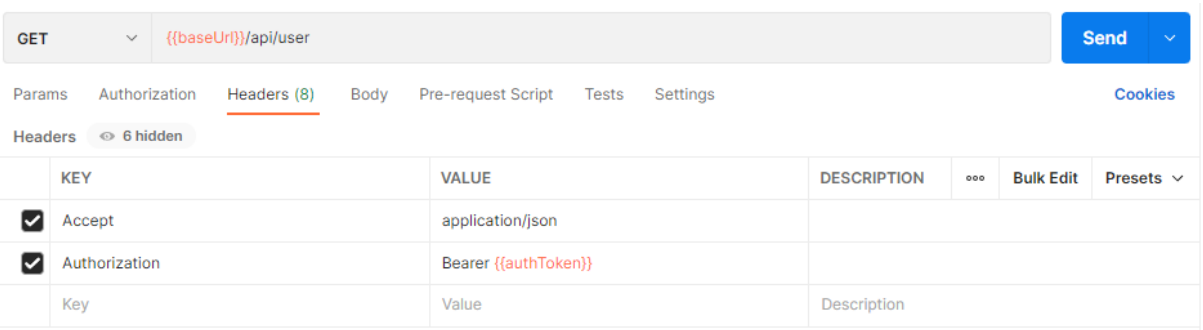


Fig 8: Cabecera de la petición de Información de Usuario

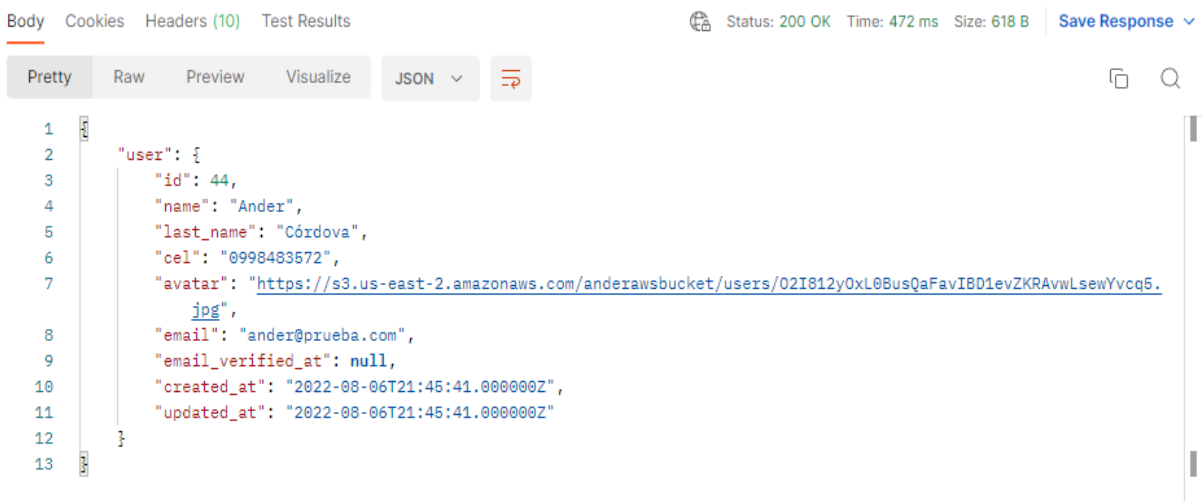


Fig 9: Respuesta a la petición del Información de Usuario

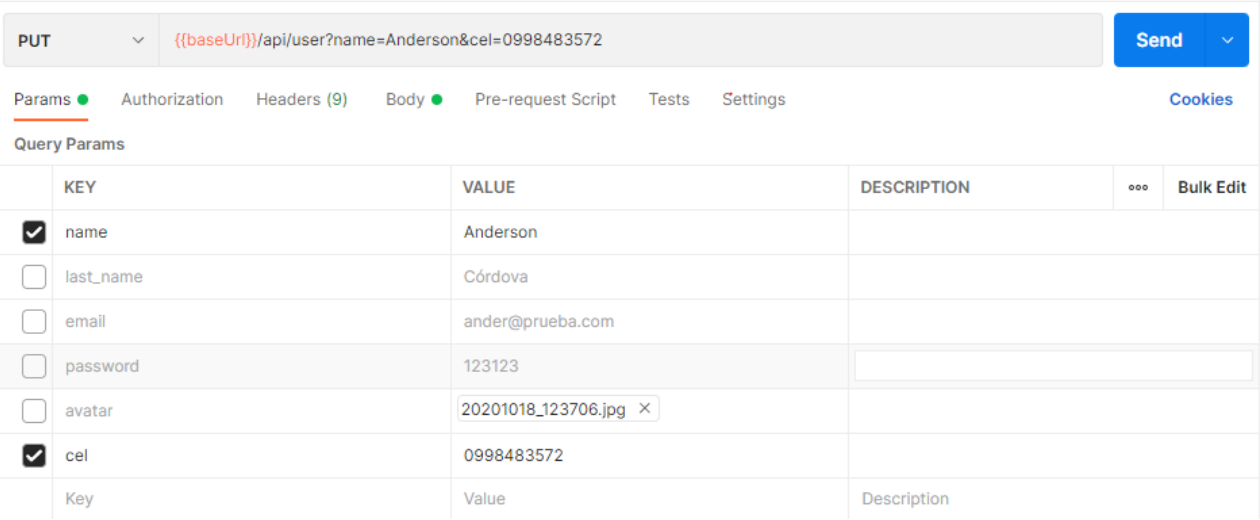


Fig 10: Cabecera de la petición de Actualizar Perfil de Usuario

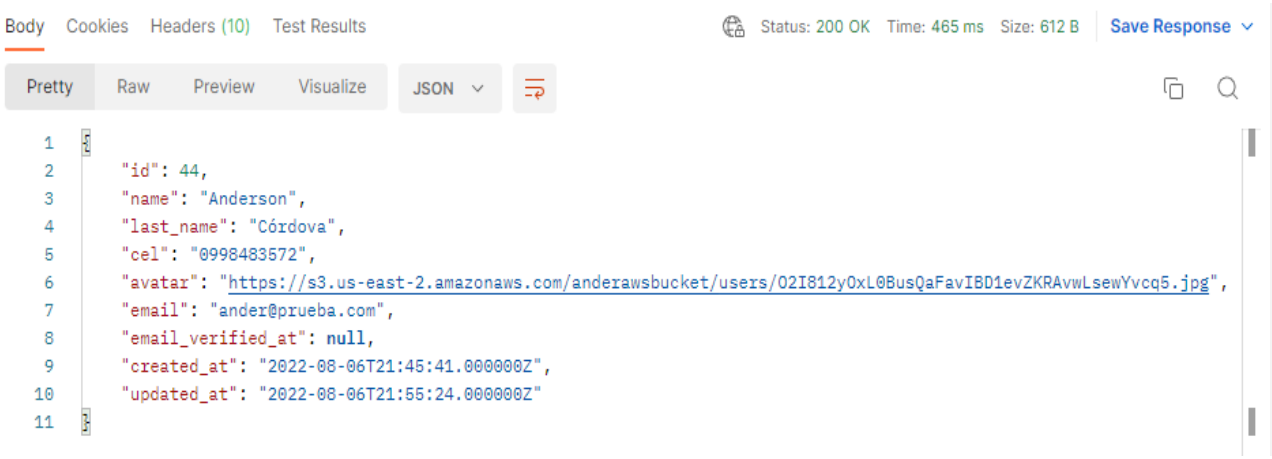


Fig 11: Respuesta a la petición de Actualizar Perfil de Usuario

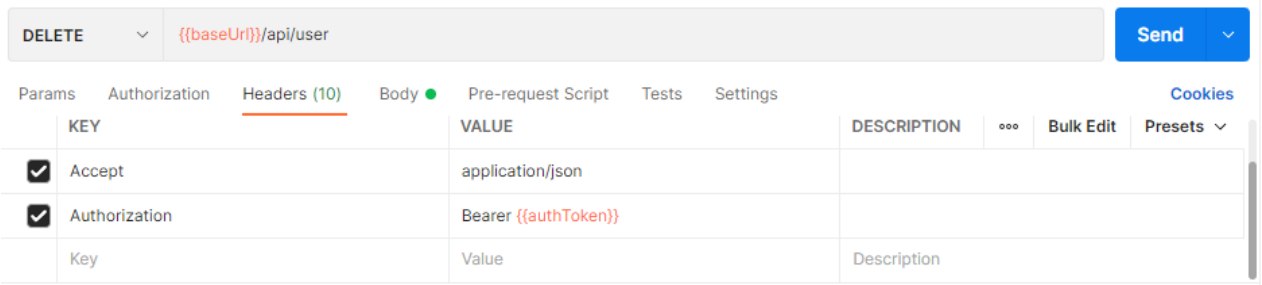


Fig 12: Cabecera de la petición de Eliminar Cuenta de Usuario

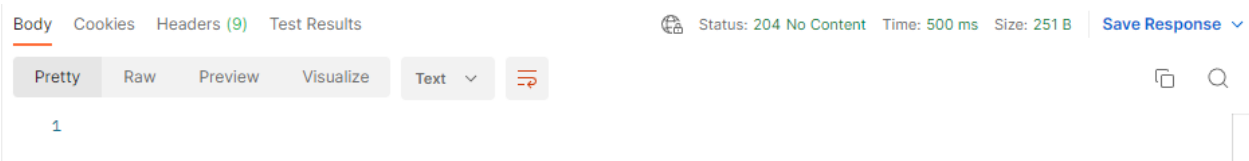


Fig 13: Respuesta a la petición de Eliminar Cuenta de Usuario

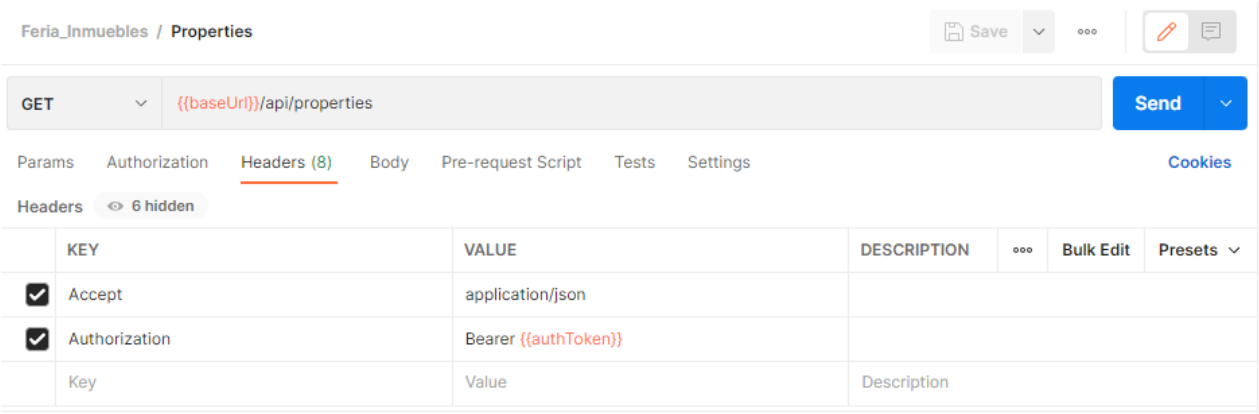


Fig 14: Cabecera de la petición de Lista de Inmuebles

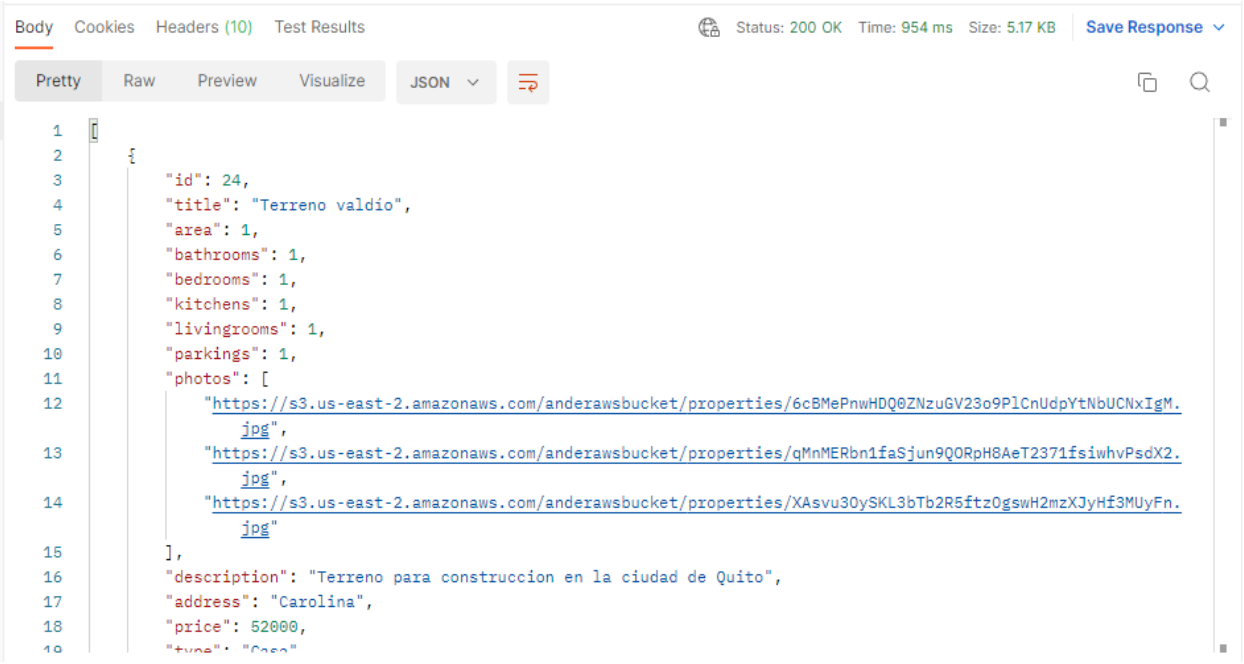


Fig 15: Respuesta a la petición de Lista de Inmuebles

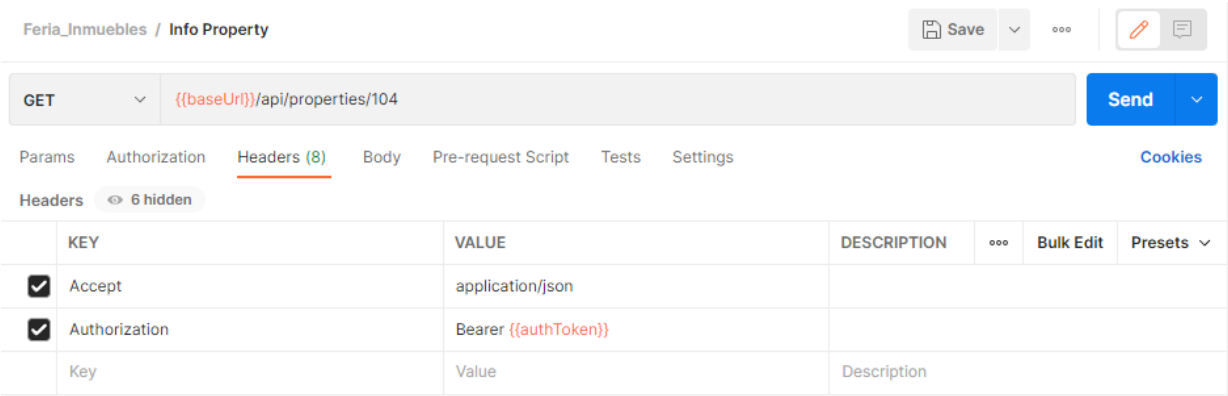


Fig 16: Cabecera de la petición de Información del Inmueble

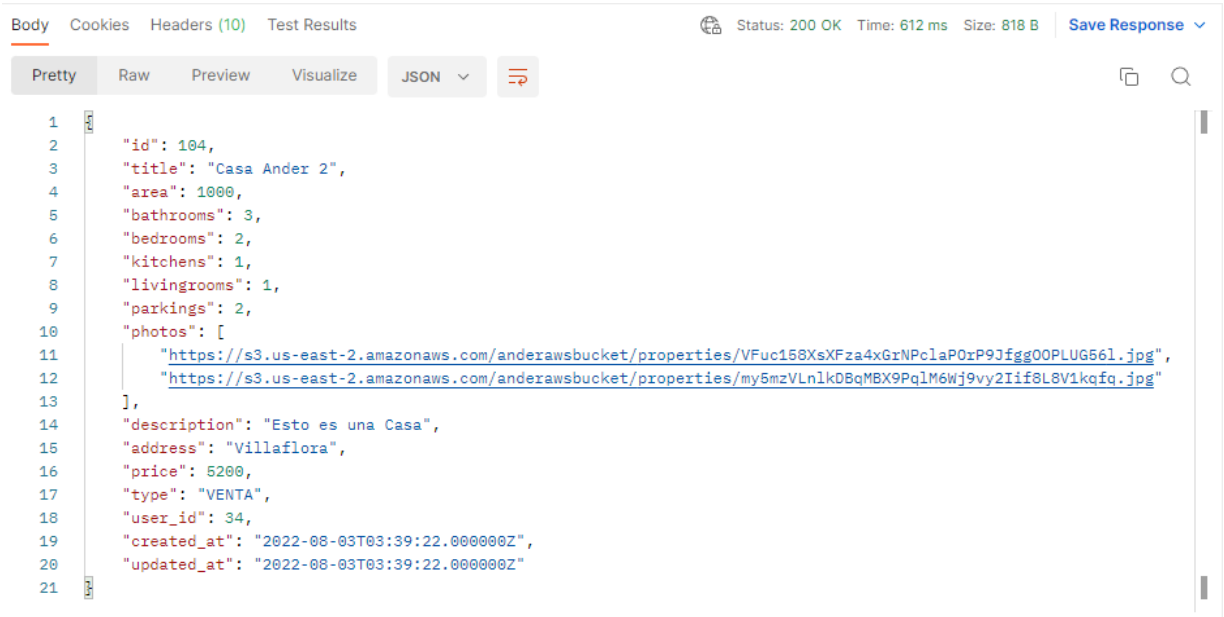


Fig 17:Respuesta a la petición de Información del Inmueble

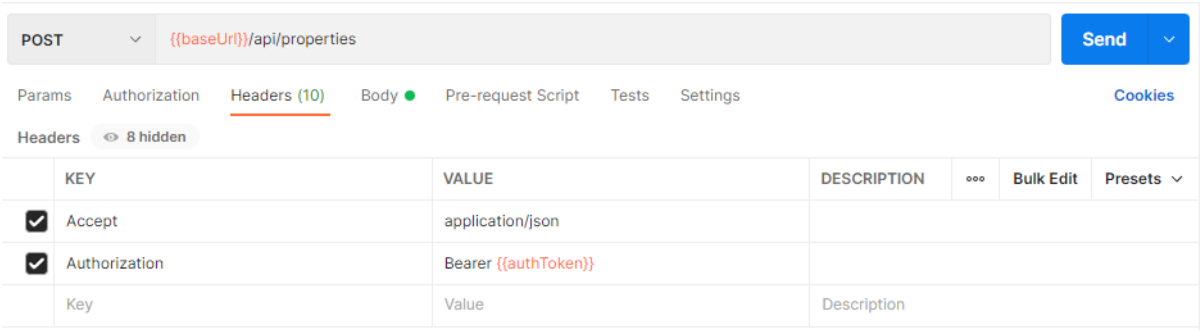


Fig 18: Cabecera de la petición para Publicar Inmueble

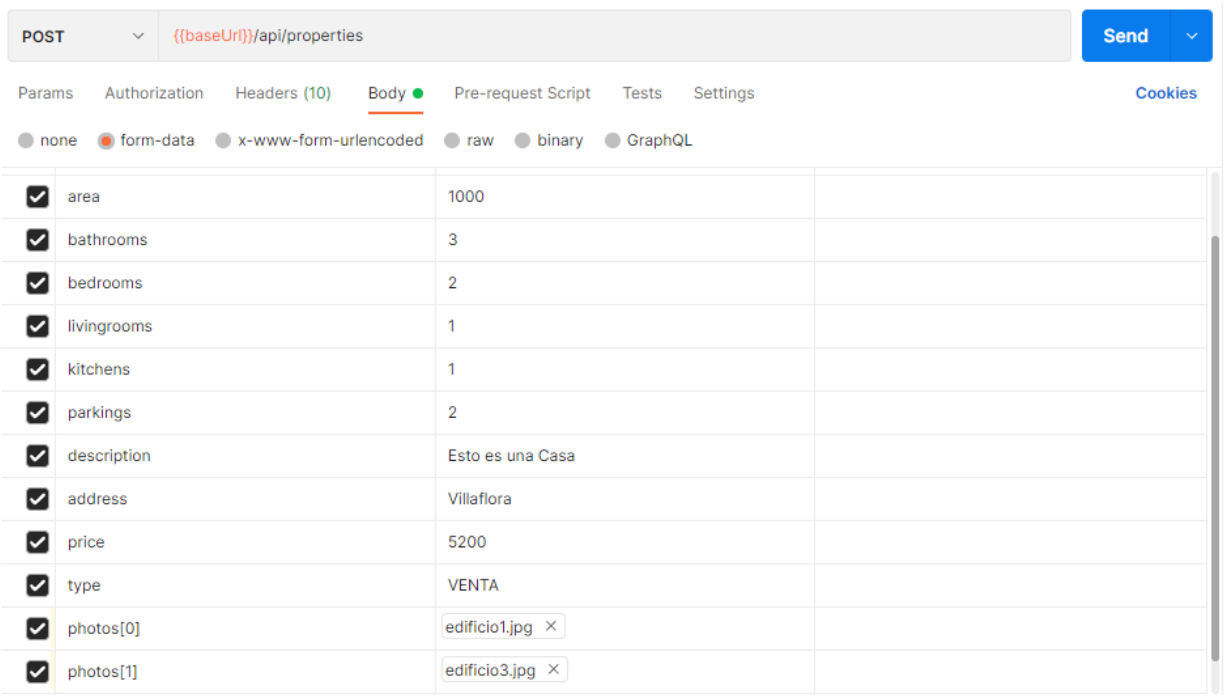


Fig 19: Cuerpo de petición para Publicar Inmueble

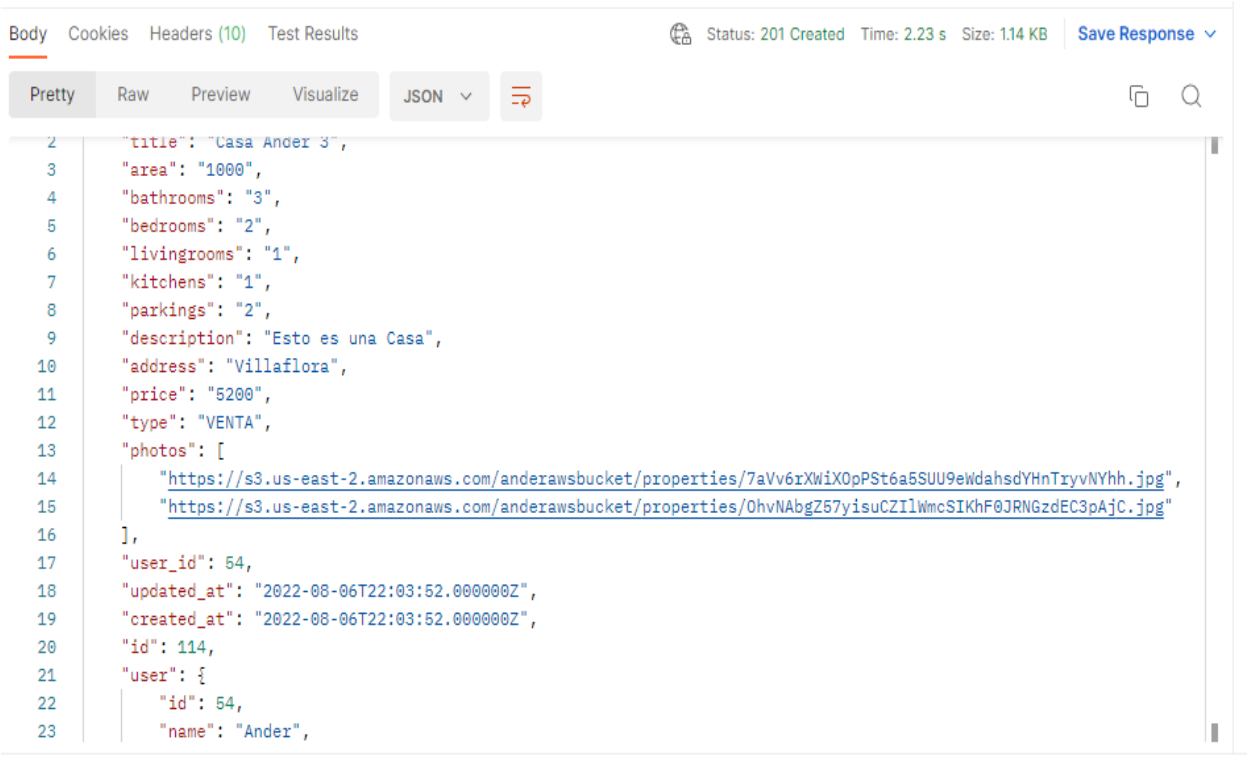


Fig 20: Respuesta a la petición de Publicar Inmueble

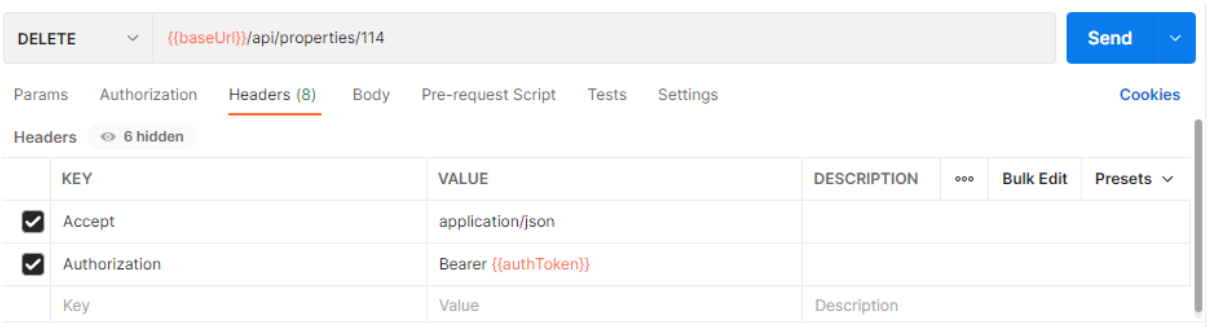


Fig 21:Cabecera de la petición para Eliminar Inmueble

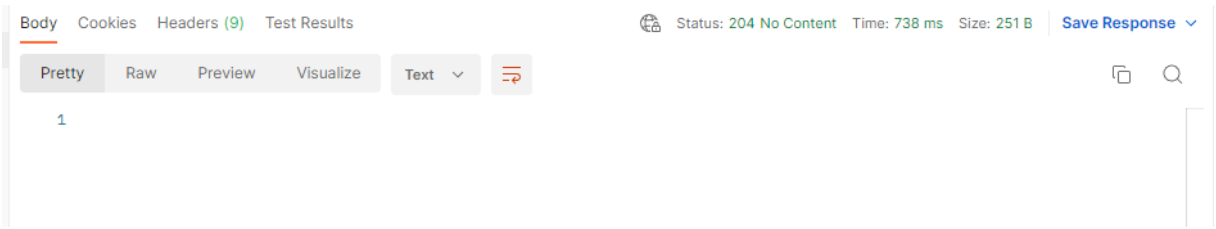


Fig 22: Respuesta a la petición de Eliminar Inmueble

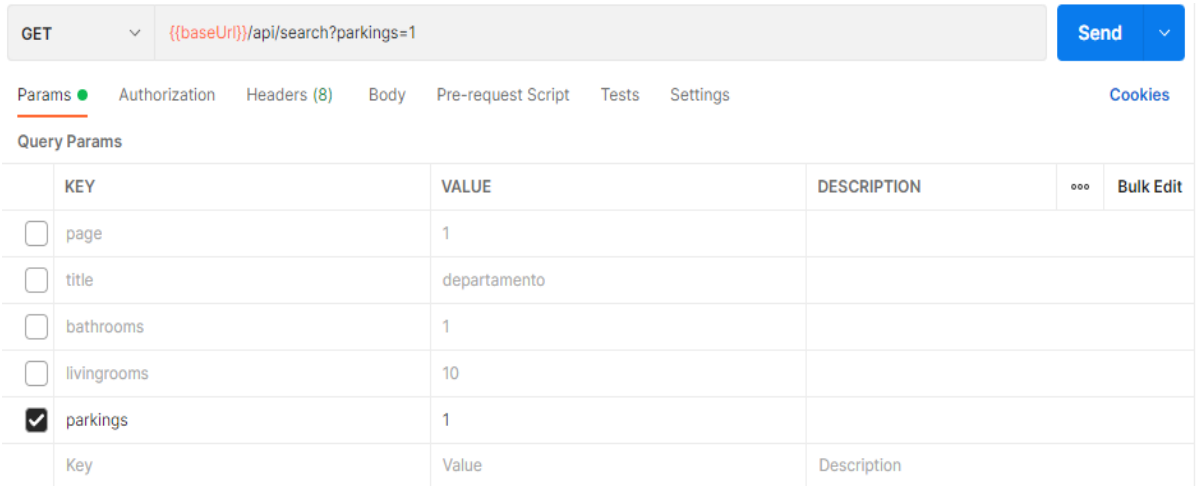


Fig 23: Cabecera de la petición para el Motor de Búsqueda

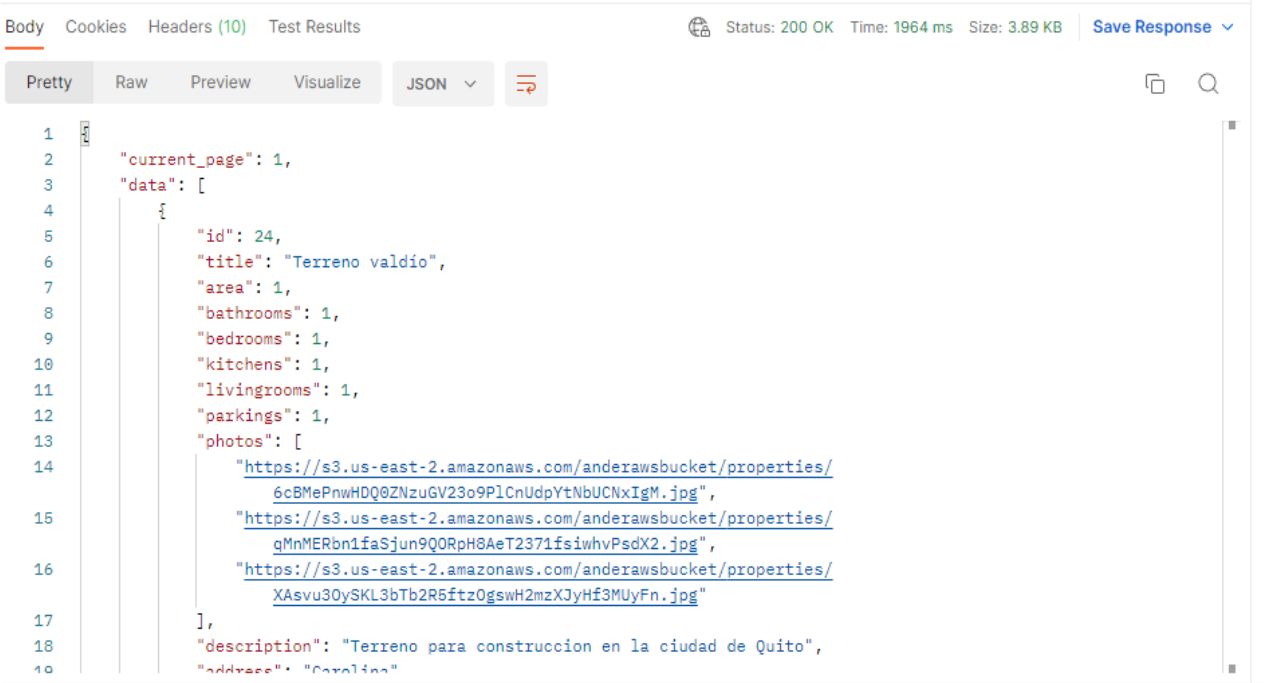


Fig 24: Respuesta a la petición del Motor de Búsqueda


```
        }
    ],
    "next_page_url": "http://api-feria-inmuebles.herokuapp.com/api/search?page=2",
    "path": "http://api-feria-inmuebles.herokuapp.com/api/search",
    "per_page": 5,
    "prev_page_url": null,
    "to": 5,
    "total": 9
}
```


Fig 25: Resultados paginados para la petición del Motor de Búsqueda

3.2. Pruebas de Aceptación

A continuación, se muestran las 10 Pruebas de Aceptación frente al desarrollador *front-end* que guía la implementación de la API.

Tabla XI: Prueba de Aceptación -- Inicio de Sesión

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA001	**Identificador historia de usuario:** HU001
Nombre prueba de aceptación: Inicio de sesión	
Descripción: El desarrollador que consuma de la API debe identificarse en el sistema a través de credenciales de: usuario y contraseña, tras su comprobación, retornar un token de validación o caso contario retornar el error correspondiente.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso con Postman. • Realizar una petición POST a la URL: https://api-feria-inmueble s.herokuapp.com/api/login • Ingresar las credenciales en el cuerpo de la petición	
Resultado deseado: Recibe el token de acceso o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. Aprobación del cliente 100%	

Tabla XII: Prueba de Aceptación -- Registro

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA002	**Identificador historia de usuario:** HU002
Nombre prueba de aceptación: Registro	
Descripción: El equipo de desarrollo que consuma de la API debe llenar un formulario de registro por cada usuario nuevo en el sistema. En el cual se contengan los siguientes campos: nombre, apellido, correo electrónico, celular, contraseña y avatar.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo	

Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición POST a la URL: https://api-feria-inmueble s.herokuapp.com/api/register • Ingresar todos los campos en el cuerpo de la petición. • En el campo de avatar cargar una foto no mayor a 2GB
Resultado deseado: Devuelve información del usuario creado en la base de datos con su token de validación o un mensaje con el error detectado
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%

Tabla XIII: Prueba de Aceptación -- Información del Usuario

	PRUEBA D E ACEPTACION
Identificador de prueba (ID): PA003	**Identificador historia de usuario:** HU003
Nombre prueba de aceptación: Información del usuario	
Descripción: El desarrollador que consuma de la API debe obtener la información individual de cada usuario registrado una vez que este se encuentre con una sesión activa en el sistema.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición GET a la URL: https://api-feria-inmueble s.herokuapp.com/api/user	
Resultado deseado: Devuelve información del usuario creado en la base de datos con su token de validación o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%	

Tabla XIV: Prueba de Aceptación -- Actualizar Perfil de Usuario

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA004	**Identificador historia de usuario:** HU004
Nombre prueba de aceptación: Actualizar perfil del usuario	
Descripción: El desarrollador que consuma de la API deberá ser capaz de actualizar la información de cualquier campo del registro (nombre, apellido, correo electrónico, celular, contraseña y avatar) para cualquier usuario previamente registrado en el sistema.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición PUT a la URL: https://api-feria-inmueble s.herokuapp.com/api/user • Establecer los campos que se deseen actualizar en los parámetros de la petición.	
Resultado deseado: Devuelve información actualizada del usuario creado en la base de datos con su token de validación o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%	

Tabla XV: Prueba de Aceptación -- Eliminar Usuario

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA005	**Identificador historia de usuario:** HU005
Nombre prueba de aceptación: Eliminar Usuario	
Descripción: El desarrollador que consuma de la API debe ser capaz de eliminar cuentas de usuarios	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo • Tener una sesión activa en el sistema.	

Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición DELETE a la URL: https://api-feria-inmueble s.herokuapp.com/api/user
Resultado deseado: Devuelve un estatus 200 para confirmar la eliminación del registro de usuario o un mensaje con el error detectado
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%

Tabla XVI: Prueba de Aceptación -- Publicar Inmueble

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA006	**Identificador historia de usuario:** HU006
Nombre prueba de aceptación: Publicar inmueble	
Descripción: El desarrollador que consuma de la API con un usuario con sesión activa tiene la posibilidad de publicar un inmueble en el sistema a través de un formulario en el que se indique: título, área, número de dormitorios, número de cocinas, número de baños, número de patios, descripción y precio.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición POST a la URL: https://api-feria-inmueble s.herokuapp.com/api/properties • Ingresar todos los campos en el cuerpo de la petición. • Se pueden agregar hasta un arreglo de 5 imágenes por cada publicación de máximo 2GB cada una	
Resultado deseado: Devuelve información de la inmueble creada en la base de datos con información del usuario propietario o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%	

Tabla XVII: Prueba de Aceptación -- Actualizar Inmueble

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA007	**Identificador historia de usuario:** HU007
Nombre prueba de aceptación: Actualizar inmueble	
Descripción: El equipo de desarrollo que consuma de la API con un usuario que ha realizado una publicación de un inmueble, tiene la opción de actualizar la información de esta publicación.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición PUT a la URL: https://api-feria-inmuebles.herokuapp.com/api/properties • Establecer los campos que se deseen actualizar en los parámetros de la petición.	
Resultado deseado: Devuelve información de la inmueble actualizada en la base de datos con información del usuario propietario o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%	

Tabla XVIII: Prueba de Aceptación -- Información Inmueble

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA008	**Identificador historia de usuario:** HU008
Nombre prueba de aceptación: Información inmueble	
Descripción: El desarrollador que consuma de la API debe obtener la información individual de cada publicación de inmueble	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo • Tener sesión activa en el sistema	

Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición GET a la URL: https://api-feria-inmuebles.herokuapp.com/api/properties{property}
Resultado deseado: Devuelve información de la inmueble individual obtenida de la base de datos con información o un mensaje con el error detectado
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%

Tabla XIX: Prueba de Aceptación -- Lista de Inmuebles

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA009	**Identificador historia de usuario:** HU009
Nombre prueba de aceptación: Lista de Inmuebles	
Descripción: El desarrollador que consuma de la API debe obtener un listado paginado de todas las publicaciones de inmueble s existentes en la base de datos	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo • Tener sesión activa en el sistema	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición GET a la URL: https://api-feria-inmuebles.herokuapp.com/api/properties	
Resultado deseado: Devuelve información de las inmueblees existentes obtenidas de la base de datos o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%	

Tabla XX: Prueba de Aceptación -- Eliminar Inmueble

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA0010	**Identificador historia de usuario:** HU0010
Nombre prueba de aceptación: Eliminar Inmueble	
Descripción: El desarrollador que consuma de la API debe ser capaz de eliminar publicaciones de inmueble s hechas por los usuarios en el sistema.	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo • Tener sesión activa en el sistema	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman. • Realizar una petición DELETE a la URL: https://api-feria-inmuebles.herokuapp.com/api/properties/{property}	
Resultado deseado: Devuelve un estatus 200 para confirmar la eliminación del registro del inmueble o un mensaje con el error detectado	
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%	

Tabla XXI: Prueba de Aceptación -- Buscar Inmuebles

PRUEBA DE ACEPTACION	
Identificador de prueba (ID): PA0011	**Identificador historia de usuario:** HU0011
Nombre prueba de aceptación: Buscar Inmuebles	
Descripción: El desarrollador que consuma de la API debe poder filtrar los resultados de las publicaciones de inmueble s a través de una búsqueda en la que el usuario proporcione palabras clave y valores específicos a los campos del inmueble	
Condiciones de prueba: • Tener conexión a internet • Tener software de testeo • Tener sesión activa en el sistema	
Pasos de ejecución: • Abrir nuestra plataforma de testeo, nuestro caso en Postman.	

• Realizar una petición GET a la URL: https://api-feria-inmueble s.herokuapp.com/api/search • Establecer los campos que se deseen actualizar en los parámetros de la petición.
Resultado deseado: Devuelve un listado paginado con los resultados filtrados en base a los parámetros proporcionados por el usuario
Evaluación de la prueba: Se comprobó el resultado esperado. **Aprobación del cliente:** 100%

3.3 Pruebas de Rendimiento

A continuación, se muestran las Pruebas de Rendimiento ejecutadas con el software dotcom-monitor obteniendo las siguientes graficas estadísticas sobre el rendimiento de la APIIREST.

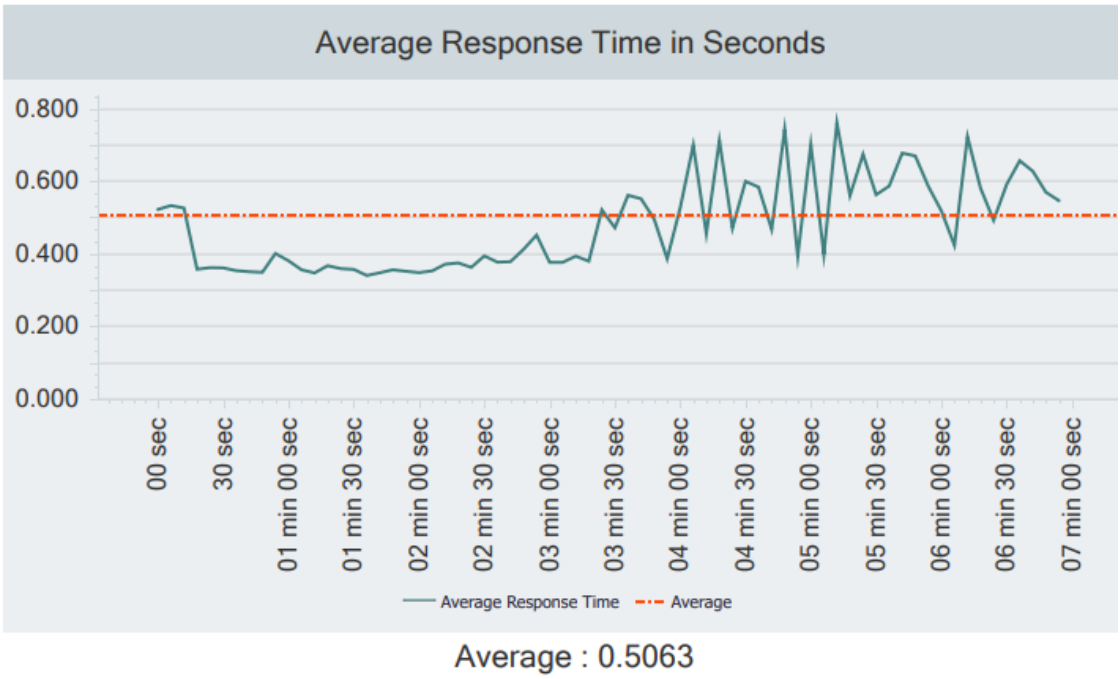


Fig 26: Gráfica del tiempo promedio de respuesta

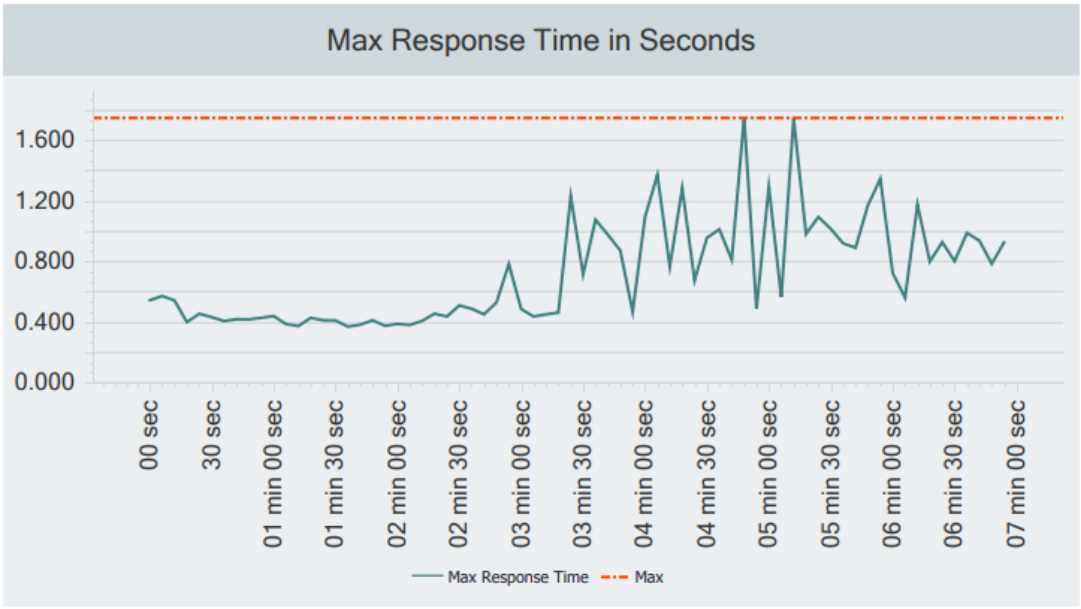


Fig 27: Gráfica del tiempo máximo de respuesta

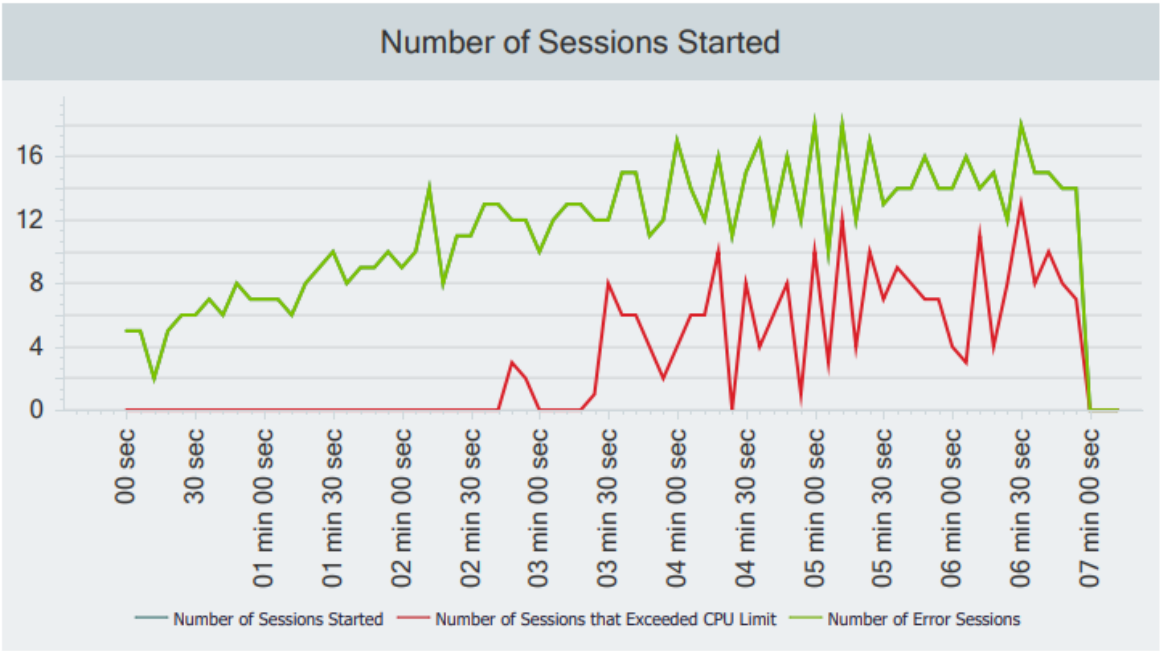


Fig 28: Gráfica del número de sesiones iniciadas y excedidas

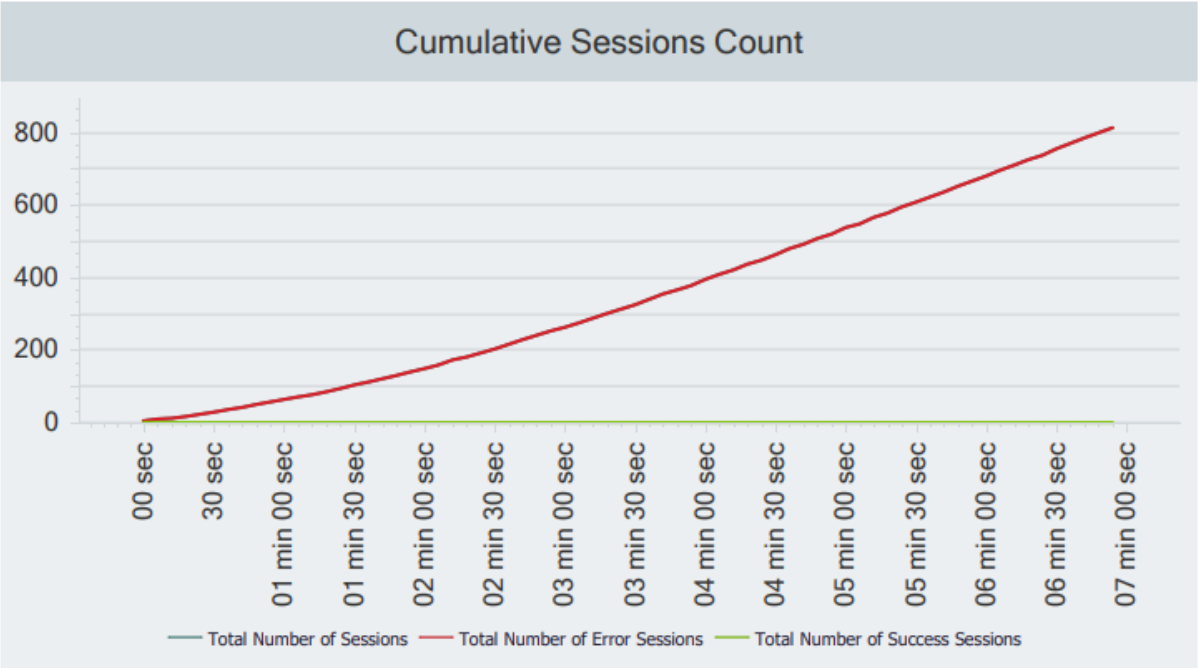
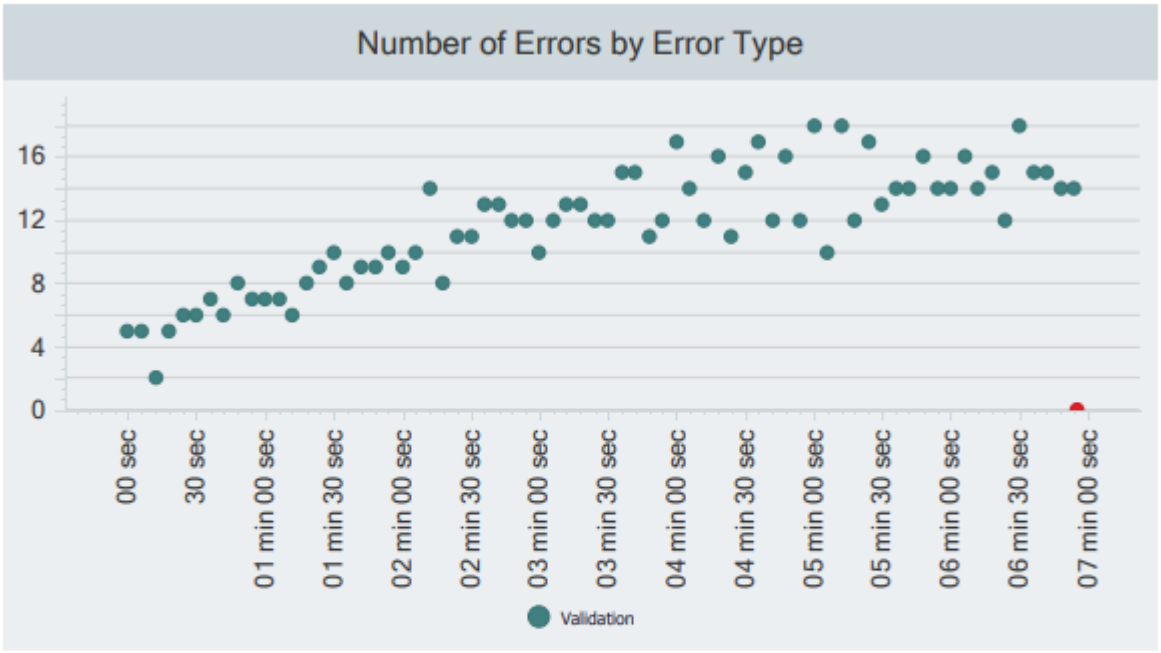


Fig 29: Gráfica de conteo de sesiones acumuladas



Number of Errors by Error Type

Error Type	Number Of Errors
Validation	813
TOTAL	813

Fig 30: Gráfica del número de errores por su tipo

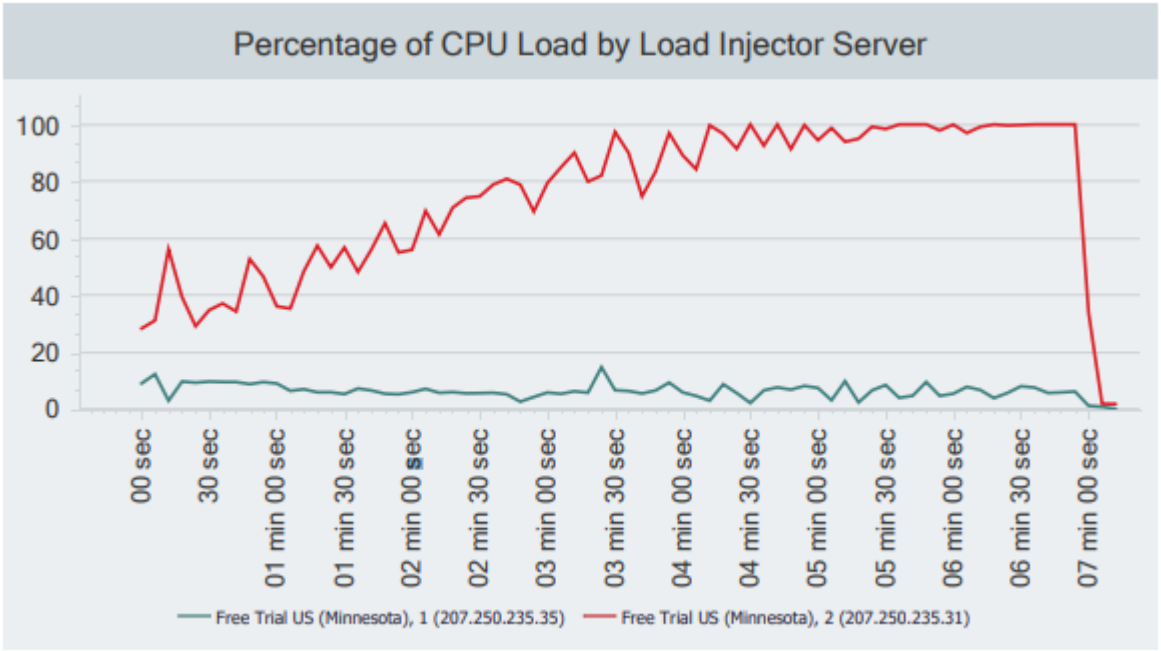


Fig 31: Gráfica del porcentaje de carga del CPU