UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Diseño e Implementación de una Plataforma & API para la Enseñanza de la Tecnología REST.

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA EN TELECOMUNICACIÓN

Autor: Cristian Lorenzo Monción Rodríguez

Director: Fernando Losilla López

Codirector: Juan José Alcaraz Espín

Cartagena, Julio, 2023

Autor	Cristian Lorenzo Monción Rodríguez
E-mail del Autor	**cristianm18@msn.com**
Director	Dr. Fernando Losilla López
E-mail del Director	fernando.losilla@upct.es
Codirector(es)	Dr. Juan José Alcaraz Espín
Título del PFC	Diseño e Implementación de una Plataforma & API para la Enseñanza de la Tecnología REST.
Descriptores	

Resumen

El proyecto que se presenta, muestra el diseño de una plataforma de enseñanza de REST, pensada para permitir a los estudiantes iniciarse en el uso de tecnología para intercambio de datos.

El objetivo es proporcionar una plataforma educativa que pueda enseñar a los alumnos a realizar peticiones a una API REST y entender las respuestas y cabeceras devueltas por la API, permitiéndoles adquirir habilidades prácticas en esta área en constante crecimiento de la programación web. La implementación de la plataforma implica el desarrollo de una aplicación con la que los alumnos podrán listar, crear, modificar y eliminar sus propios recursos individualmente.

Para la realización del proyecto utilizamos tecnologías como Python, SQLAlchemy, HTML, CSS y JavaScript que son herramientas para crear servicios web y son utilizadas por muchas empresas en la industria.

Titulación	Máster Universitario en Ingeniería de Telecomunicación
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Julio, 2023

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, **Fernando Losilla López**, Ha sido un excelente guía en todo momento, instruyéndome y enseñándome nuevas referencias, que han aportado un gran valor añadido a este trabajo.

Gracias a la **Universidad Politécnica de Cartagena**, por darme la oportunidad de haber sido seleccionado para cursar una maestría en sus instalaciones y haber realizado este Trabajo Final de Máster, gracias a la **ETSIT** y todos sus excelentes profesores.

Por último, gracias a mi familia en especial a mis hermanas **Griselda Monción**, **Clara Monción**, mi cuñado **Fernando Petinal**, mi padre **Lorenzo Monción Román**, mis madres **Carlita Vargas** y **Bienvenida Rodríguez**, mi tío **Amable Peña** y amigos que han supuesto en todo momento un apoyo moral muy importante, ayudándome a superar los obstáculos y enfocado el proyecto en el camino correcto.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En la actualidad el intercambio de información o datos mediante servicios web se ha convertido en algo sencillo gracias a la tecnología REST. Esto se debe a que muchas empresas están optando por desarrollar sus productos utilizando la arquitectura API REST (REpresentational State Transfer), ya que permite una comunicación y transferencia de datos simple y fácil de integrar en distintas aplicaciones, así como diferentes plataformas o sistemas. La finalidad de este trabajo es buscar que los alumnos que desean introducirse en el mundo del desarrollo de software y la programación web puedan comprender los conceptos de REST. Esta plataforma de enseñanza se utilizará en la asignatura de Conceptos Avanzados de Internet.

El trabajo propone una plataforma educativa para la enseñanza de API REST desarrollada utilizando el lenguaje de programación Python. El objetivo principal es proporcionar una plataforma de enseñanza para que los alumnos tenga la oportunidad de aprender y practicar.

Esta plataforma se encargará que el profesor enseñe las mejores prácticas a los alumnos sobre la arquitectura REST, además de los conceptos de HTTP y sus verbos GET, POST, PUT, PATCH, DELETE, autenticación de usuario mediante tokens y la representación de datos en formato JSON, que es una de los formatos más populares que existen actualmente para intercambiar información utilizando diferentes tipos tecnologías. En la programación se ha escogido el lenguaje Python y bibliotecas como el Framework Flask y SQLAlchemy para la base de datos.

Esperamos que la plataforma se convierta una herramienta importante para estudiantes, desarrolladores y cualquier persona que desee adquirir habilidades prácticas en el diseño y desarrollo de API REST, así como promover el crecimiento y el aprendizaje de esta tecnología en la programación web.

1.1. Objetivos

Los objetivos principales del proyecto:

- Se pretende aprender una de las principales tecnologías para crear API REST, como lo es el lenguaje de programación Python y algunas de sus bibliotecas que serían Flask, SQLAlchemy y SQLite en el Back-End.
- Profundizar en tecnologías del Font-End como HTML, CSS3, Bootstrap y JavaScript con la creación de un administrador o CMS.
- Entender los conceptos de API REST y desarrollar una aplicación de servicio web que sea capaz de recibir y responder peticiones HTTP y códigos de estado.

Con la herramienta desarrollada en el presente proyecto, los estudiantes aprenderán el uso básico de los servicios web haciendo peticiones cliente / servidor, adquirirán conocimiento sobre el protocolo de comunicaciones HTTP, todo esto con el estilo de arquitectura REST, más utilizado por los desarrolladores y grandes empresas para compartir datos de una manera sencilla y segura.

1.2. Motivación

Mi motivación para realizar este proyecto consiste en que tanto los alumnos como la asignatura de Conceptos Avanzados de Internet puedan contar con una plataforma propia para la enseñanza sin depender de un servicio web de tercero. Más que nada se busca que como una herramienta de enseñanza cumpla con los objetivos que la universidad propone para las generaciones venideras.

2. Tecnologías usadas / Marco teórico

En este apartado se muestran las definiciones de las tecnologías que se utilizó para emplear este proyecto, todas las definiciones son referenciadas con paginas oficiales y reguladoras de estandarización como son W3C, 3WShools, Mozilla etc.

2.1. API (Application Programing Inteface) [1]

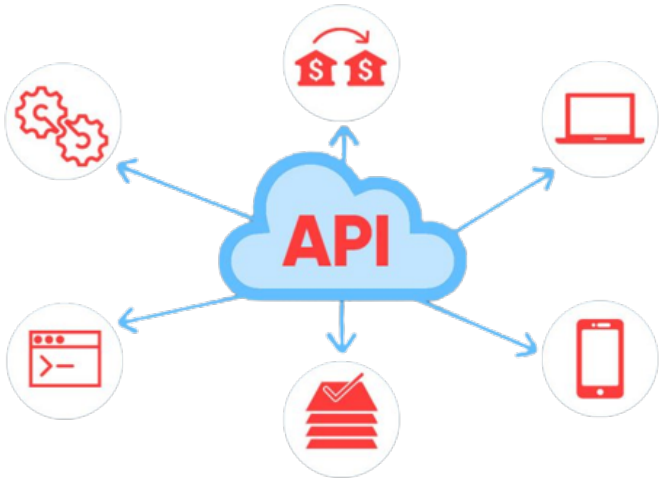


Figura 2.1. *Estructura de una API. [Fig. 1]*

Una API (interfaz de programación de aplicaciones) es una pieza de código que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades. Una API es un intermediario entre dos sistemas, que permite que una aplicación se comunique con otra y pida datos o acciones específicas. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la

programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software.

Las API pueden ser locales y remotas.

- **Locales:** son APIs que tienen cabida o funcionamiento dentro de un mismo entorno o aplicación en que está instalada.
- **Remotas:** son APIs externas que pueden ser consumidas por diferentes tipos de aplicaciones o servicios web. Éstas se comunican a través de internet utilizando un sinfín de variedades de tecnologías para el Back-End como pueden ser Python, NodeJS, PHP, C# entre otras.

Las APIs tienen varias maneras de ser utilizadas. Por ejemplo:

Integración de aplicación o tecnología	Permite que diferentes tipos de aplicaciones de software se comuniquen y compartan datos entre sí.
Móviles y Web	Mediante un Back-End se puede integrarse aplicaciones y servicios.
Creación de complementos de terceros	Las API permiten a desarrolladores de terceros crear complementos para ampliar la funcionalidad de la aplicación o software.
Creación de aplicaciones web	Son utilizadas en el desarrollo de aplicaciones para acceder a recursos y servicios.
Comercio electrónico	Permiten el procesamiento seguro de pagos, el cumplimiento de pedidos y la gestión de inventario.
Mejora la experiencia de los clientes	Por su flexibilidad y rapidez son más atractiva para el cliente.
Mejores para compartir datos	Eficiencia para compartir datos entre diferentes sistemas, como entre los internos y externos de una empresa y sus socios.

Tabla 2.1. *Uso que se le pueden dar a una API*

2.2. REST (REpresentational State Transfer) [2]

Figura 2.2. *Representational State Transfer (REST). [Fig. 2]*

La transferencia de estado representacional (representational state transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por **Roy Fielding**, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

2.2.1. Principios y restricciones de REST

Para que una API pueda ser considerada REST, debe contar con las siguientes características.

1. **Protocolo Cliente/Servidor:** El sistema cliente/servidor ejecuta una API REST, el servidor almacena información y el cliente solicita información a través de la API.
2. **Sin estado (Stateless):** Todas las solicitudes enviadas al servidor deben contener toda la información necesaria para comprender y responder a la solicitud. El servidor no debe almacenar información sobre el estado del cliente entre solicitudes.
3. **Cacheable:** La respuesta del servidor debe indicar si la información se puede almacenar en caché.
4. **Interfaz uniforme (Uniform Interface):** La solicitud y la respuesta deben ser coherentes para facilitar el uso de la API. Este principio incluye varias subreglas, incluida la identificación de recursos, la manipulación de recursos a través de la representación y la autodescripción del mensaje.

5. **HATEOAS (Hypermedia As The Engine Of Application State)**: En el contexto de REST, este principio se refiere a la capacidad de la API para proporcionar hipervínculos o controles en respuesta a las solicitudes de los clientes. Estas comprobaciones informan al cliente de los próximos pasos apropiados, sin conocimiento previo de la estructura o el enrutamiento de la API.

6. **Documentar:** Que tenga una buena documentación y muestre la forma de cómo utilizarla es un punto de gran importancia.

2.2.2. Métodos HTTP en REST

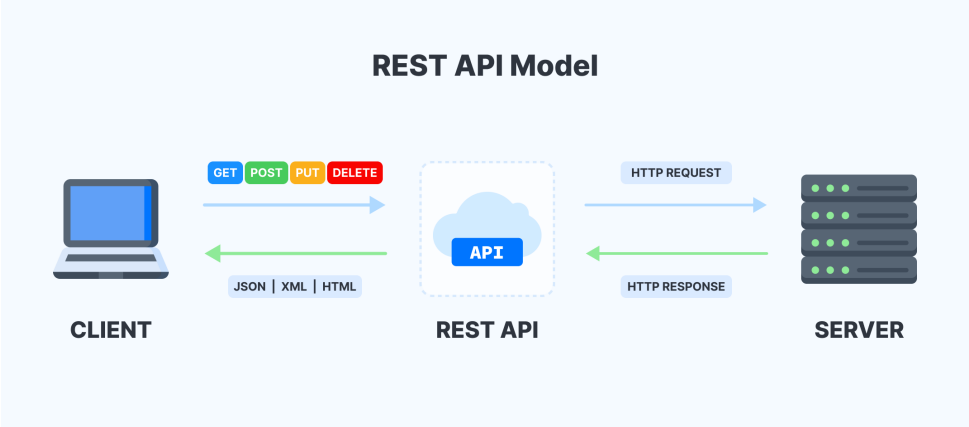


Figura 2.3. *Métodos HTTP en la web. [Fig. 3]*

A través de solicitudes HTTP utilizando los siguientes métodos entre un cliente y un servidor que realizan peticiones.

- **GET:** se utiliza para recuperar una lista de recursos, estos pueden estar almacenados en una base de datos o en archivos con extensiones JSON, XML, txt, etc. Al hacer una petición (GET) al servidor, esta devuelve un código de estado 200 (OK), una respuesta típica en caso de éxito.

- **POST:** se utiliza para crear un nuevo recurso en una lista de recursos, devuelve como respuesta un código de estado 201 (CREATED). Aparte de tener este funcionamiento en REST (POST) puede utilizarse para otras operaciones.

- **PUT:** se utiliza para modificar y/o actualizar un recurso existente en una lista de recursos, el servidor devuelve como respuesta el código de estado 200 (OK), una respuesta típica en caso de éxito.

- **PATCH:** Actualiza un recurso en campo específicos.
- **DELETE:** Elimina un recurso existente.

2.2.3. Códigos de estados HTTP (Status Code)

Figura 2.4. Códigos de estado HTTP en la web

Los códigos de estado representan el tipo de respuesta que devuelve el servidor a la hora de hacer una petición HTTP.

- **100-199:** significa que el servidor esta retornando un código de respuesta informativas.
- **200-299:** el servidor retorna satisfactorias la peticiones.
- **300-399:** respuesta de redireccionamientos.
- **400-499:** Error en la petición del cliente.
- **500-599:** Errores del servidor.

2.2.4. Códigos más comunes que aparecerán en la aplicación

A continuación, se muestran los códigos de estados más comunes que pueden encontrarse al utilizar un sitio web.

Códigos de estado 2xx (Exitosos)	
200 (OK)	La petición ha sido procesada satisfactoriamente.
201 (Created)	Se creó un nuevo recurso correctamente.
202 (Accepted)	Indica que la solicitud se ha recibido, pero aún no se ha completado.
204 (No Content)	El servidor ha cumplido con la solicitud, pero no necesita devolver un cuerpo de respuesta.
Códigos de estado 3xx (Redireccionamiento)	
301 (Moved Permanently)	La URL del recurso solicitado se ha cambiado de forma permanente.
302 (Found)	La URL del recurso solicitado se ha cambiado temporalmente.
304 (Not Modified)	Indica al cliente que la respuesta no se ha modificado, y el cliente puede usar la misma versión almacenada en la caché.
Códigos de estado 4xx (Error del cliente)	
400 (Bad Request)	Faltan datos en la solicitud.
401 (Unauthorized)	El dato subministrado es inválido.
403 (Forbidden)	No cuenta con el permiso para acceder al sistema.
404 (Not found)	No se encuentra el recurso.
Códigos de estado 5xx (Error del servidor)	
500 (Internal server error)	No se completó la petición por problema en el servidor.
501 (No Implementd)	El método HTTP no es compatible con el servidor
505 HTTP Version Not Supported (Experimental)	El servidor no admite la versión HTTP utilizada en la solicitud.

Tabla 2.2. Códigos de estados

2.2.5. JSON (JavaScript Object Notation)[3]

Figura 2.5. *logo JSON. [Fig. 4]*

Es un formato para transferir datos con una estructura sencilla construida pares clave – valor, este es muy parecido a la sintaxis de objeto JavaScript. JSON formato muy extendido para el intercambio de datos, se han desarrollado API para distintos lenguajes (por ejemplo ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi, E, Eiffel, Java, JavaScript, ML, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Lua y Visual FoxPro) que permiten analizar sintácticamente, generar, transformar y procesar este tipo de dato. Un objeto es un conjunto desordenado de pares que tiene propiedades asociadas en él. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa).

En la siguiente figura se muestra la estructura de un objeto.

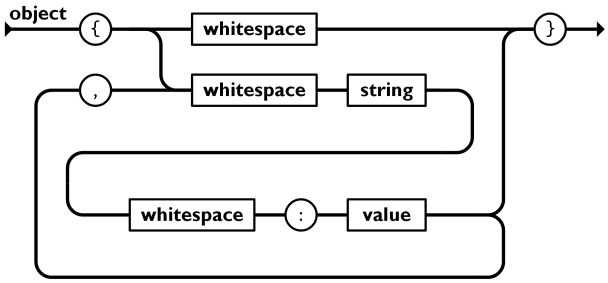


Figura 2.6. *Estructura de un objeto **JSON** [Fig. 5]*

El nombre del objeto puede ser cualquier texto que cumpla el estándar y sus propiedades se identifican porque están entre comillas y están separadas con dos puntos (:).

```
var People = {
    "name" : "John",
    "email" : "john@email.com",
        age : 28
};
```

Figura 2.7. *Código de un objeto JSON*

En este ejemplo, tenemos un objeto JSON que representa información sobre una persona. La estructura de este objeto se detalla a continuación:

- El valor de la clave "name" es "John", que es una cadena.
- El valor de la clave "email" es "john@email.com", que es una cadena.
- El valor de la clave "age" es 28, que es un número entero.


```
{
  "data": [
    {
      "email": "carlasalinas@example.com",
      "first_name": "Martina",
      "id": 1,
      "last_name": "Ferrando"
    },
    {
      "email": "ruperto55@example.com",
      "first_name": "Paz",
      "id": 2,
      "last_name": "Madrid"
    },
    {
      "email": "amaya60@example.com",
      "first_name": "Victor Manuel",
      "id": 3,
      "last_name": "Perea"
    },
    {
      "email": "afiguerola@example.org",
      "first_name": "Eduardo",
      "id": 4,
      "last_name": "Hidalgo"
    }
  ]
}
```


Figura 2.8. *Código de un objeto JSON validado*

2.3. FRONT END

2.3.1. HTML (HyperText Markup Language) [4]

Figura 2.9. *Lenguaje de marcado HTML. [Fig. 6]*

Como sus siglas lo describen **Hyper Text Markup Language** es un lenguaje de marcado que es renderizado por un navegador web. Fue publicado en el año 1993 con su versión 1.1, con el pasar de los años fueron lanzando nuevas versiones siendo una de la más destacadas a nivel de tecnología HTML 4.0. En la 5 versión de HTML publicada en el 2014 la World Wide Web la especificó como definitiva del estándar estando la W3C detrás del desarrollo y regularización.

Figura 2.10. *Lenguaje de marcado v5 HTML5. [Fig. 7]*

HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas ** ** y ****, pero tienen un significado semántico, como por ejemplo **<nav>** (bloque de navegación del sitio web) y **<footer>**, que facilitan el desarrollo de los sitios web de forma responsiva y sencilla.

Junto a las características mencionadas anteriormente no podemos olvidar que la evolución que ha tenido esta tecnología es gracias que, junto a CSS y JavaScript, haciendo que el contenido multimedia, gráficos y animaciones tengan una mejor aspecto de visualización .

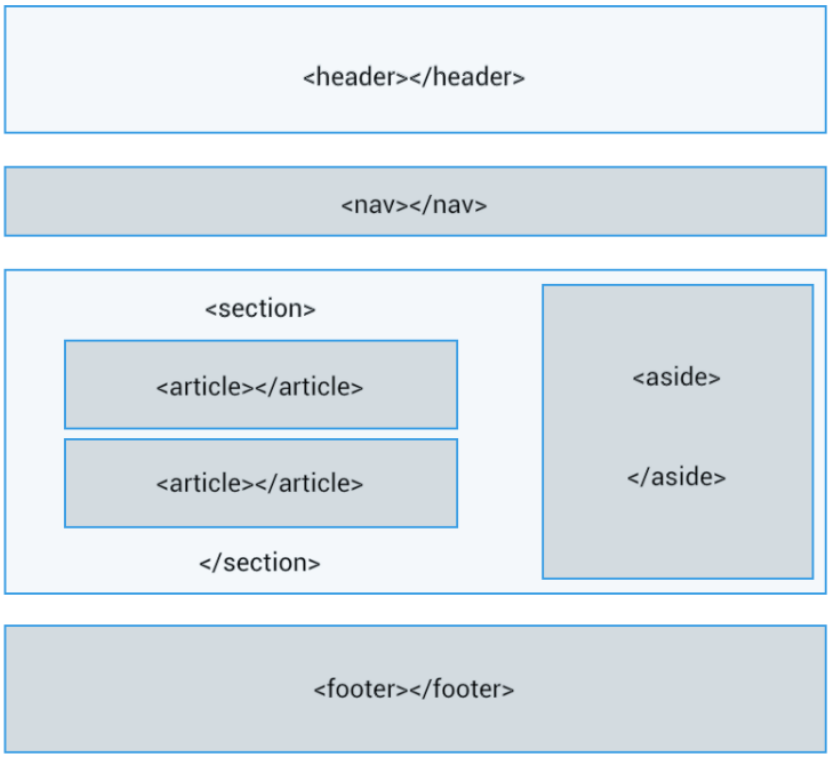


Figura 2.11. *Semántica para crear un estándar de html5 responsive. [Fig. 8]*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML5 Semántica</title>
</head>
<body>
    <header>Header Here</header>

    <nav>Nav Here</nav>
    <section>
        <article>Article Here</article>
        <aside>Aside Here</aside>
    </section>
    <footer>Footer Here</footer>
</body>
</html>
```

Figura 2.12. *Etiquetas html5 en un archivo con extensión .html*

2.3.2. CSS3 (Cascading Style Sheets) [5]

Figura 2.13. *Cascading Style Sheets CSS3. [Fig. 9]*

CSS es un lenguaje de diseño que es utilizado para darle vida a una página web creada con un lenguaje de marcado en este caso HTML, CSS define reglas de estilo refiriéndose a un elemento o tag (etiqueta html) para modificar sus atributos, esto puede ser el color, el tamaño, posición etc. CSS se maneja con selectores que pueden ser desde una etiqueta, así como incluir atributos como (cases o id) que determinan en un código html.

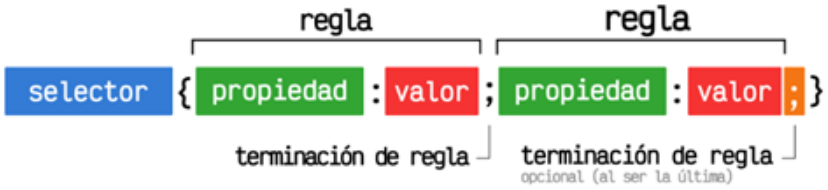


Figura 2.14. *Estructura de CSS. [Fig. 10]*

CSS3 es la última versión, esta proporciona una amplia gama de características y mejoras. Los desarrolladores pueden crear páginas web con diseños agradable y con movimientos solo con el lenguaje, utilizando selectores avanzados, efectos de transición y animación, flexibilidad adaptable con diseño responsivos para distintos tamaños de dispositivos. Ejemplo de código en CSS.

```css
* {
    margin: 0;
    padding: 0;
    font-family: sans-serif;
}
.container {
    max-width: 1300px;
    margin: auto;
}
```

Figura 2.15. *Código CSS3.*

2.3.3. JavaScript (JS) [6]

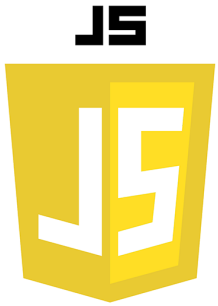


Figura 2.16. *Logo JavaScript. [Fig. 11]*

Es un lenguaje de programación desarrollado a mediado de los años 90, es un lenguaje de scripting que se ejecura del lado del cliente y se ejecuta en el navegador web. Se ha convertido en los últimos años en uno de los lenguajes más populares en la web por su interactividad en tiempo real en solicitudes HTTP con AJAX, es el único lenguaje de programación que interpretan los navegadores web. JavaScript tiene implementado el **Document Object Model (DOM)** que es una interfaz que permite interactuar puede decirse acceder, añadir y cambiar dinámicamente la estructura de un documento HTML en la web..

JavaScript es un lenguaje de secuencias de comandos versátil que admite tanto la programación imperativa como la funcional. Además de su uso en el desarrollo web, JavaScript también se aplica al desarrollo de aplicaciones de servidor (Node.js), aplicaciones móviles híbridas (que usan marcos como React Native e Ionic), desarrollo de juegos y más. Ejemplo de código.

```
// Carga la función listar al iniciar
window.addEventListener('load', function(){
    listApi(1);
})

// Variables de los elementos
let idRandom = Math.floor(Math.random() * 3500) + 1600;
let labelUrl = document.getElementById('labelUrl')
let labelStatus = document.getElementById('labelStatus')
let request = document.getElementById('req')
let response = document.getElementById('resp')
let hiddenDiv = document.getElementById('hd')

// DateTime
const updateTime = () => {
    const datetime = new Date()
    return datetime.toISOString()
}
```

Figura 2.17. *Código CSS3.*

2.3.4. Bootstrap [7]

Figura 2.18. *Logo Bootstrap. [Fig. 12]*

Es un framework CSS de desarrollo de código abierto para diseño de web, hasta ahora es el marco de trabajo más popular que juntando HTML, CSS y JavaScript se pueden crear sitio web con gran calidad visual y profesional. Esta herramienta creada por Twitter 2010 puede hacer que los desarrolladores de Front-End construyan aplicaciones web adaptables a diferentes dispositivos con una gran facilidad y hacen que se ajusten dinámicamente a dispositivos como ordenadores, tables y móviles.

En este proyecto se implementó Bootstrap en la parte de administración utilizando una plantilla SB Admin2 Free. Esta moderna plantilla creada con Bootstrap 4 su versión lite es gratuita y cuenta con una versión Pro que es de pago y cuenta con una mayor funcionabilidad.

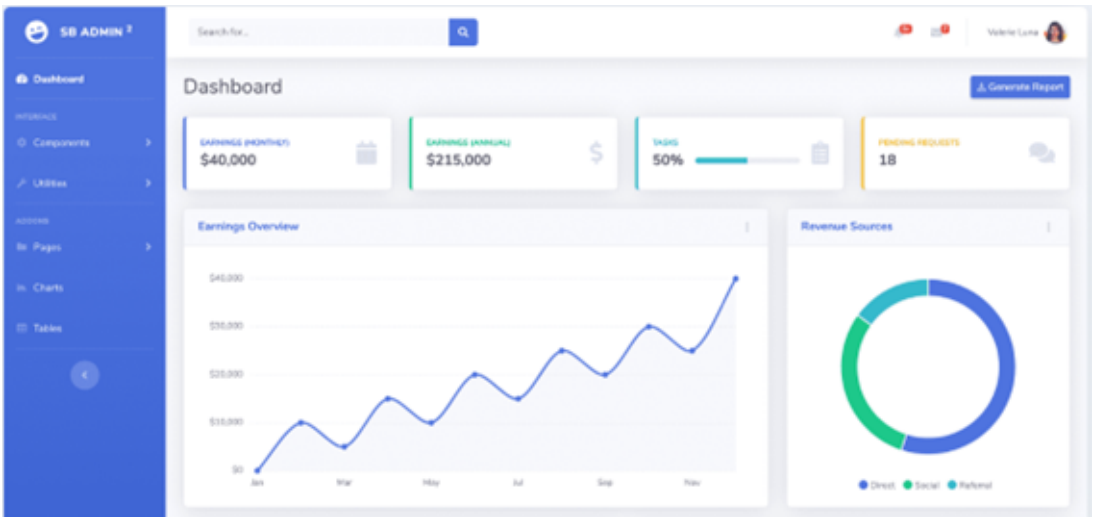


Figura 2.19. *Plantilla Admin Bootstrap.*

2.4. BACK END

2.4.1. Python [8]

Figura 2.20. *Logo Python. [Fig. 13]*

Todos conocemos el lenguaje de programación más popular a nivel mundial **Python**, como un lenguaje de alto nivel es uno de lo más utilizado en de desarrollo de aplicaciones ya que es dinámico, interpretado y multiplataforma. Por su legibilidad se encuentra presente en innumerables servicios web de renombre como son Facebook, Netflix, Spotify, Instagram, etc. Así también empresas como Google, Microsoft, Amazon incorporan este popular lenguaje en algunos de sus proyectos.

Este lenguaje de programación cuenta con bastantes librerías y frameworks para desarrollar aplicaciones de todo tipo. En este caso utilizaremos el framework de Python llamado Flask. Toda la codificación se realizó en Python, es un lenguaje sencillo y con todas sus librerías hace que construir un servicio web sea bastante fácil. Según encuestas realizadas en el 2022 por distintas páginas web Python sigue colocado entre los primeros cincos lenguajes mejor valorados indica Stack Overflow.

2.4.2. Flask (Framework for Python) [9]

Figura 2.21. *Logo framework* ***Flask****. [Fig. 14]*

Es un framework de Python que con pocas líneas de código se puede montar un servidor web y crear aplicaciones de manera fácil. Este incluye módulos de Seguridad y Autenticación, Bases de Datos, Control de Rutas, etc. así como un motor de plantilla llamado Jinja2 que facilita la integración de código Python en nuestras plantillas HTML. Portales web que utilizan este framework podemos encontrarnos con Pinterest, LinkedIn entre otros.

A continuación, se muestra la URL e información donde está corriendo el servidor creado con Flask, también facilita un modo Debug que muestra los errores que puede arrojar el código de una aplicación. Con Flask se realizó toda la lógica de programación.

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 739-225-998
```

Figura 2.22. *Salida en consola servidor en* ***Flask****.*

2.4.3. SQLAlchemy (ORM) [10]

Figura 2.23. *ORM* ***SQLAlchemy****. [Fig. 15]*

Es una herramienta SQL de Python que utiliza la técnica mapeo de objeto relacional (Object-Relational mapping), esto facilita a los desarrolladores diseñar bases de datos utilizando un lenguaje de pro-

gramación orientado a objetos. Las bases de datos diseñadas con ORM son eficientes y de alto rendimiento, con SQLAlchemy en conjunto con SQLite se realiza el diseño para almacenar los datos en un archivo con extensión .db.

2.5. Herramientas de desarrollo utilizadas

Este apartado cuenta con el uso de las herramientas que tuvieron participación para llevar a cabo este proyecto, entre ellas destacan.

2.5.1. Visual Studio Code [11]

Figura 2.24. *IDE VSC. [Fig. 16]*

Es un editor de código fuente creado por Microsoft para varias plataformas, fue lanzando en año 2015 y desde entonces ha sido uno de los mejores editores preferido para los desarrolladores, en el podemos trabajar con una gran parte de las tecnologías que implementa Microsoft como son C, C++, C#, .NET, así también una gran variedad de lenguajes como Python, Java PHP, JavaScript, Go, CSS ect. Este también cuenta con innumerables extensiones que ayudan al desarrollo ágil.

Visual Studio Code se basa en Electron, un framework que se utiliza para implementar Chromium y Node.js como aplicaciones para escritorio, que se ejecuta en el motor de diseño Blink.

2.5.2. DB Browser for SQLite [12]

Figura 2.25. *DB Browser for SQLite. [Fig. 17]*

En una herramienta utilizada para crear, diseñar y editar bases de datos que sean compatibles con SQLite, este software cuenta con una interfaz gráfica intuitiva y fácil de manejar. Es simple para y buena para llevar pequeños proyectos y en el aprendizaje de SQL.

Este programa fue desarrollado originalmente por Mauricio Piacentini (@piacentini) de Tabuleiro Producoes, como Arca Database Browser. La versión original se usó como una herramienta complementaria gratuita para Arca Database Xtra, un producto comercial que incorpora bases de datos SQLite con algunas extensiones adicionales para manejar datos comprimidos y binarios.

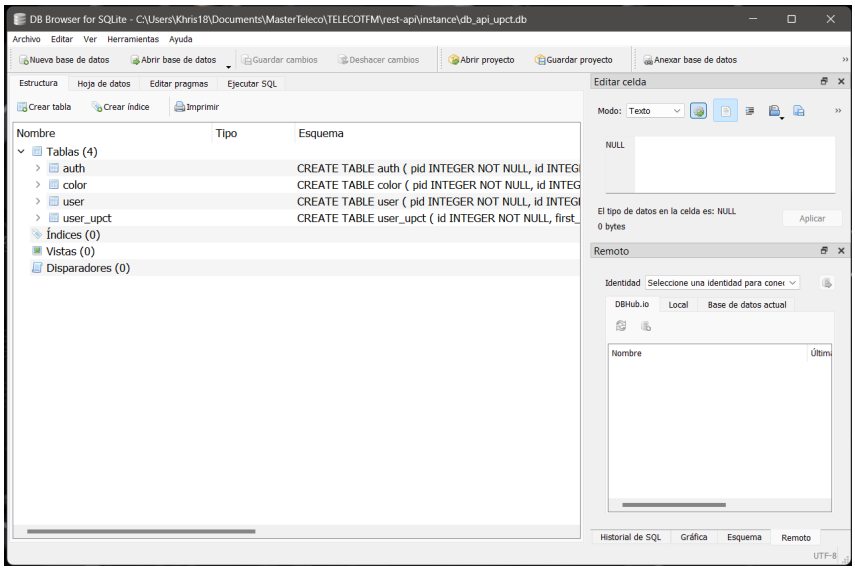


Figura 2.26. *Logo DB Browser SQLite interfaz.*

2.5.3. Postman Client [13]

Figura 2.27. *Logo Postman Client. [Fig. 18]*

Postman es la plataforma de API líder en el mundo. Las funciones de Postman simplifican cada paso de la creación de una API y agilizan la colaboración para ayudar a crear mejores API, más rápido. Lanzada en el año 2012, Postman es un cliente para testear APIs, por el momento es más utilizada para realizar peticiones API REST de manera simple, esta gestiona nuestra API, así como documentarla.

- **Repositorio de API:** permite a los usuarios almacenar, catalogar y colaborar en torno a artefactos de API en una plataforma central dentro de redes públicas, privadas o de socios .
- **Creador de API:** ayuda a implementar un flujo de trabajo de diseño de API a través de especificaciones que incluyen OpenAPI , GraphQL y RAML . Integra diversos controles de fuente, CI/CD , puertas de enlace y soluciones APM.
- **Herramientas:** cliente de API, diseño de API , documentación de API , pruebas de API, servidores simulados y detección de API.
- **Inteligencia:** advertencias de seguridad, búsqueda en el repositorio de API, espacios de trabajo, informes, gobierno de API.
- **Espacios de trabajo:** los espacios de trabajo personales, de equipo, de socios y públicos permiten a los desarrolladores colaborar interna y externamente.

3. Diseño de la aplicación

El objetivo de ese trabajo es proveer a los estudiantes de la UPCT y la asignatura de **Conceptos Avanzados de Internet** de una plataforma que pueda tener la facilidad de aprendizaje sobre API REST, con esto el profesor encargado podrá compartir sus conocimientos a través de esta plataforma de enseñanza.

3.1. ¿Qué se busca con esto?

Anteriormente la asignatura **Conceptos Avanzado de Internet** utilizaba API REST en servidores externos a la UPCT. Con estos Con estos servicios web externos se impartían bien las clases, pero con algunos defectos que se han corregido en ésta. Al crear la API REST principal se buscaba una manera de como el estudiante podría interactuar con sus propios datos sin que otro pueda visualizarlos, editarlos o eliminarlos sin consentimiento. Para esto se tomó la idea de que cada alumno se le asigne un número de usuario en la URL, este describe que datos pertenecen a cada usuario y que esto solo los recursos o registros que tengan ese número les aparezcan para él.

Para llevar a cabo este parte tan importante en el desarrollo del sistema, se tomó la idea de la página web reqres.in que se utiliza para impartir la práctica uno de **Conceptos Avanzado de Internet** (**CAI**). El servicio web que se usaba como aprendizaje, se encuentra en esta URL https://reqres.in, esta plataforma contaba con algunos problemas que hacían que las tareas propuestas por la práctica uno no se realizasen correctamente.

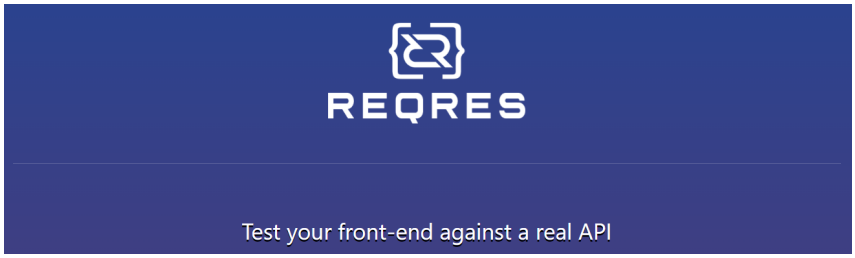


Figura 3.1. *Servicio web para aprendizaje REST. [Fig. 19]*

3.1.1. Problemas del servidor externo utilizado

Los problemas de reqres.com se detallan a continuación.

1. Al utilizar un servidor web externo, la disponibilidad de la práctica depende de la disponibilidad del servidor. Por lo que si el servidor no esta disponible en el momento de realizar esa práctica de seguro conllevaría un problema grave.
2. La plataforma API REST de reqres.in utiliza datos que son constantes para uso general, lo que significa que estos datos son siempre permanentes para todos los usuarios.
3. Un usuario que realice peticiones como crear, modificar y eliminar, al hacer un GET al recurso mediante su identificador, éste obtenía como respuesta el recurso sin ningún cambio realizado.

Recurso con los campos first_name y last_name modificado y de id 1 utilizando el método PUT.


```
{
    "id": 1,
    "email": "george.bluth@reqres.in",
    "first_name": "John",
    "last_name": "Doe",
    "avatar": "https://reqres.in/img/faces/1-image.jpg"
}
```


Respuesta del servidor con un campo **updateAt** indicando fecha de la modificación, en este caso se da por seguro que el recurso fue actualizado y almacenado nuevamente.


```
{
  "id": 1,
  "email": "george.bluth@reqres.in",
  "first_name": "John",
  "last_name": "Doe",
  "avatar": "https://reqres.in/img/faces/1-image.jpg",
  "updatedAt": "2023-07-10T21:53:56.150Z"
}
```


A continuación, se realiza un GET al recurso y se visualiza el mismo sin ningún cambio.

```
{
  "data": {
    "id": 1,
    "email": "george.bluth@reqres.in",
    "first_name": "George",
    "last_name": "Bluth",
    "avatar": "https://reqres.in/img/faces/1-image.jpg"
  },
```


Si un estudiante crea un recurso REST tiene como una de sus reglas que ese recurso creado tenga una URL de localización en la cabecera lo que marca como un defecto el no tenerlo. Recurso en código JSON que se desea crear.


```
{
    "email": "carlos.matos@reqres.in",
    "first_name": "Carlos",
    "last_name": "Matos"
}
```


Recurso creado añadiendo un campo createAt que indica la fecha de creación.


```
{
  "email": "carlos.matos@reqres.in",
  "first_name": "Carlos",
  "last_name": "Matos",
  "id": "839",
  "createdAt": "2023-07-10T22:12:41.149Z"
}
```


Respuesta de las cabeceras donde debería encontrar la localización del recurso.

Response Headers

Header	Value
date	Mon, 10 Jul 2023 22:12:41 GMT
content-type	application/json; charset=utf-8
content-length	126
connection	close
x-powered-by	Express

Estos errores mencionados anteriormente fueron corregidos en el proyecto de la nueva plataforma REST. Los estudiantes podrán interactuar de una forma sencilla y efectiva el acceso, creación, ac-tualización y eliminación de los datos adquiriendo los conociendo y fundamentos de la tecnología.

3.2. Detalles de la aplicación

La aplicación esta compuesta de dos importantes módulos principales, el Front end y el Back end, esto para dividir las responsabilidades de las interfaces que sirven solo de teoría como de práctica.

3.2.1. Creación del Front End

En la interfaz de Frond end tenemos la parte de teoría que indica el correcto funcionamiento de una API REST, dando a conocer el tipo de acción asociado a su método HTTP correspondiente, asi como la URL que lo involucra y el código de estado.

Figura 3.2. *Página de inicio CAI API REST v1*

1. En esta interfaz encontramos las acciones LIST, SINGLE, SINGLE NO FOUND, CREATE, UPDATE Y DELETE con los métodos que definen el comportamiento de una API REST (GET, POST, PUT, PATCH y DELETE).

2. El Request muestra cómo deben de utilizarse las URL para hacer este tipo de peticiones HTTP.

3. El Response muestra el tipo de código de estado asociado la petición, en este caso observamos un código 200 que significa que es correcto.

4. La respuesta de los datos solicitado en formato JSON.

En la implementación de esta interfaz utilizamos las tecnologías siguientes.

HTML: en el maquetado de la página principal con sus importantes etiquetas.

Bootstrap: en el diseño para darle un terminado con colores suaves a los botones.

Javascript: consumismo la API REST principal para mostrar los recursos utilizando la función (fetch). Puede profundizar acerca de esta función en el siguiente enlace.
`https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch`

3.2.2. Planificación de la Administración

En este apartado se realizó un administrador de contenido para manejar la plataforma, desde aquí el profesor puede visualizar los alumnos y los datos que están almacenado en la base de datos, también puede crear, actualizar y eliminar. Por la parte del estudiante, éste solo podrá visualizar sus recursos que fueron creados por él ya teniendo un número asignado.

Se pueden editar y eliminar los alumnos que ya están utilizando la plataforma y han creado un perfil con rol normal. Solo el profesor puede realizar esta acción ya que un alumno con un rol normal solo puede visualizar y eliminar sus recursos que ha generado.

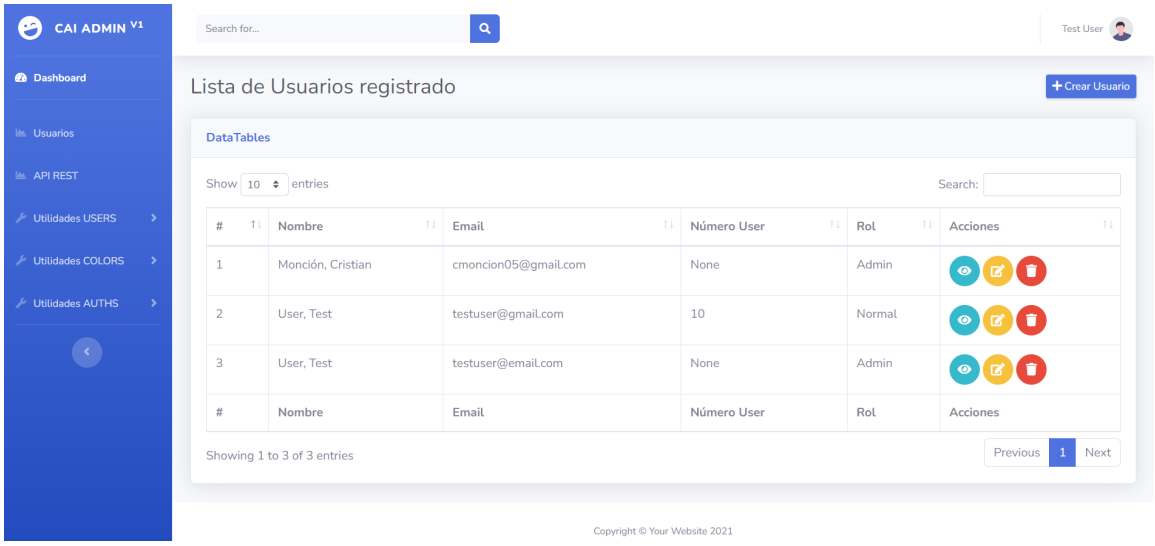


Figura 3.3. *Página de inicio CAI API REST v1*

Se ha creado un menú donde se manejaría cada módulo de la API REST desde el área de administración, ya que con esto buscamos hacerla más dinámica. Aquí podemos ver los siguientes apartados.

Usuarios: muestra los usuarios que están activos, así como el rol que tiene asignado.

API REST: al hacer click te envía a la vista principal del cliente o servicio web.

Utilidades USERS: se utiliza para agregar datos falsos en la API users. Aquí se pueden agregar datos individuales, datos masivos y eliminar los datos existentes.

Utilidades COLORS: se utiliza para agregar datos falsos en la API colors. Aquí se pueden agregar datos individuales, datos masivos y eliminar los datos existentes.

Utilidades AUTHS: solo para eliminar los datos existentes.

3.2.3. Creación BackEnd

Para el desarrollo del Back end se ha dividido en tres módulos API REST como aparece en el manual de práctica, estas son **User**, **Color** y **Auth**. Con esto buscamos que sea lo más autentica a las guías creadas por el profesor a los largos de los años impartiendo la práctica de servicios web con API REST. A continuación, describimos los tres módulos principales.

User: se trata de la API REST principal que hace inicio a la práctica, está cuenta con varios campos que son para mostrar en las acciones de request y response, estas peticiones cliente-servidor la realizarán los estudiantes con los métodos HTTP. Esta compuesta por varias lógicas en código Python que harían funciones como las siguientes.

Descripción del API USERS

Recurso - URL	GET	POST	PUT	DELETE
/users	Lee la lista de users	Añade nuevo user	Sobrescribe la lista de users	
/users/{id}	Lee un user mediate un id		Añade / Sobrescribe un user mediante un id	Elimina un user mediante un id

Tabla 3.1. HTTP users

Color: se trata de una API similar a la de User, la diferencia es que maneja diferentes tipos de información y lo que hace que el alumno tenga variedad en visualizar datos ya sea texto, número y fecha.

Descripción del API COLORS

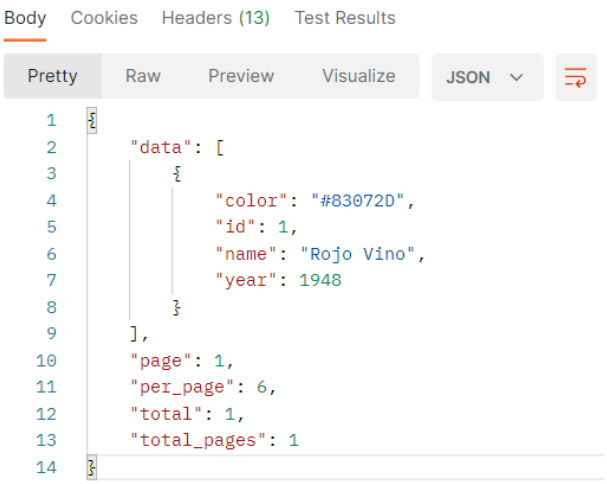
Recurso - URL	GET	POST	PUT	DELETE
/colors	Lee la lista de colors	Añade nuevo color	Sobrescribe la lista de colors	
/colors/{id}	Lee un color mediante un id		Añade / Sobrescribe un color mediante un id	Elimina un color mediante un id

Tabla 3.2. HTTP colors

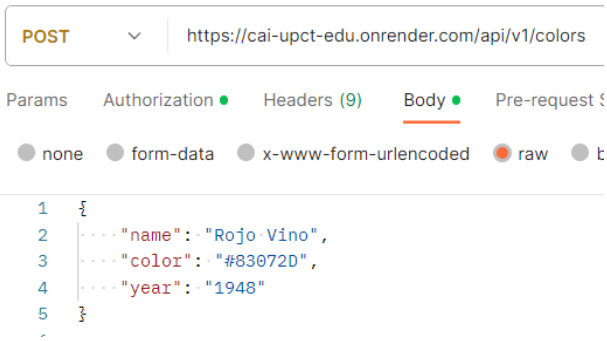
A continuación, en las siguientes imágenes se muestra la realización de un GET y POST, el resto de métodos son haciendo lo mismo mostrado en la API User.

Petición a la URL colors con GET.

Respuesta del servidor



Creando un recorso con POST a la URL colors.

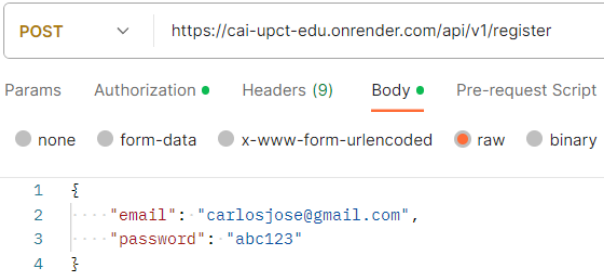


Respuesta del servidor al crear el recurso.

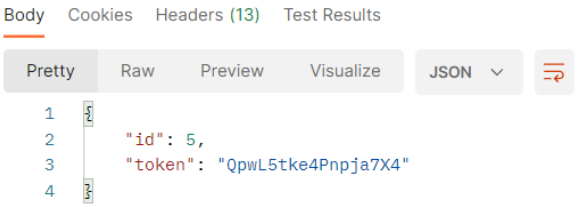
Auth: con auth podemos probar el funcionamiento de autenticación de usuarios mediante un token de acceso. Con esto se busca que una API o servicio web se mantenga seguro o lleve un control de cada usuario que intente utilizar estos servicios.

Peticiones del API Auth

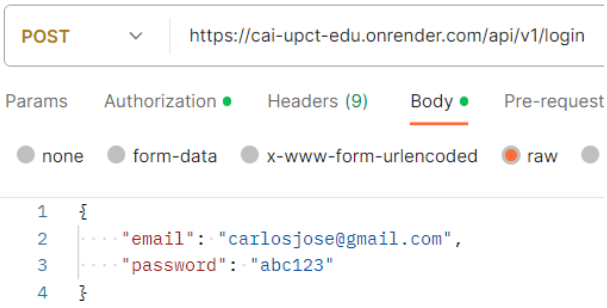
Realizando una de alta (register) con email y contraseña.



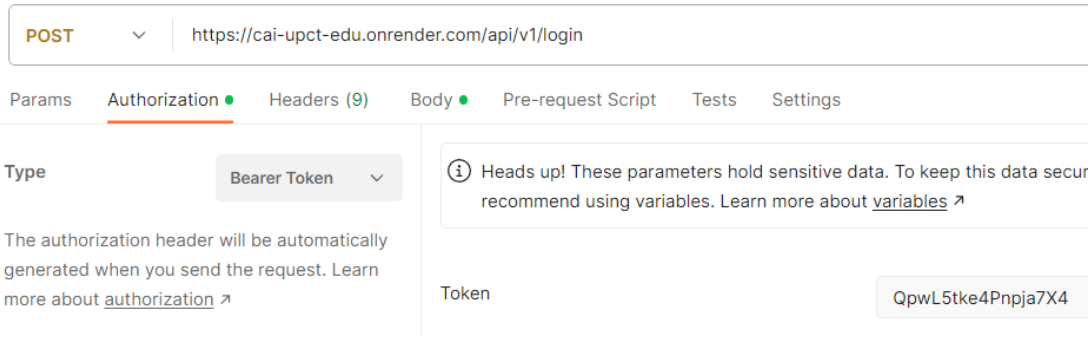
Al realizar el registro el servidor responde con un token y el id del usuario creado.



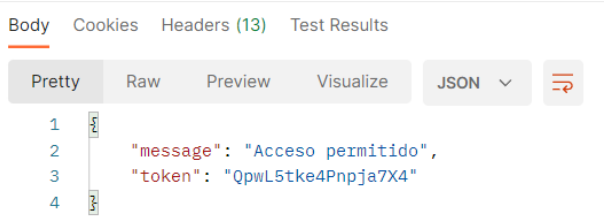
Realizando un acceso al sistema (login) con email, contraseña y token.



Añadiendo el token en la sección de authorization y tipo bearer token.



Respuesta del servidor al momento de haber introducido los datos de login y token válidos.



Si el usuario que desea ingresar al sistema llega a introducir datos erróneos, la API auth puede responder con diferentes mensajes dependiendo la situación. A continuación, se muestran las distintas respuestas que puede retornar el servidor.

1. ”**Necesita el token de acceso**”: si el usuario no coloca el token en authorization.
2. “**El usuario ingresado no existe**”: si el usuario coloca un email que no esté registrado.
3. “**La contraseña es errónea**”: si el usuario coloca una contraseña errónea.
4. “**Acceso permitido**”: si el usuario coloca datos válidos.
5. “**El token no es válido**”: si el token no es el que retorna la el servidor al usuario.

3.3. Codificación en Python

A continuación, se muestra la estructura de directorios del proyecto.

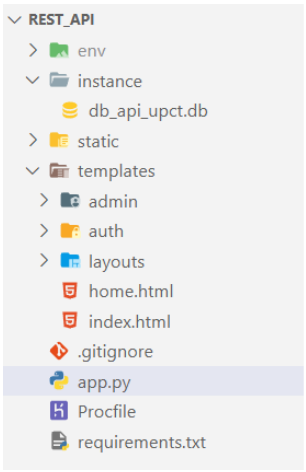


Figura 3.4. *Estructura de archivos*

env: entorno virtual de Python donde almacena todas las dependencias que da funcionamiento a la aplicación. Aquí se encuentran los módulos y librerías necesarios.

instance: contiene en su interior la base de datos creada automáticamente por la el módulo de SQLAlchemy en formato SQLite.

static: contiene las carpetas de archivos estáticos tales como archivos css, imágenes, javascript que son necesarios para el funcionamiento de la aplicación.

templates: se encuentran los archivos que tienen ver con la vista, aquí se almacenan los html y plantillas.

Archivos como **.gitignore** y **Procfile** se explican en al apartado de despliegue de la aplicación a un servidor remoto o producción.

En el archivo **app.py** se encuentra toda la lógica de programación del proyecto.

tequirements.txt: en el se encuentra todas las dependencias y módulos y sus versiones correspondientes.

3.3.1. Describiendo la lógica de programación

Primero debemos de importar todos los módulos de desarrollo de Python, lo que más destacan tenemos los módulos de **flask** y **faker**.

```
from flask import Flask, request, jsonify, render_template, redirect, url_for, flash
from flask_sqlalchemy import SQLAlchemy
from   werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.exceptions import BadRequest
from flask_login import login_user, logout_user, login_required, current_user, UserMixin, LoginManager
import jwt
import requests
from faker import Faker
from flask_cors import CORS
from datetime import datetime, timedelta
```

- En el módulo **flask** un sinfín de librerías para la creación de servicios web, entre las importante están (**request**, **jsonify**, **redirect**, **url_for**) con estas podemos generar una API REST simple pero funcional.
- El módulo **werkzeug** se utiliza para el manejo de la seguridad de una web.
- El módulo **flask_login** manejamos el control de acceso y sesiones de usuarios.
- El módulo **faker** con el creamos los datos para que las tablas de la base de datos inicio con contenido de prueba.
- El módulo **flask_cors** se utiliza para que pueda llevar el control de acceder o hacer peticiones en aplicaciones que estén en diferentes dominios.
- El módulo **datetime** para el manejo de hora y fecha.

La configuración principal de la aplicación la tenemos en este código.

```
# FRAMEWORK FLASK APP
app = Flask(__name__)

# ALL CONFIG VAR
app.config['SECRET_KEY'] = 'secret_key'
app.config["JSON_AS_ASCII"] = False
app.config["JSONIFY_MIMETYPE"] = "application/json; charset=utf-8"

CORS(app)
fake = Faker(['es_ES'])
login_manager=LoginManager(app)

# SQLALCHEMY DATABASE
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///db_api_upct.db"
db = SQLAlchemy()
db.init_app(app)
```

Las configuraciones más destacadas están la creación de la app son:

app = Flask(__name__): con esta creamos la aplicación en general o llamado core de la app.

SQLAchemy: es la instancia para la creación de la base de datos. A partir de eso se pueden crear los modelos de las tablas que tendrán uso en la aplicación. A continuación, mostramos los modelos utilizados para las tablas **user_upct**, **user**, **color** y **auth**.

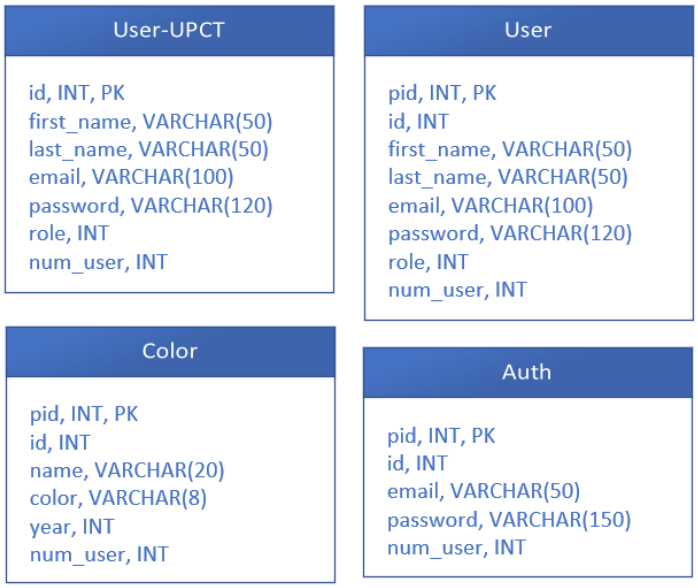


Figura 3.5. *Entidades de tablas del sistema base de datos*

Para trabajar con este ORM primero se debe instalar la dependencia en un proyecto existente de Python con [pip install Flask-sqlalchemy].

```
# SQLALCHEMY DATABASE
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///db_api_upct.db"
db = SQLAlchemy()
db.init_app(app)
```

El código para generar las tablas en con el ORM SQLAlchemy es el siguiente.

```
class UserUPCT(db.Model, UserMixin ):
    id  = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(50))
    last_name = db.Column(db.String(50))
    email = db.Column(db.String(100))
    password = db.Column(db.String(120))
    role = db.Column(db.Integer)
    num_user = db.Column(db.Integer)
```

(a) Class UPCT user

```
class User(db.Model):
    pid = db.Column(db.Integer, primary_key=True)
    id = db.Column(db.Integer)
    first_name = db.Column(db.String(50))
    last_name = db.Column(db.String(50))
    email = db.Column(db.String(100))
    password = db.Column(db.String(120))
    num_user = db.Column(db.Integer)
```

(b) Class User

```
class Color(db.Model):
    pid = db.Column(db.Integer, primary_key=True)
    id = db.Column(db.Integer)
    name = db.Column(db.String(20))
    color = db.Column(db.String(8))
    year = db.Column(db.Integer)
    num_user = db.Column(db.Integer)
```

(c) Class Color

```
class Auth(db.Model):
    pid = db.Column(db.Integer, primary_key=True)
    id = db.Column(db.Integer)
    email = db.Column(db.String(50))
    password = db.Column(db.String(150))
    num_user = db.Column(db.Integer)
```

(d) Class Auth

Figura 3.6. *Entidades de tablas SQLAchemy ORM*

Código que genera la acción GET [**list resources**]

```
@app.route('/api/v1/users', methods=['GET'])
def getAllUsersApiUrl():
    page = request.args.get('page', 1, type=int)
    per_page = 6
    sqlusers = User.query.paginate(page=page, per_page=per_page)
    users = [user.serialize_api_user_url() for user in sqlusers]
    pages_users = {

        "page": page,
        "per_page": per_page,
        "total" : sqlusers.total,
        "total_pages": sqlusers.pages,
        "data": users
    }
    return jsonify(pages_users)
```

El método **getAllUserApiUrl()** es el que lista todos los recursos al hacer un llamado a la API, en el código podemos comentar varias de sus líneas más importantes que serian.

request.args.get(): extrae el querystring de valor un número entero que se le pasa por parámetro para ser la paginación .

User.query.paginate(): Modelo User que extrae los datos de la BD mediante funciones ORM.

[user.serialize_api_user_url() for user in sqlusers]: itera los datos y luego lo almacena en la variable users.

pages_users: parseamos los valores en diccionario de Python y luego los retornamos en formato JSON con la función jsonify(), defecto crea un código de estado 200 OK en la respuesta.

Código que genera la acción POST [**Create resource**]

```python
@app.route('/api/v1//users', methods=['POST'])
def createUserApiUrl():
    if request.method == 'POST':
        first_name = request.json['first_name']
        last_name = request.json['last_name']
        email = request.json['email']
        password = 'abc123'
        user = User(None, id=None,
                    first_name=first_name,
                    last_name=last_name,
                    email=email,
                    password=password,
                    num_user=None)
        db.session.add(user)
        db.session.commit()
        createdAt = datetime.now()
        mod_user = {
            "id": user.pid,
            "first_name": user.first_name,
            "last_name": user.last_name,
            "email": user.email,
            "createdAt": createdAt.strftime("%Y-%m-%d, %H:%M:%S")
        }

        return jsonify(mod_user), 201, {"Location":f"{request.base_url}/{str(user.pid)}"}
```

El método **createUserApiUrl()** es el que crea un recurso en la API, en el código podemos comentar varias de sus líneas más importantes que serian.

first_name, **last_name**, **email** y **resquest.json[]:** contienen los valores que son enviado por el alumno a la hora de crear un recurso.

User: modelo que contiene los campos que se almacenaran en la tabla de la base de datos.

mod_user: parseamos los valores en diccionario de Python y luego los retornamos en formato JSON con la función jsonify() en este caso incluye la localización del recurso y el código de estado 201 en la respuesta.

Código que genera la acción GET **[One resource]**

```python
@app.route('/api/v1/users/<int:id>', methods=['GET'])
def getOneUserApiUrl(id):
    user = User.query.filter_by(pid=id).first()
    if not user:
        return jsonify({"message":"User not found"}), 404
    return jsonify(user.serialize_api_user_url())
```

El método **getOneUserApiUrl()** obtiene un recurso en la API. **User:** realiza una consulta filtrada por el id a la tabla de la base de datos para extraer un único recurso. Si este recurso no existe retorna un mensaje "User not found" en formato JSON, y un código de estado 404 como respuesta.

Código que genera la acción PUT [**Update resource**]

```
@app.route('/api/v1/users/<int:id>', methods=['PUT'])
def updateUserApiUrl(id):
    if request.method == 'PUT':
        user = User.query.filter_by(pid=id).first()
        if not user:
            return jsonify({"message": "User not exit"}), 404
        user.first_name = request.json['first_name']
        user.last_name = request.json['last_name']
        user.email = request.json['email']
        db.session.commit()
        updatedAt = datetime.now()
        mod_user = {
                "id": user.pid,
                "first_name": user.first_name,
                "last_name": user.last_name,
                "email": user.email,
                "updatedAt": updatedAt.strftime("%Y-%m-%d, %H:%M:%S")
            }
        return jsonify(mod_user)
```

El método **updateUserApiUrl()** modifica un recurso en la API.

mod_user: parseamos los valores en diccionario de Python y luego los retornamos en formato JSON con la función jsonify(), por defecto se obtiene el código de estado 200 OK en la respuesta.

Código en Python remueve un recurso [**Delete resource**]

```
@app.route('/api/v1/users/<int:id>', methods=['DELETE'])
def deleteUserApiUrl(id):
    user = User.query.filter_by(pid=id).first()
    if not user:
        return jsonify({"message":"User not found"}), 404
    db.session.delete(user)
    db.session.commit()
    return jsonify({"message":"User deleted"})
```

User: realiza una consulta filtrada por el id a la tabla de la base de datos para extraer un único recurso. Si este recurso no existe retorna un mensaje "User not found" en formato JSON, y un código

de estado 404 como respuesta.

Para el módulo **Color** tenemos la misma lógica de User planteada anteriormente. Los modelos tienen el mismo comportamiento por lo que esta parte nos reservamos su planteamiento.

3.3.2. Tokens de acceso personal

Los tokens de acceso permiten a los clientes llamar de forma segura a las API web protegidas. Las API web usan tokens de acceso para hacer la autenticación y autorización.

Se busca que el alumno se familiarice y vaya entendiendo lo importante que es la seguridad en los servicios web. En ese caso se hará con un código estático (nunca cambia) para evitar confusiones con los alumnos a la hora de colocarlos.

Código en Python que crea un registro.

```python
@app.route('/api/v1/register', methods=['POST'])
def registerApiUrl():
    if request.method == 'POST':
        email = request.json['email']
        password = request.json['password']
        reg_user = Auth(id=None, email=email, password=password, num_user=None)
        db.session.add(reg_user)
        db.session.commit()
        new_user = {
            'id': reg_user.pid,
            'token': 'QpwL5tke4Pnpja7X4'
        }
    return jsonify(new_user), 201, {"Location":f"{request.base_url}/{str(reg_user.pid)}"}
```

Código en Python que realiza un login

```python
@app.route('/api/v1/login', methods=['POST'])
def loginApiUrl():
    if request.method == 'POST':
        email = request.json['email']
        password = request.json['password']
        headers = request.headers
        bearer = headers.get('Authorization')
        if bearer == None:
            return jsonify({"message":"Necesita el token de acceso"})

        token = bearer.split()[1]
        login_user = Auth.query.filter_by(email=email).first()

        if not login_user:
            return jsonify({"error":"El usuario ingresado no existe"})
        if login_user.password != password:
            return jsonify({"error":"La contraseña es errónea"})
        if token == 'QpwL5tke4Pnpja7X4':
            return jsonify({"message":"Acceso permitido","token": token})
        else:
            return jsonify({"error":"El token no es válido"})
```

4. Pruebas y resultados

4.1. Probando la aplicación

Se demuestra los resultados obtenidos en prueba realizada en local al sistema principal de la API en el módulo de usuarios (USERS) esto con la herramienta Postman.

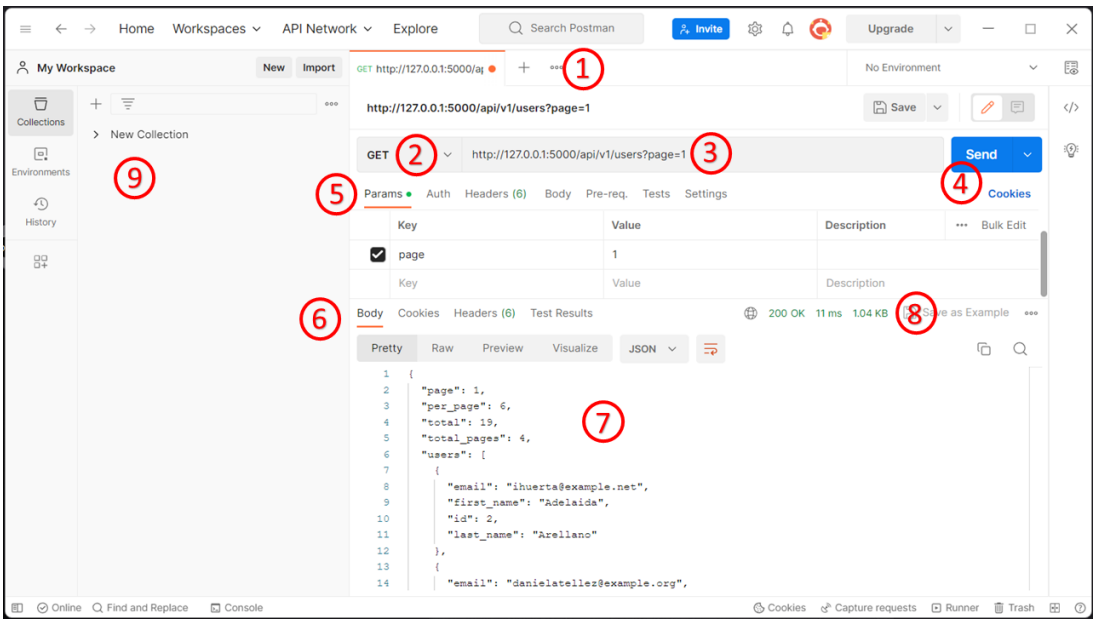


Figura 4.1. *Interfaz de Postman*

Funcionamiento de la herramienta Postman.

1. Botón (+) donde se añaden las pestañas para realizar peticiones.
2. Lista contiene los distintos métodos http [GET, POST, PUT, PATCH, DELETE, etc.].
3. Se añaden las URLs donde se realizarán las peticiones.
4. Botón donde se envía la petición seleccionada.

5. Opciones para incluir datos o cabeceras en la petición.
6. Opciones para visualización, cabeceras y resultado de la respuesta.
7. Resultado o datos retornado al realizar la petición.
8. Código de estatus que retorna la respuesta.
9. Se muestra el historial de todos los métodos que se ha utilizados.

4.1.1. ¿Cómo funciona la API REST en Postman?

Se muestra los pasos para interactuar con la API en modo regular para cualquier usuario que desea hacer solicitudes, para esto el sistema retornará peticiones en formato JSON.

{protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

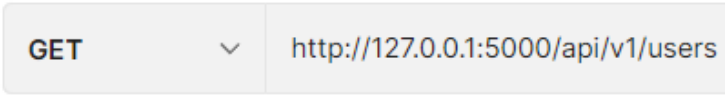
Figura 4.2. *Estructura de una URL*

Para listar se debe saber cuál es la URL principal que en este caso será [**http://127.0.0.1:5000**] y cual es ruta del recurso [**/api/v1/users**] que retorna la petición.

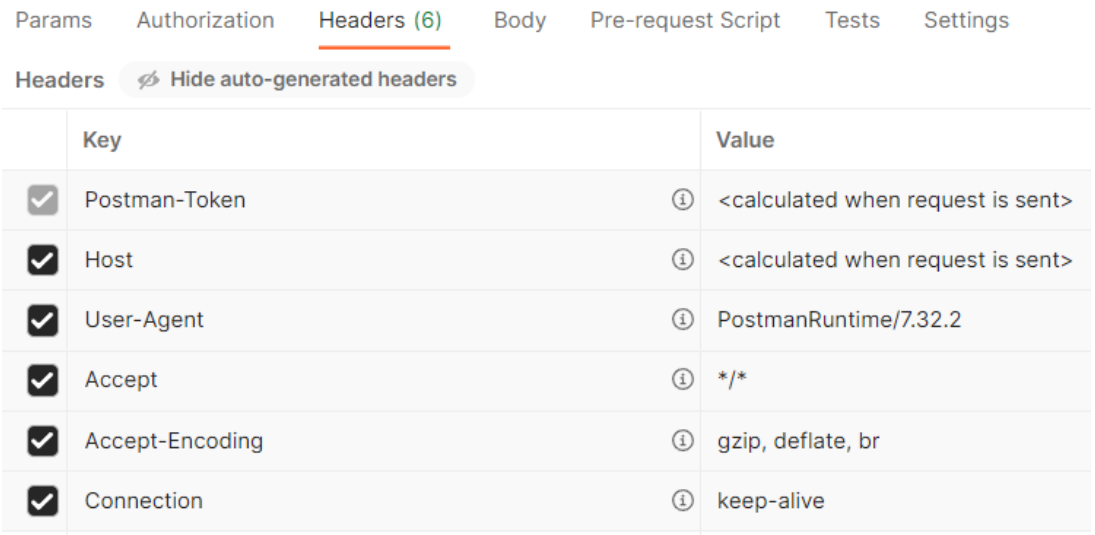
Figura 4.3. *Los distintos métodos HTTP más usados en Postman*

En primer lugar, se elige el método, que sería la acción donde en este caso queremos que nos muestre todos los recursos almacenados [Listar].

Petición a la URL (Request) [http://127.0.0.1:5000/api/v1/users] GET

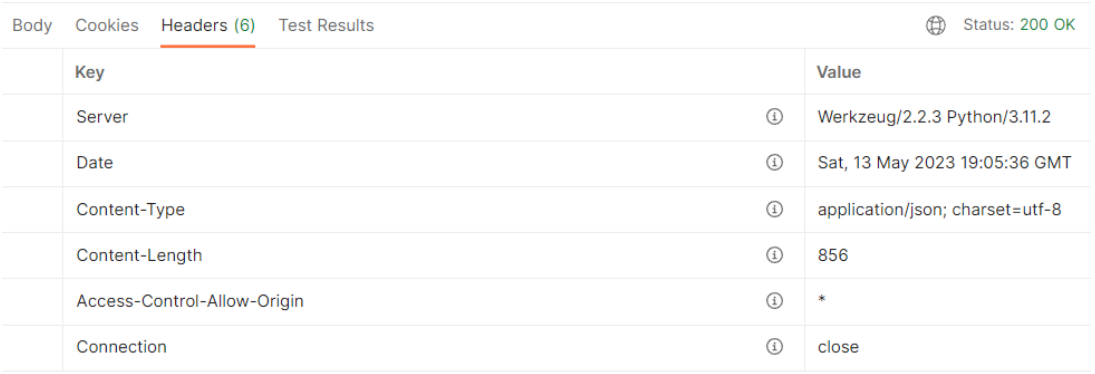


Cabeceras que se envían en la petición.



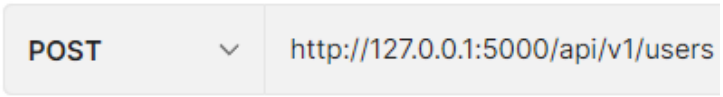
Respuesta del servidor, los datos en formato JSON.

Cabeceras que retorna el servidor más código de estado.

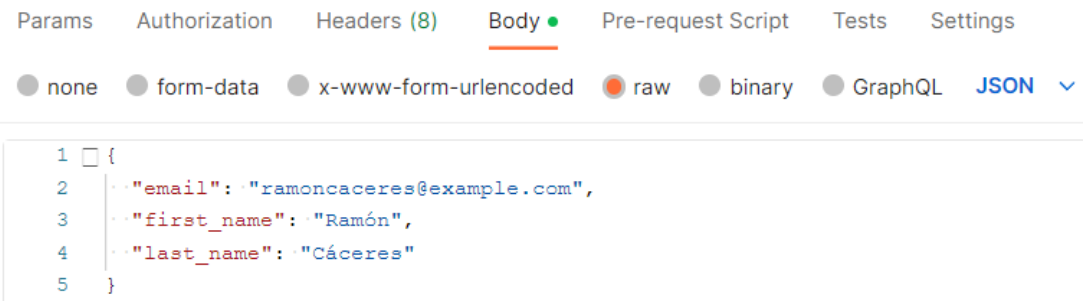


Para crear un recurso en la API REST utilizamos el método POST a la misma ruta de recursos, esto crea una cabecera muy importarte que se trata de (Location) donde indica donde se encuentra el recurso creado.

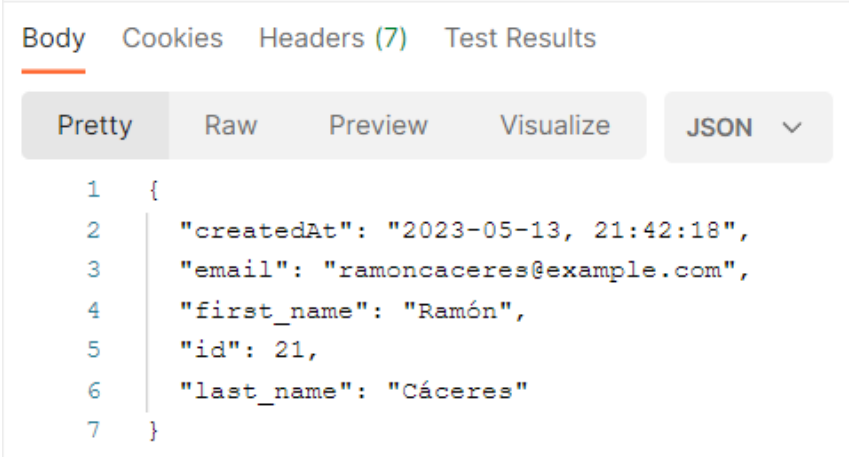
Petición a la URL (Request) [http://127.0.0.1:5000/api/v1/users].



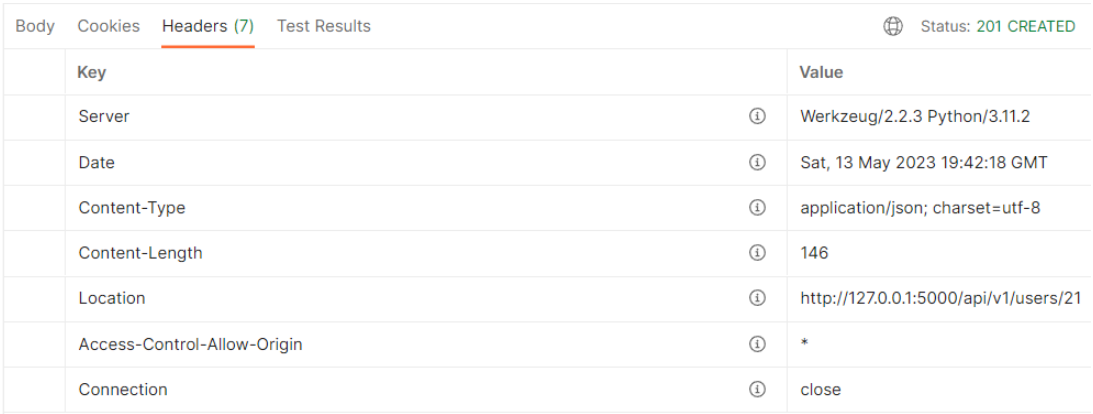
Petición recurso que se quiere crear a la URL (Request) [http://127.0.0.1:5000/api/v1/users] POST.



Respuesta del servidor con el recurso creado en formato JSON.

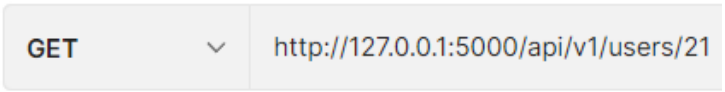


Cabeceras que retorna el servidor más código de estado.

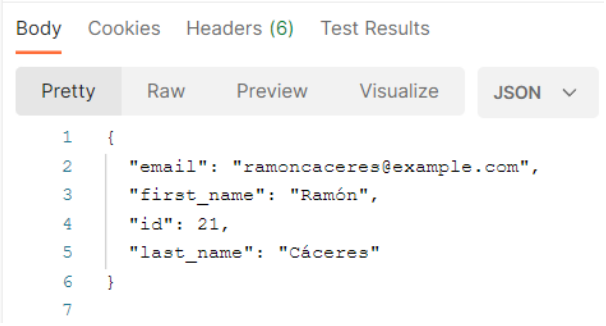


Extraer un recurso en específico mediante un identificador con el método GET, esto retorna un único recurso de una lista de datos almacenado.

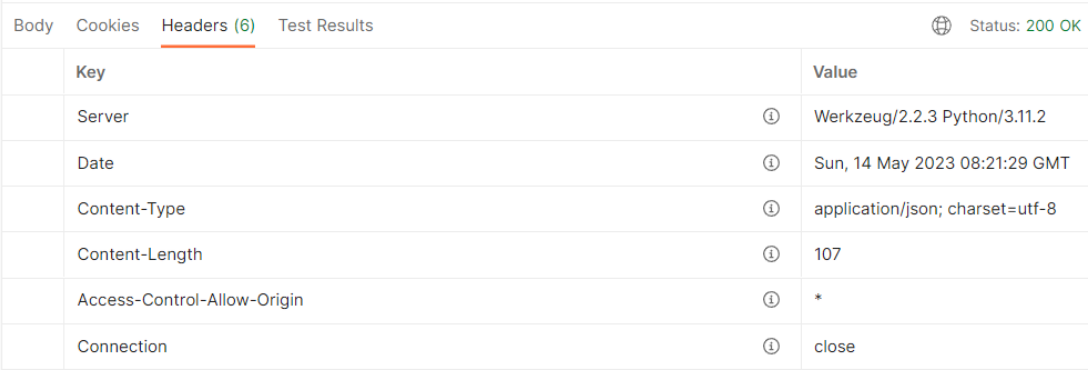
Petición a la URL (Request) [http://127.0.0.1:5000/api/v1/users/21] GET.



Respuesta del servidor en formato JSON.



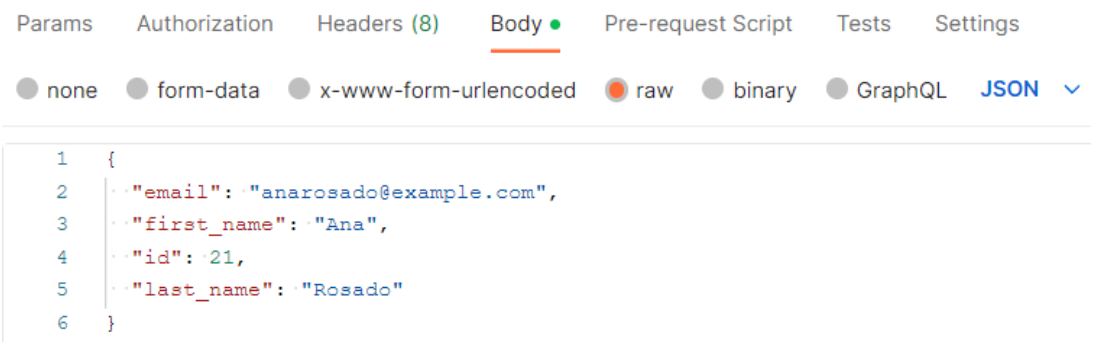
Cabeceras que retorna el servidor más código de estado



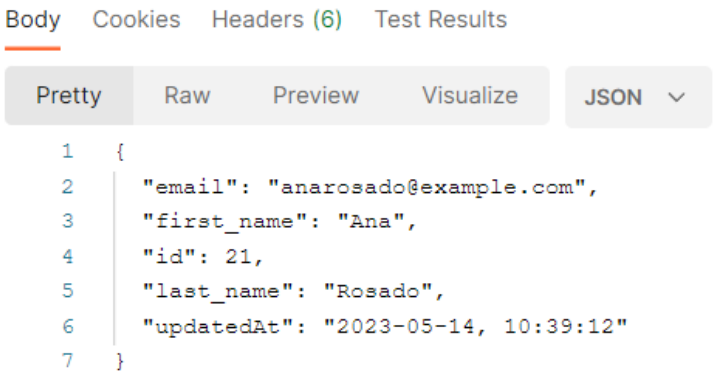
Actualizando (modificando) un recurso existente mediante un identificador único que representa el recurso (21) utilizando el método HTTP PUT. Con este método es obligatorio pasarle este identificador en la URL.

Petición a la URL (Request) [http://127.0.0.1:5000/api/v1/users/21] PUT.

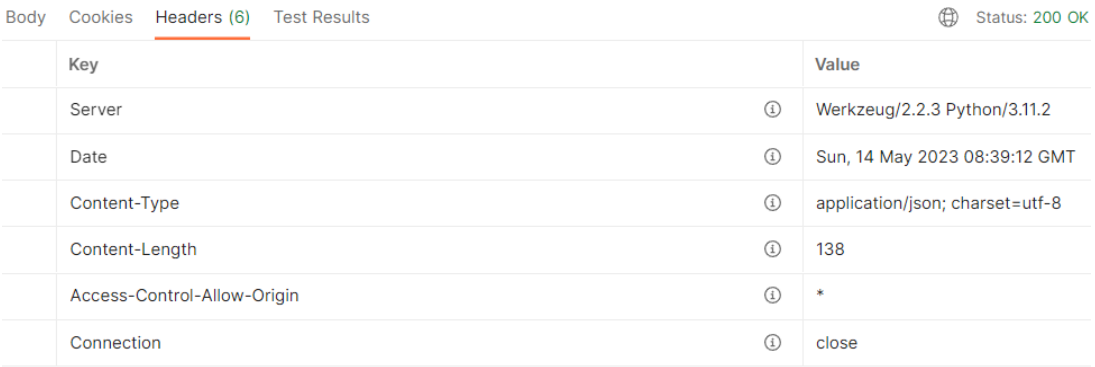
Petición al recurso que se quiere modificar a la URL (Request) [http://127.0.0.1:5000/api/v1/users/21] PUT.



Respuesta del servidor con el recurso modificado en formato JSON.



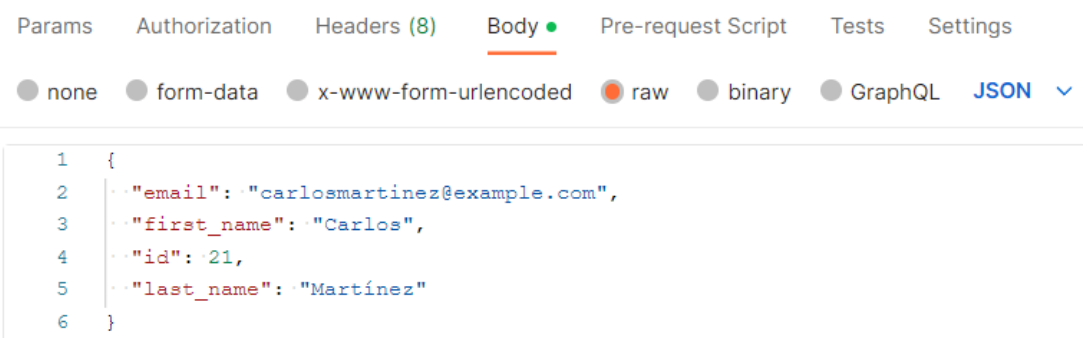
Cabeceras que retorna el servidor más código de estado.



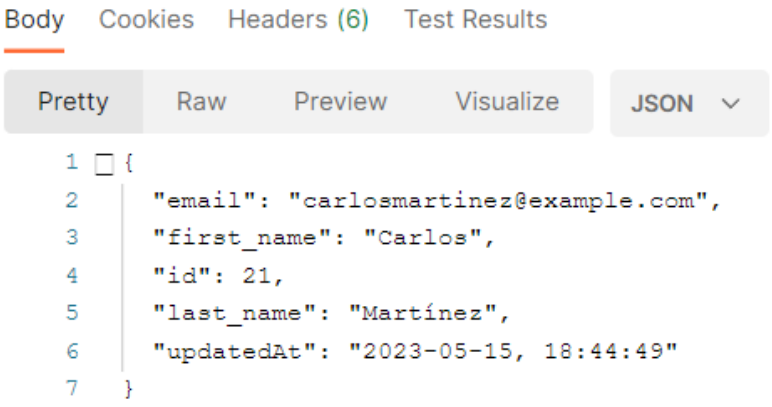
Actualizando (modificando) un recurso existente mediante un identificador único que representa al recurso (21) utilizando el método HTTP PATCH.

Petición a la URL (Request) [http://127.0.0.1:5000/api/v1/users/21] PATCH.

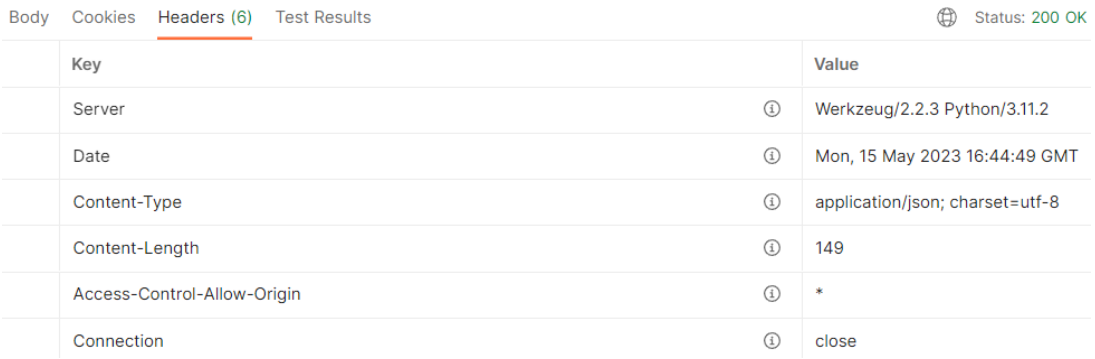
Petición al recurso que se quiere modificar a la URL (Request) [http://127.0.0.1:5000/api/v1/users/21] PATCH.



Respuesta del servidor con el recurso modificado en formato JSON.



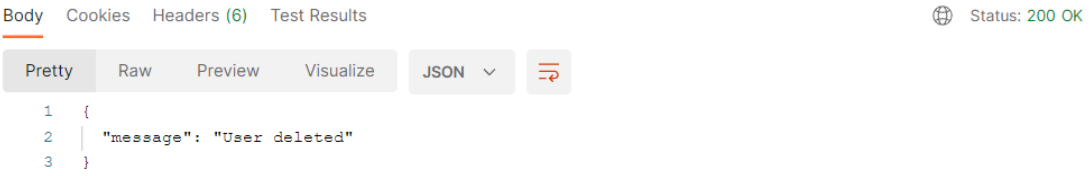
Cabeceras que retorna el servidor más código de estado.



Para eliminar un recurso utilizamos el método HTTP DELETE, mediante un identificador en este caso el mismo que identificador (21) que se ha estado modificando en los anteriores métodos.

Petición a la URL (Request) [http://127.0.0.1:5000/api/v1/users/21] DELETE.

Respuesta del servidor con el recurso eliminado en formato JSON.



También se puso a prueba en una plataforma para servicios web llamada Render.com, esta cuenta con distintos planes de hosting para aplicaciones en distintos lenguajes en nuestro caso utilizamos la alternativa gratuita que sirve para que desarrolladores que le interesen probar su aplicación tenga la oportunidad de exhibirla en el internet. A continuación, se muestra como desplegar una aplicación en un portal paso a paso. En primer lugar, necesitamos un par de herramientas extra y muy importantes estas son GIT Y GITHUB.

4.2. Herramientas de producción

4.2.1. ¿Qué es Git? [14]

Figura 4.4. *Logo de Git. [Fig. 20]*

Git es un sistema (Software) de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta proyectos muy grandes, con rapidez y eficiencia. Es utilizado por las grandes empresas que buscan tener el control de sus aplicaciones basada en versiones por cualquier inconveniente o error que pueda suceder, con solo con un simple comando de git este puede retroceder a una versión anterior estable. Comandos que se utilizaran en el despliegue de la aplicación.

Comandos	Utilización
git init	Crea un repositorio de git en la carpeta del proyecto.
git status	Visualiza, despliega o nos da información sobre cuales archivos en el proyecto han sido creados o modificados.
git add .	Agrega los archivos en estado de preparado.
git commit	Realiza una captura de los cambios realizados actualmente en el proyecto.
git remote	Se conecta o establece una conexión al repositorio remoto mediante una url.
git branch	Crea, lista y borra ramas que existen en un determinado repositorio.
git push	Se utiliza para enviar los archivos de un repositorio local a un remoto.

Tabla 4.1. Comandos para realizar despliegues en git.

4.2.2. GitHub [15]

Figura 4.5. *Logo de GitHub. [Fig. 21]*

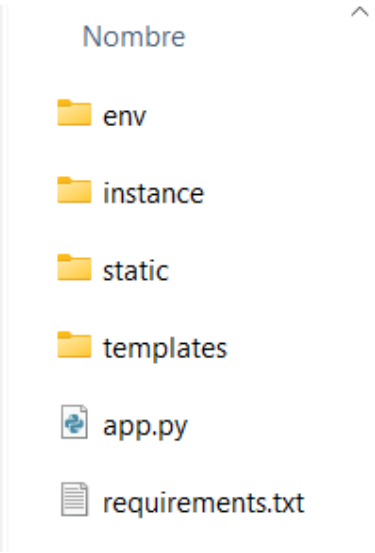
Es una plataforma para desarrollo colaborativo, github te permite almacenar tus proyectos para darte a conocer como desarrollador o más bien todo tipo de empresas guardan sus códigos en esta plataforma con el fin de compartir proyectos ya sea entres sus desarrolladores o con otras entidades externa.

En esta se crea un espacio para alojar los archivos llamado repositorios que puede ser tanto públicos como privado despendiendo el tipo de proyecto y si puede ser compartido. Obviamente existen alternativas a GitHub como son GitLab, BitBucket. Launchpad, Gitea, entre otras.

4.3. Repositorios de GitHub

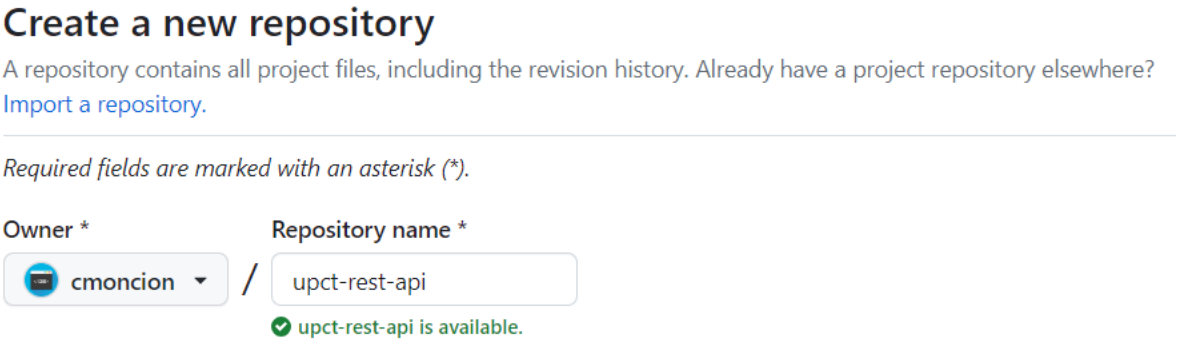
Se recomienda tener una cuenta o estar dado de alta en **GitHub**, así como la herramienta de control de versión **git** instalado en el ordenador o computadora.

Estructura del proyecto en local para subir a github.



Primero entramos a GitHub y creamos un repositorio.

A continuación, le damos un nombre al reprositorio.

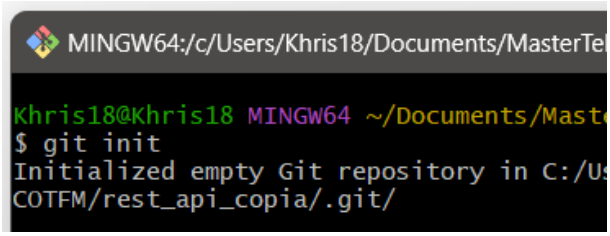


Comando que provee GitHub que se van a utilizar para subir el repositorio local.

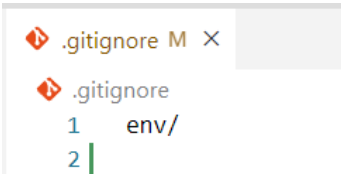
...or create a new repository on the command line

```
echo "# upct-rest-api" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/cmoncion/upct-rest-api.git
git push -u origin main
```

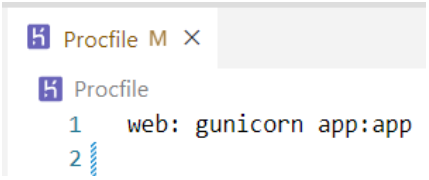
Creamos el repositorio en el directorio del proyecto con git init utilizando una terminal o la consola.



Como estos tipos de plataforma (Render) es un servicio **Platform as a Service (Paas)**, es decir no tiene que preocuparse por los servidores, el almacenamiento, el sistema operativo, por lo que ya se tienen software integrado como Python en su SO, por lo que podemos obviar la carpeta **env** del entorno virtual porque este puede ser creado automáticamente en el despliegue de la aplicación. Para esto crearemos un archivo de nombre **.gitignore**, este se utiliza para anular la subida ya sea de archivos y carpetas que no desea que se desplieguen, contenido que debe de tener este archivo se presenta en la imagen.



Así también creamos el archivo **Procfile** necesario para el funcionamiento en la aplicación en Render, se muestra el contenido en el archivo en la imagen.



Con todo esto iniciamos la subida a GitHub con los comandos dados, usamos git status.

```
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        Procfile
        app.py
        instance/
        requirements.txt
        static/
        templates/

nothing added to commit but untracked files present (use "git add" to track)
```

Seguido de git add .

```
$ git add .
```

Utilizamos git commit para guardar o capturar el estado actual.

```
$ git commit -m "Primer commit a GitHub API REST"
[main (root-commit) ab7fdaf] Primer commit a GitHub API REST
```

Nos conectamos al repositorio creado anteriormente en github.

```
$ git remote add origin https://github.com/cmoncion/upct-rest-api.git
```

Utilizamos el comando git push para subir los archivos a github.

```
$ git push -u origin main
Enumerating objects: 1903, done.
Counting objects: 100% (1903/1903), done.
Delta compression using up to 8 threads
Compressing objects: 100% (1898/1898), done.
Writing objects: 100% (1903/1903), 5.66 MiB | 2.52 MiB/s, done.
Total 1903 (delta 210), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (210/210), done.
To https://github.com/cmoncion/uptc-rest-api.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

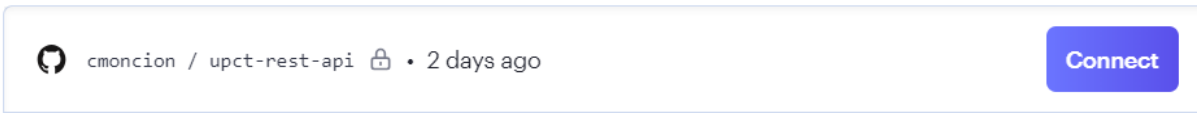
Se muestran los archivos del proyecto subido en el repositorio remoto.

4.4. Despliegue en un servicio cloud gratuito

4.4.1. Render Cloud [16]

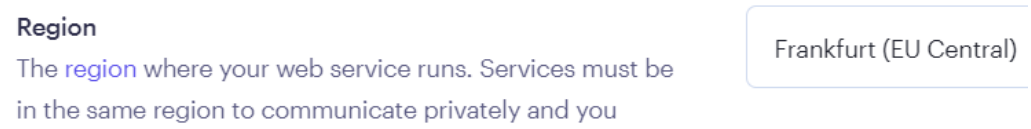
Render es una plataforma para brindar servicios en la nube, que te facilita crear y ejecutar todas tus aplicaciones y sitios web con certificados TLS gratuitos, CDN global, redes privadas y despliegues automáticos desde Git, haciéndolo ver de una manera simple. Esta cloud cuenta con alojamiento para diferentes tipos de lenguaje como Python, NodeJS, Go, Rust, PHP, entre otros, todo empezando con un plan gratuito.

Luego teniendo ya una cuenta en **render** utilizando el perfil de **github**, para crear un servicio web, clickeamos en el botón **New** + y seleccionamos **Web Service** entres las opciones. Aquí podremos ver como se encuentran vinculados todos los proyectos que tenemos en nuestros repositorios. En nuestro caso con el que vamos a desplegar.



Les asignamos un nombre al servicio, en nuestro caso le ponemos cai-upct-educ.

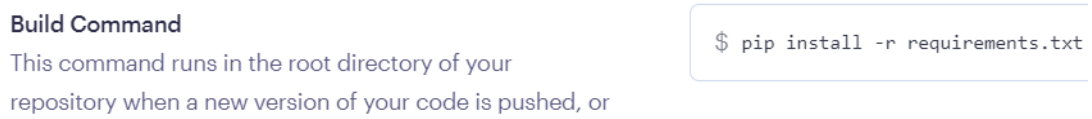
Elegimos la región en la que queremos que se encuentre alojado nuestro servicio.



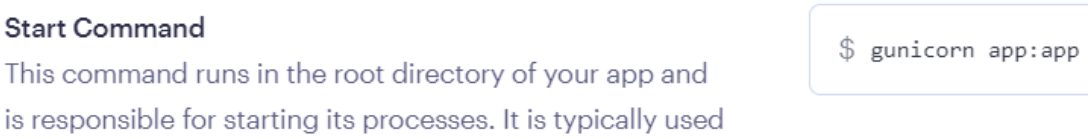
La rama principal que se hará el push en nuestro proyecto en el repositorio en github.

Lenguaje de programación en que fue creado el proyecto.

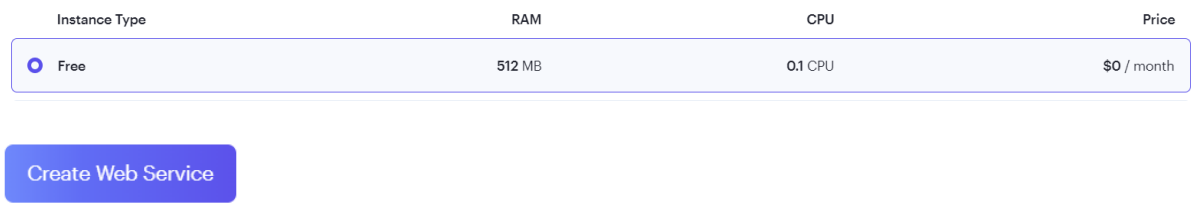
El archivo requirements.txt es donde se encuentran todas las dependencias que hacen funcionar el servicio. Una vez iniciado el entorno virtual en la nube, se corre el comando.



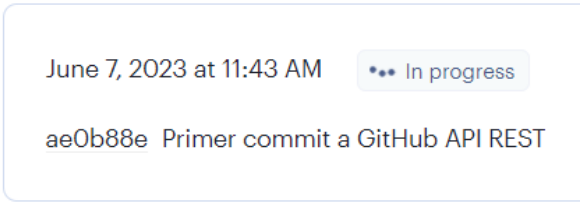
Comando que ejecuta la aplicación, podemos encontrarlo en el archivo **Procfile** mencionado anteriormente.



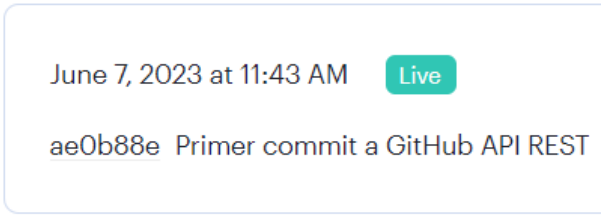
Elegimos el tipo de plan que queremos, en nuestro caso es el Free y presionamos el botón crear servicio.

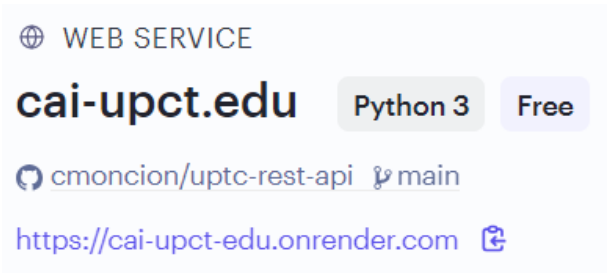


Esperamos a que se finalice el despliegue



Cuando el despliegue este listo veremos que ya está en producción **Live** y todo salió correcto, por lo que podemos visitar el servicio en el siguiente enlace o URL.





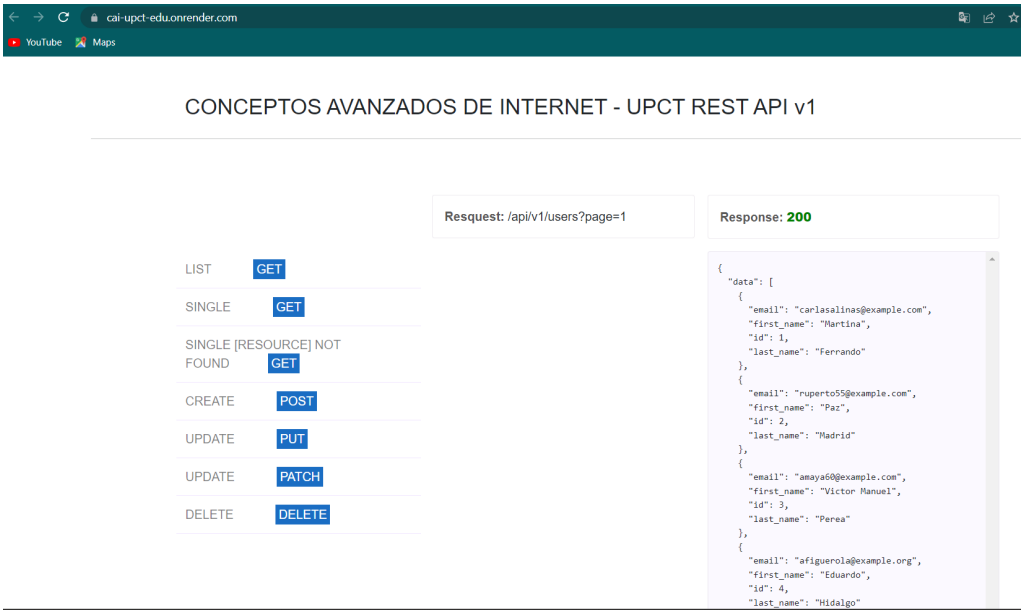


Figura 4.6. *Servicio web en el internet.*

4.5. Pruebas en producción

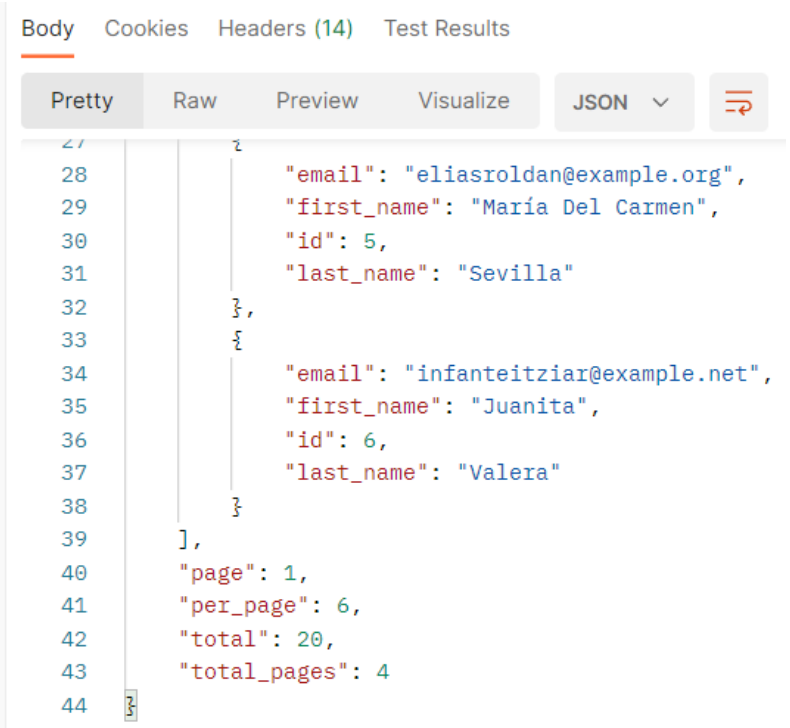
El servicio ya se encuentra en el internet por lo que es necesario saber si este cumple con lo esperado, así que les realizaremos las pruebas para obtener su correcto funcionamiento en producción con la aplicación Postman.

En nuestro caso recordamos que la URL en producción es [**https://cai-upct-edu.onrender.com**] por lo que para acceder a la API seria [**https://cai-upct-edu.onrender.com/api/v1/users**].

4.5.1. GET USERS LIST

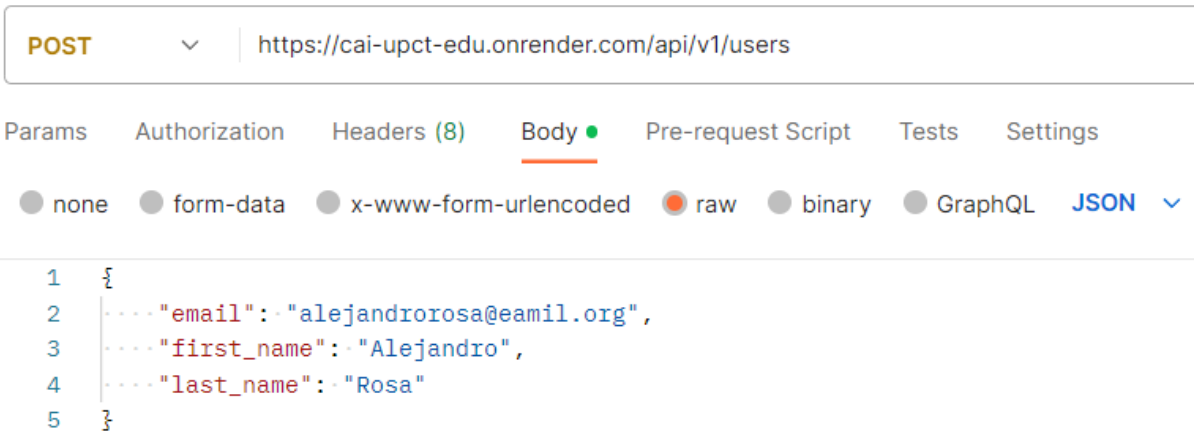
Request:

Response:



4.5.2. POST CREATE USER

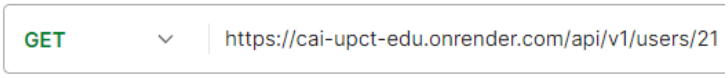
Request:



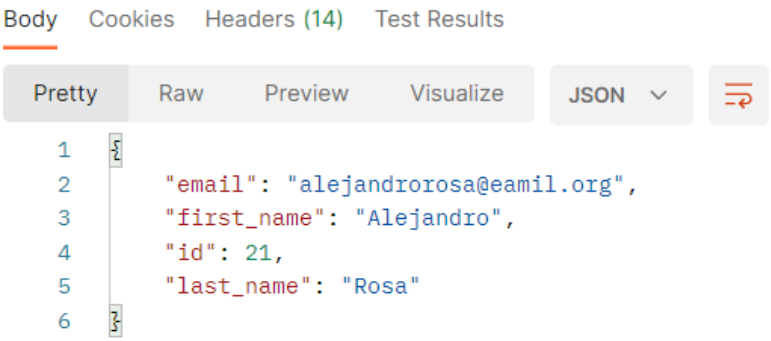
Response:

4.5.3. GET ONE USER

Request:

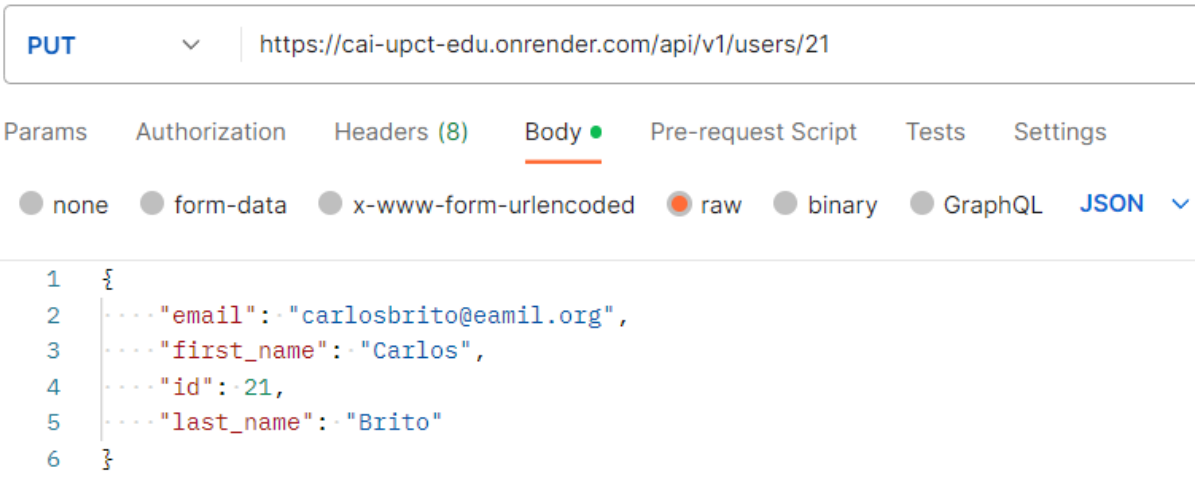


Response:

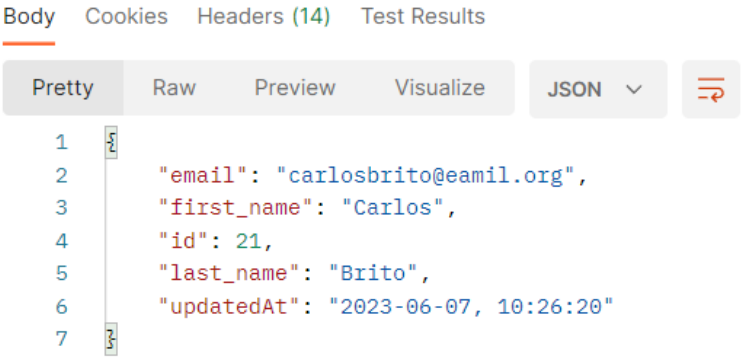


4.5.4. PUT UPDATE USER

Request:



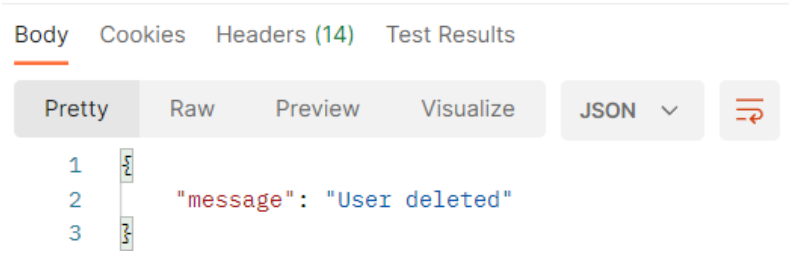
Response:



4.5.5. DELETE ON USER

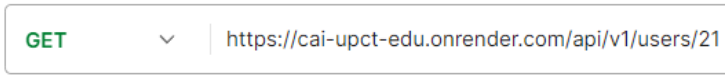
Request:

Response:

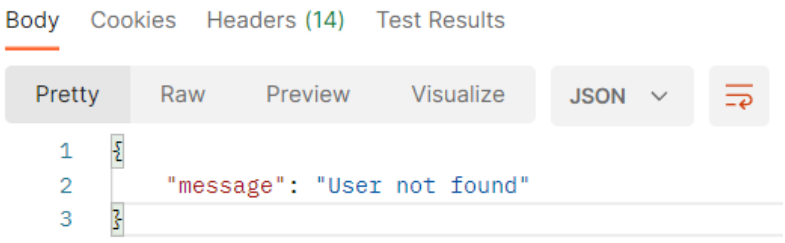


4.5.6. USER NOT FOUND

Request:



Response:



Como pudimos observar se utilizaron todos los métodos HTTP con los que trabaja una API REST.

4.6. API REST para alumno individual

4.6.1. Multi-Tenancy [17]

Tenencia múltiple o **multitenencia** en informática corresponde a un principio de arquitectura de software en la cual una sola instancia de la aplicación se ejecuta en el servidor, pero sirviendo a múltiples clientes u organizaciones (tenedor o instancia). Este modelo se diferencia de las arquitecturas con múltiples instancias donde cada organización o cliente tiene su propia instancia instalada de la aplicación. [wiki]

Si desean profundizar más del tema Multi-Tenancy en la etiqueta de la figura se muestra un enlace que provee de una buena explicación.

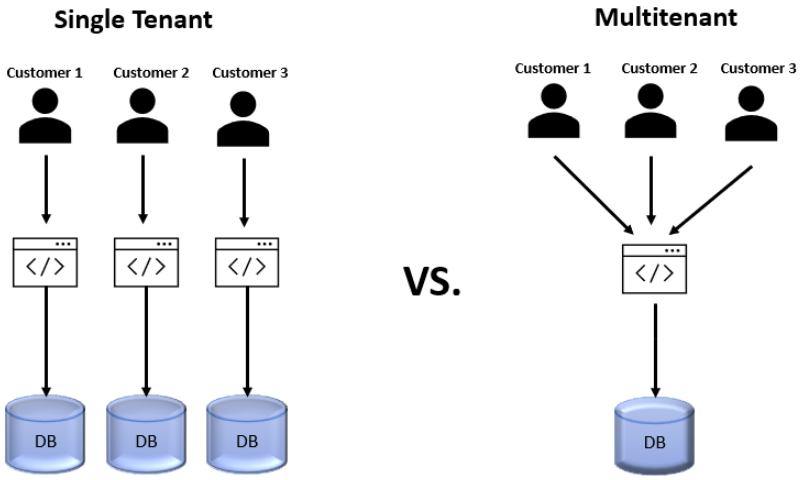


Figura 4.7. *Tipos de Tenancy. [Fig. 22]*

En nuestro caso la plataforma de aprendizaje REST, tiene la capacidad de poder asignar una instancia a un alumno individual mediante un número para que este pueda manipular sus propios datos. Por lo que en este apartado probaremos en modo resumido cuál es el funcionamiento de lo que se espera de la API de aprendizaje individual.

En primer lugar se puede considerar la misma URL pero con un identificador de alumno. A continuación, se dan detalle de cómo obtener este número.

1. El profesor encargado de la asignatura deberá de proveer a los alumnos de la URL para hacer un registro (de alta) en el sistema. Este debe de ingresar su nombre y apellido, correo, así como una contraseña.

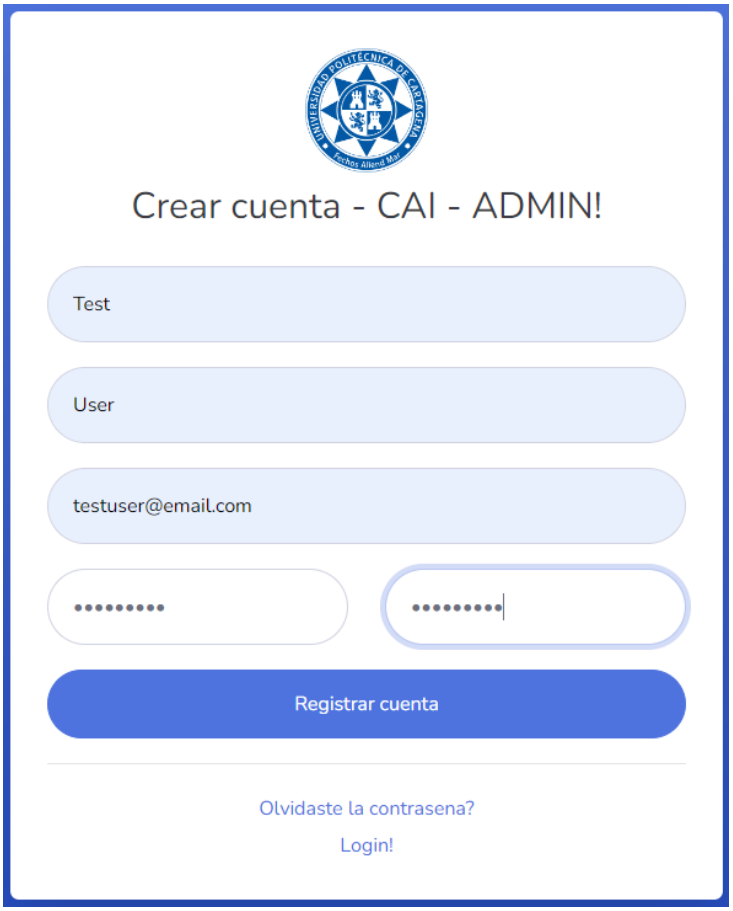


Figura 4.8. *Formulario de registro [http://localhost:5000/auth/register]*

2. Después de haberse dado de alta el estudiante debe ingresar a la pantalla de ingreso (Login) esto lo llevará a la pantalla de manejo estudiante.

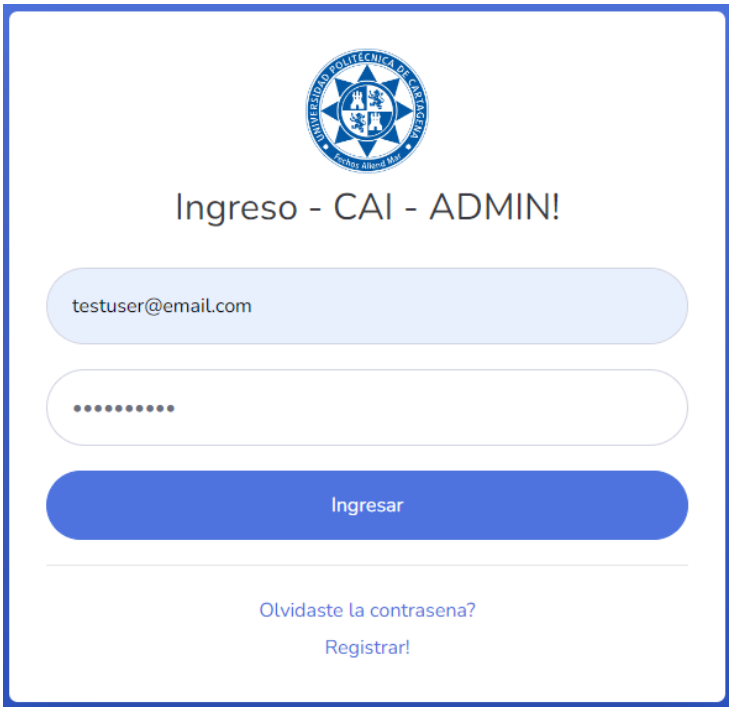
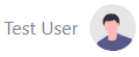


Figura 4.9. *Formulario de acceso [http://localhost:5000/auth/login]*

3. Luego de haber accedido al sistema, el alumno debe de dirigirse a su perfil 

e introducir el número asignando por el profesor. En la parte donde se muestran los datos personales del alumno, existe un campo de número asignado, es aquí donde debe añadir el número y guardar.

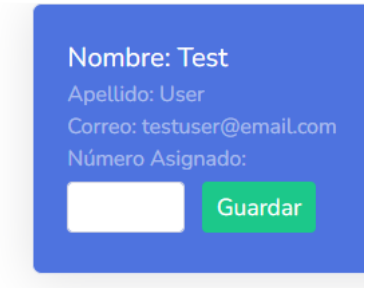


Figura 4.10. *Perfil del alumno en el sistema*

4. Una vez ya con el número el alumno puede trabajar con la API REST individual, para esto debe de utilizar la estructura de la siguiente URL en su aplicación en este caso Postman.

https://cai-upct-edu.onrender.com/api/v1/10/users

- **URL:** URL base del servicio web.
- **Api:** ruta donde se encuentra el servicio REST.
- **v1:** versión de la API.
- **10:** Número de alumno individual.
- **users**: datos almacenados de la API.

Visualizamos a los alumnos **número 10** y **número 20** en un cliente HTTP llamado **Thunder Client** en Visual Studio Code, como podemos observar ambos no tienen datos almacenados en sus usuarios en el sistema. Para esto utilizamos una operación GET.

Luego añadiremos un recurso independiente a cada usuario con el método POST.

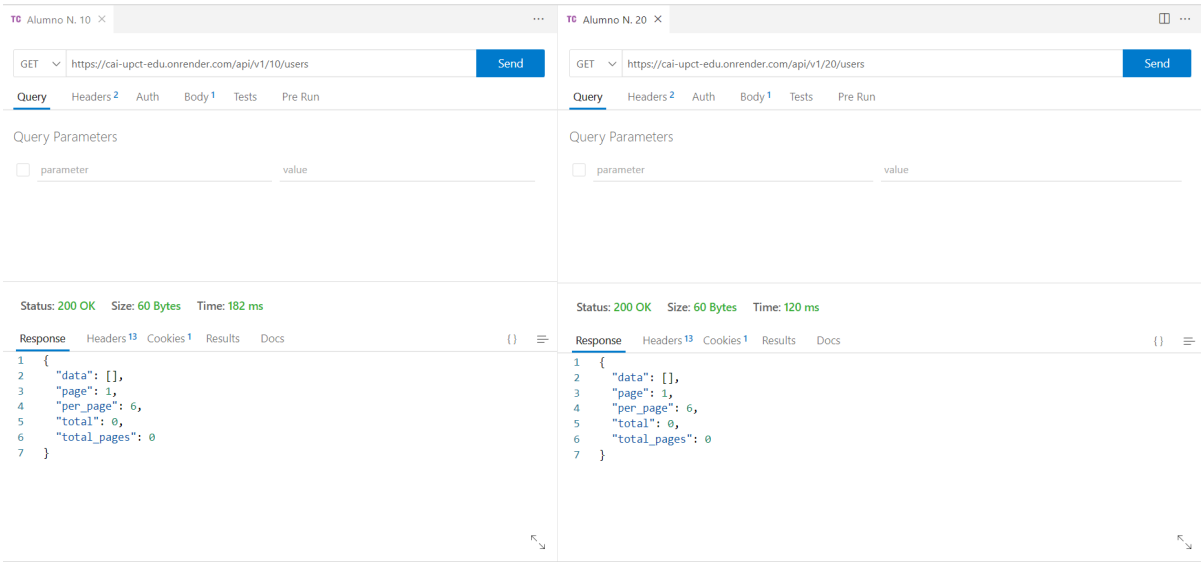


Figura 4.11. *GET alumno individual I*

Luego añadiremos un recurso independiente a cada usuario con el método POST

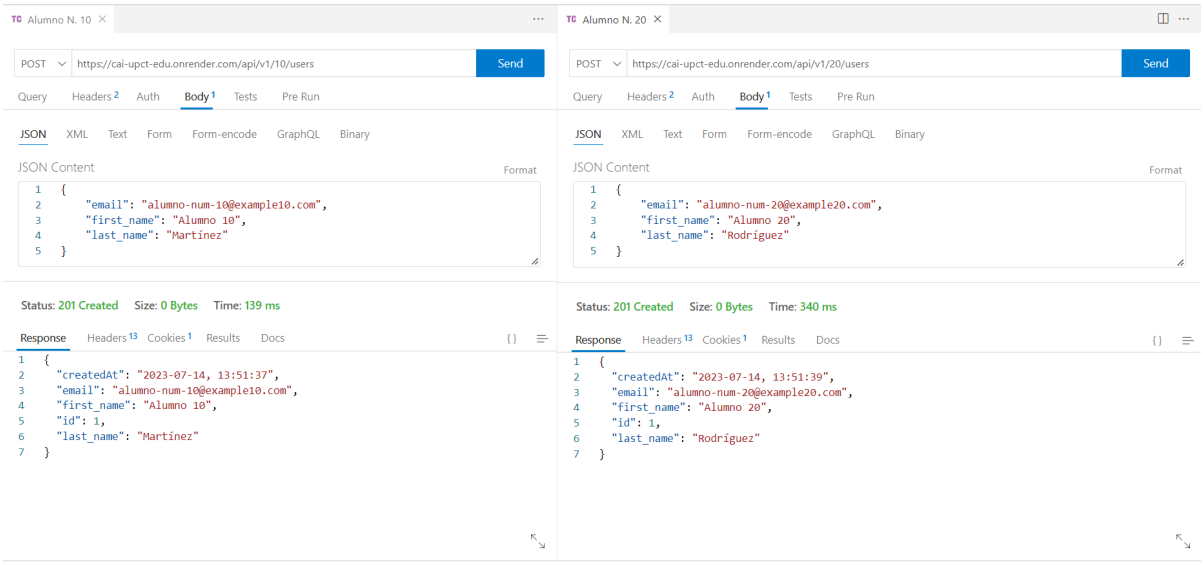


Figura 4.12. *POST alumno individual*

Realizamos un GET a todos los recursos y obtenemos este resultado.

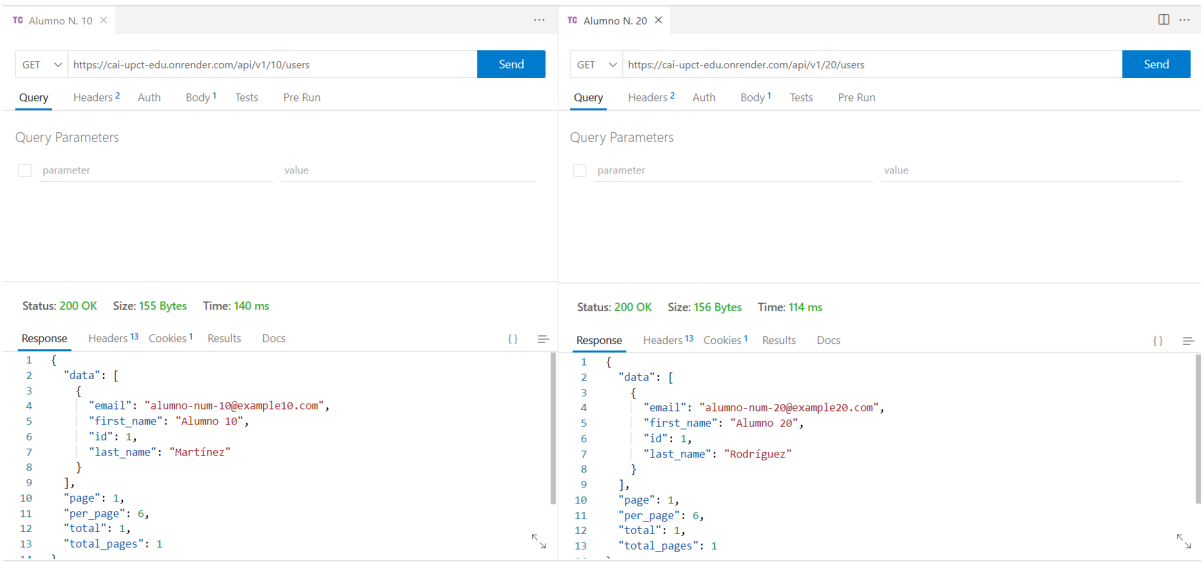


Figura 4.13. *GET alumno individual II*

Actualizamos los recursos mediante el método PUT.

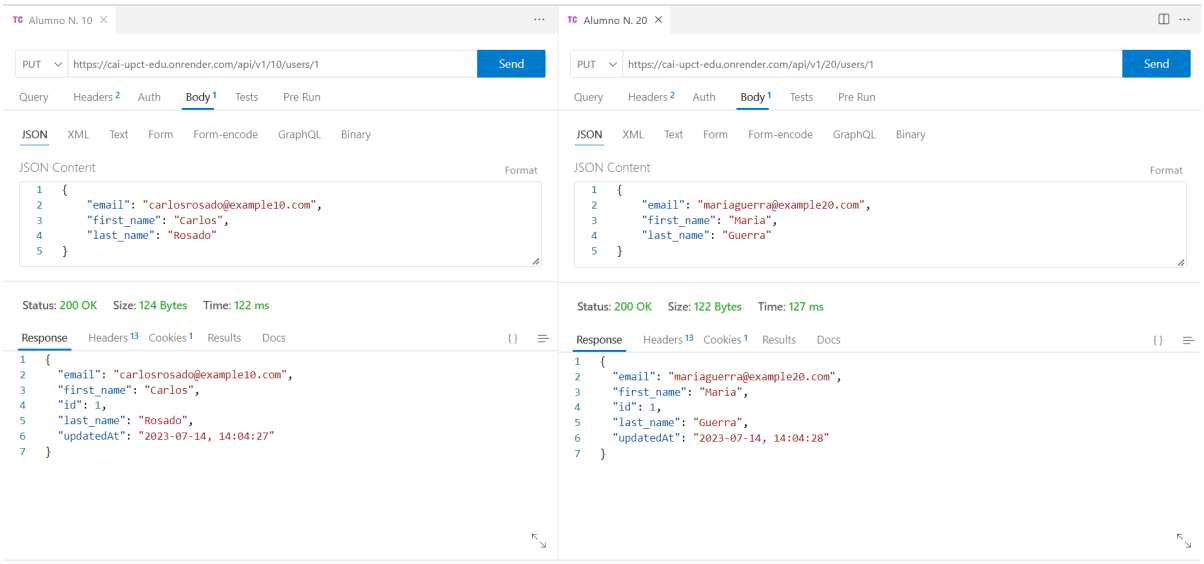


Figura 4.14. *PUT alumno individual*

Realizamos un GET al recurso con el identificador.

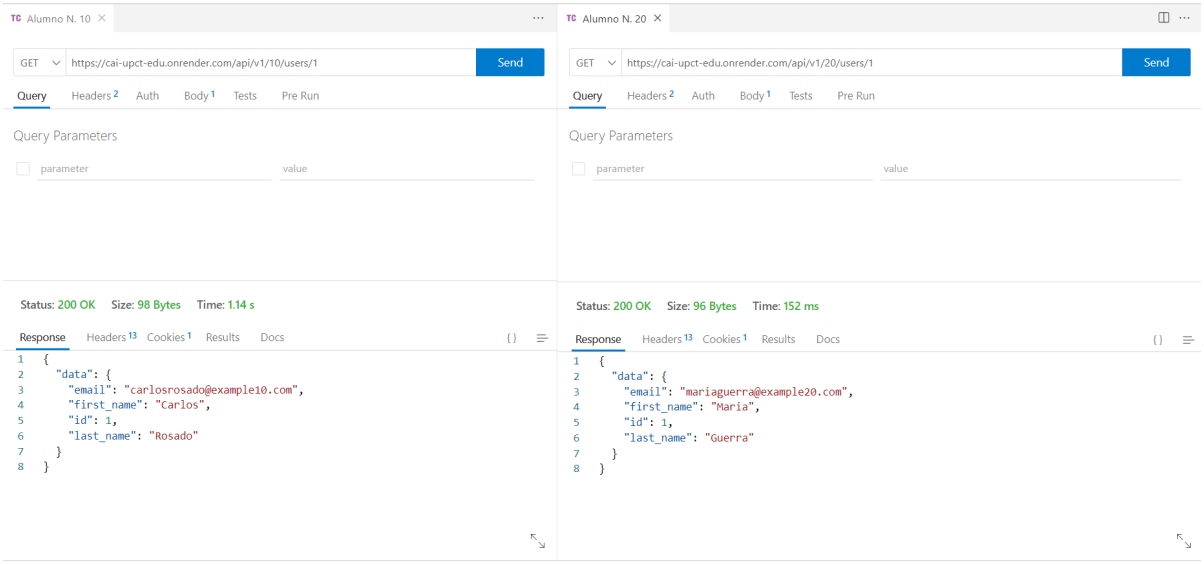


Figura 4.15. *GET alumno individual III*

Eliminamos el recurso con un DELETE.

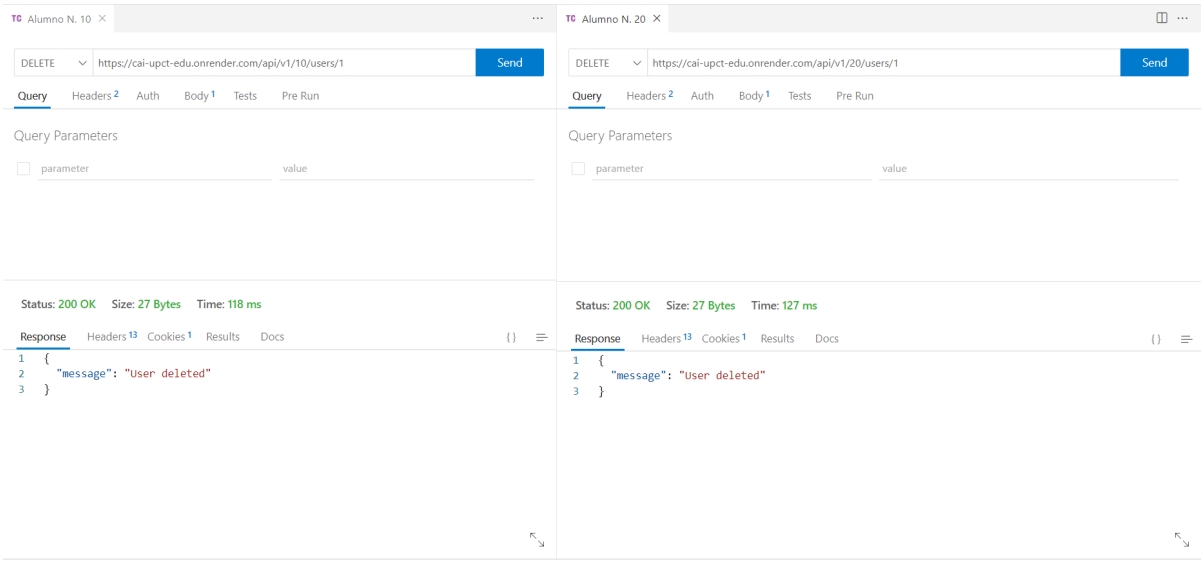


Figura 4.16. *DELETE alumno individual*

Realizamos el Request que no se encuentra el recurso.

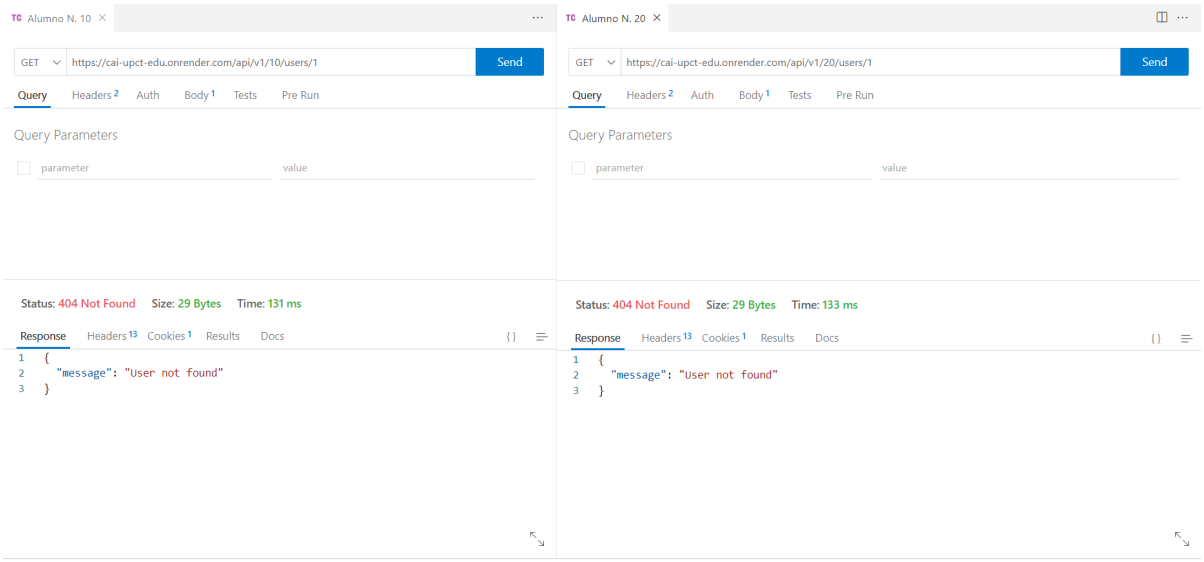


Figura 4.17. *GET alumno individual IV*

5. Conclusión

Este proyecto concluye destacando la importancia de las plataformas para el aprendizaje de las API REST. Esta plataforma está diseñada para proporcionar a los estudiantes de Conceptos Avanzados de Internet una herramienta interactiva para aprender los conceptos básicos de la API REST y su estructura.

Se espera que la plataforma demuestre su eficacia al ofrecer formas de aprendizaje estructurados, ejemplos prácticos de realizar peticiones de datos, conocer los distintos métodos de manipulación y que con esto facilite a los estudiantes entender esta tecnología. Además, se enfatiza la importancia de las mejores prácticas en el diseño y desarrollo de API REST, lo que brindará a los estudiantes una sólida base de los aspectos clave de la creación de un servicio web seguros, escalables y eficientes.

Usar Python como mi principal lenguaje de programación me permitió familiarizarme con las bibliotecas y herramientas involucradas en el desarrollo de API REST. Esto me proporcionó una base sólida para seguir explorando y ampliando mis conocimientos de programación con Python y en este campo en constante crecimiento.

Espero que esta plataforma sea una herramienta que demuestre lo que se espera de ella, y que sea de importancia para estudiantes, desarrolladores y cualquier persona que busque obtener habilidades prácticas de API REST. También esperamos poder modernizarla en distintas versiones para así hacerla más completa.

6. Bibliografía

[1] Wikipedia, *API — Wikipedia, La enciclopedia libre*, [Internet; descargado 30-junio-2023], 2023. dirección: https://es.wikipedia.org/w/index.php?title=API&oldid=152173694.

[2] Wikipedia contributors, *Representational state transfer — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=1164422855.

[3] M. MDN, *JavaScript Object Notation (JSON)*, [Internet; developer.mozilla.org]. dirección: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON.

[4] Wikipedia, *HTML5 — Wikipedia, La enciclopedia libre*, [Internet; descargado 3-junio-2023], 2023. dirección: https://es.wikipedia.org/w/index.php?title=HTML5&oldid=151616266.

[5] M. MDN, *CSS: Cascading Style Sheets*, [Internet; developer.mozilla.org]. dirección: https://developer.mozilla.org/en-US/docs/Web/CSS.

[6] M. MDN, *JavaScript*, [Internet; developer.mozilla.org]. dirección: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[7] Wikipedia, *Bootstrap (framework) — Wikipedia, La enciclopedia libre*, [Internet; descargado 22-junio-2023], 2023. dirección: https://es.wikipedia.org/w/index.php?title=Bootstrap_(framework)&oldid=152008510.

[8] Wikipedia contributors, *Python (programming language) — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1164530008.

[9] Wikipedia contributors, *Flask (web framework) — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=Flask_(web_framework)&oldid=1165042747.

[10] P. SQLAlquimia, *SQLAlquimia*, [Internet; pypi.org]. dirección: https://pypi.org/project/SQLAlchemy.

[11] Wikipedia contributors, *Visual Studio Code — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=Visual_Studio_Code&oldid=1164064847.

[12] M. Piacentini, *DB Browser for SQLite*. dirección: https://sqlitebrowser.org/.

[13] Wikipedia contributors, *Postman (software) — Wikipedia, The Free Encyclopedia*, [Online; accessed 14-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=Postman_(software)&oldid=1165163661.

[14] Wikipedia, *Git — Wikipedia, La enciclopedia libre*, [Internet; descargado 1-junio-2023], 2023. dirección: https://es.wikipedia.org/w/index.php?title=Git&oldid=151585023.

[15] Wikipedia contributors, *GitHub — Wikipedia, The Free Encyclopedia*, [Online; accessed 15-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=GitHub&oldid=1165215037.

[16] Render, *Rdender cloud*. dirección: https://render.com/docs/web-services.

[17] Wikipedia contributors, *Multitenancy — Wikipedia, The Free Encyclopedia*, [Online; accessed 15-July-2023], 2023. dirección: https://en.wikipedia.org/w/index.php?title=Multitenancy&oldid=1140001478.

6.1. Referencias de figuras externas

[Fig. 1] ***Palmacedar:*** *https://palmacedar.com/apis-and-how-they-are-making-the-world-a-better-place-1of-2*

[Fig. 2] ***OPC Router:*** *https://www.opc-router.com/what-is-rest/*

[Fig. 3] ***Midium:*** *https://medium.com/codestorm/best-practices-for-rest-api-development-8434352f3213*

[Fig. 4] ***Wikipedia - JSON:*** *https://en.wikipedia.org/wiki/JSON*

[Fig. 5] ***JSON:*** *https://www.json.org/json-en.html*

[Fig. 6] ***Bajrangi Tech:*** *https://bajrangitech.com/html/l*

[Fig. 7] ***Wikipedia - HTML5:*** *https://en.wikipedia.org/wiki/HTML5*

[Fig. 8] ***ENIUM:*** *https://www.eniun.com/html5-estructura-basica-elementos-semanticos/*

[Fig. 9] ***Wikipedia - CSS3:*** *https://en.wikipedia.org/wiki/CSS*

[Fig. 10] ***Platzi:*** *https://platzi.com/clases/2008-html-css/31071-anatomia-de-una-regla-de-css/*

[Fig. 11] ***Alura:*** *https://www.alura.com.br/artigos/operadores-matematicos-em-javascriptl*

[Fig. 12] ***Wikipedia - Bootstrap:*** *https://es.wikipedia.org/wiki/Bootstrap_(framework)*

[Fig. 13] ***SEEKELOGO:*** *https://seeklogo.com/vector-logo/332789/pythonl*

[Fig. 14] ***Wikimedia COMMONS:*** *https://commons.wikimedia.org/wiki/File:Flask_logo.svg*

[Fig. 15] ***quintagroup:*** *https://quintagroup.com/cms/python/images/sqlalchemy-logo.png/view*

[Fig. 16] ***StickPNG:*** *https://gerardorenteria.blog/2023/07/16/visual-studio-code-version-updated/*

[Fig. 17] ***LINUXIAC:*** *https://linuxiac.com/sqlite-db-browser/l*

[Fig. 18] ***SINGULAR:*** *https://www.sngular.com/es/postman-i-comenzando-a-explorarlo/*

[Fig. 19] ***REQRES:*** *https://reqres.in/*

[Fig. 20] ***Wikipedia - Git:*** *https://es.wikipedia.org/wiki/Archivo:Git-logo.svg*

[Fig. 21] ***Vecteezy - GitHub:*** *https://www.vecteezy.com/vector-art/17119660-github-logo-git-hub-icon-with-text-on-white-and-black-background*

[Fig. 22] ***Blog SAP****: https://blogs.sap.com/2022/08/27/fundamentals-of-multitenancy-in-sap-btp/*

[Recurso] ***SB Admin Template - Bootstrap:*** *https://startbootstrap.com/theme/sb-admin-2*