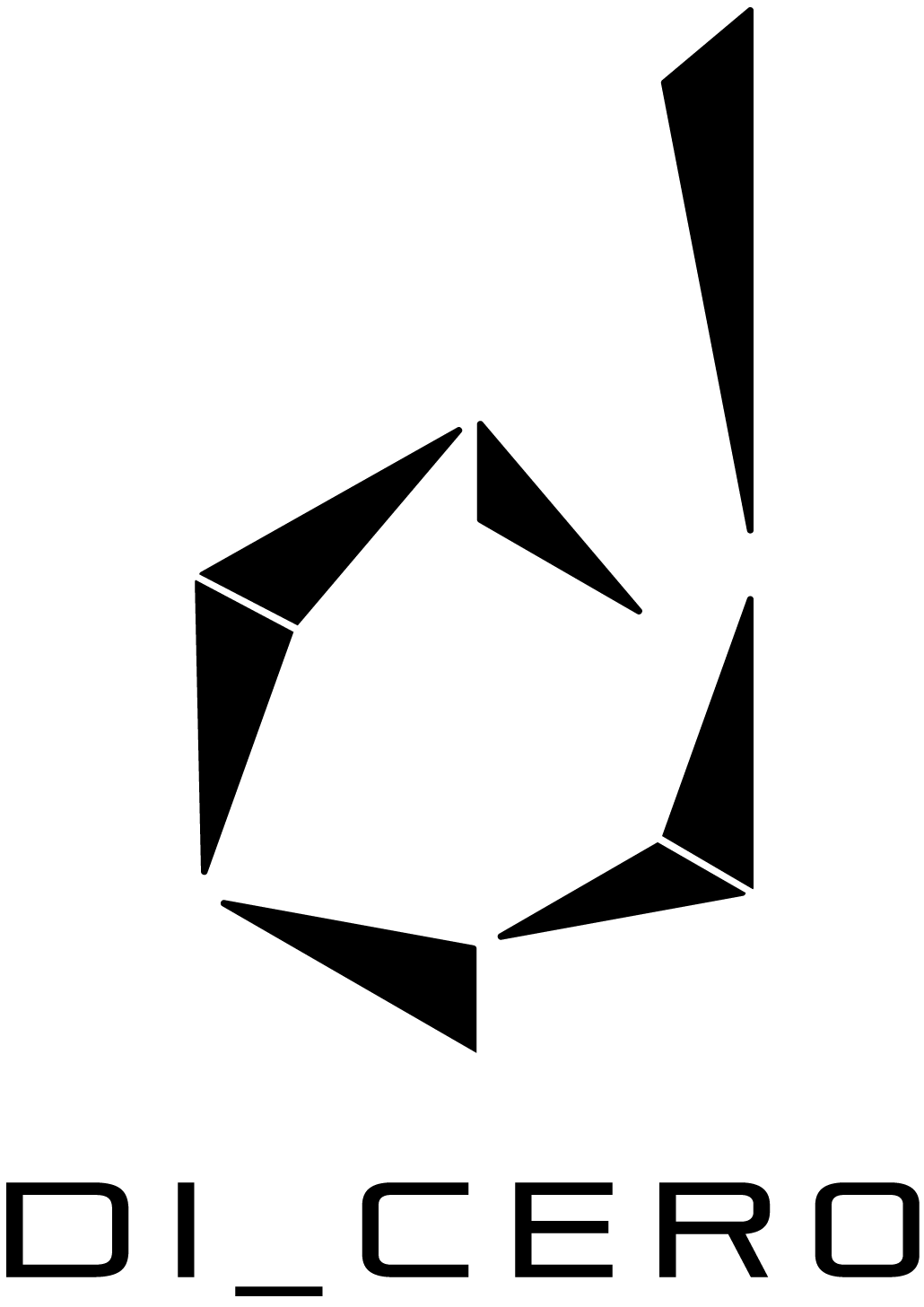
INGENIERÍA MECATRÓNICA



Diego Cervantes Rodríguez

Programación: Desarrollo Backend

Postman, Insomnium, Python, etc.

Introducción a la

Programación Backend

Contenido

[Backend Developer 2](#_Toc198850292)

[Backend, Frontend y Fullstack Developer 2](#_Toc198850293)

[Cómo se Construye el Backend: **Concepto de API** 4](#_Toc198850294)

[HTTP: HyperText Transfer Protocol 4](#_Toc198850295)

[Estructura REST: API con Protocolo HTTP 6](#_Toc198850296)

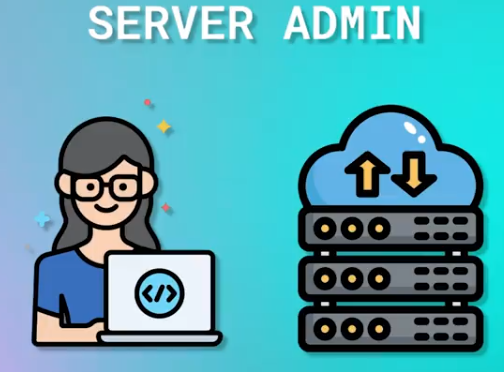
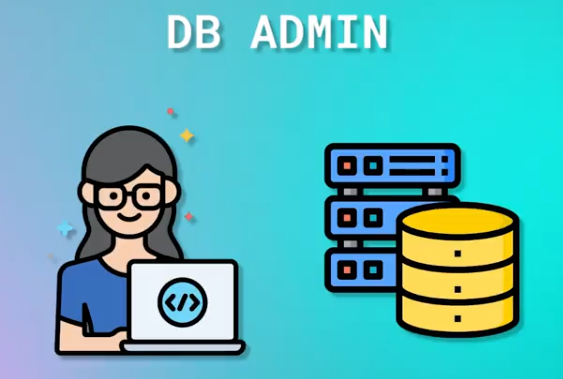
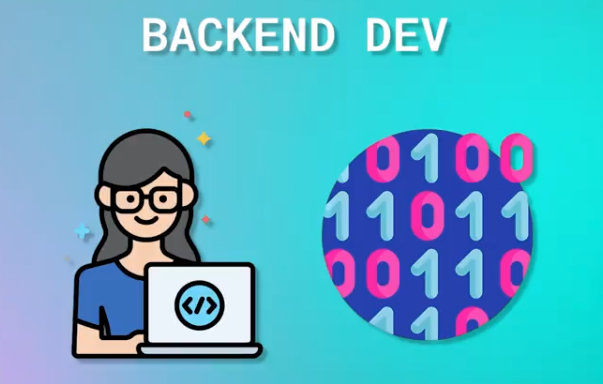
[API Testing: **Insomnia y Postman** 7](#_Toc198850297)

[Referencias 13](#_Toc198850298)

# Backend Developer

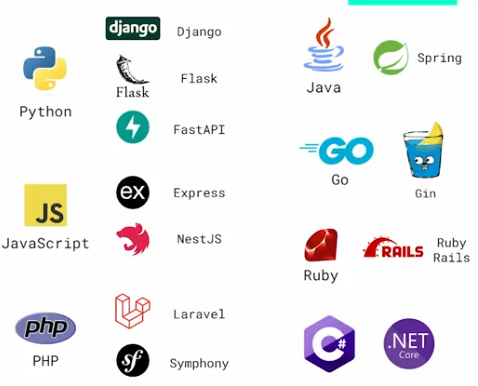
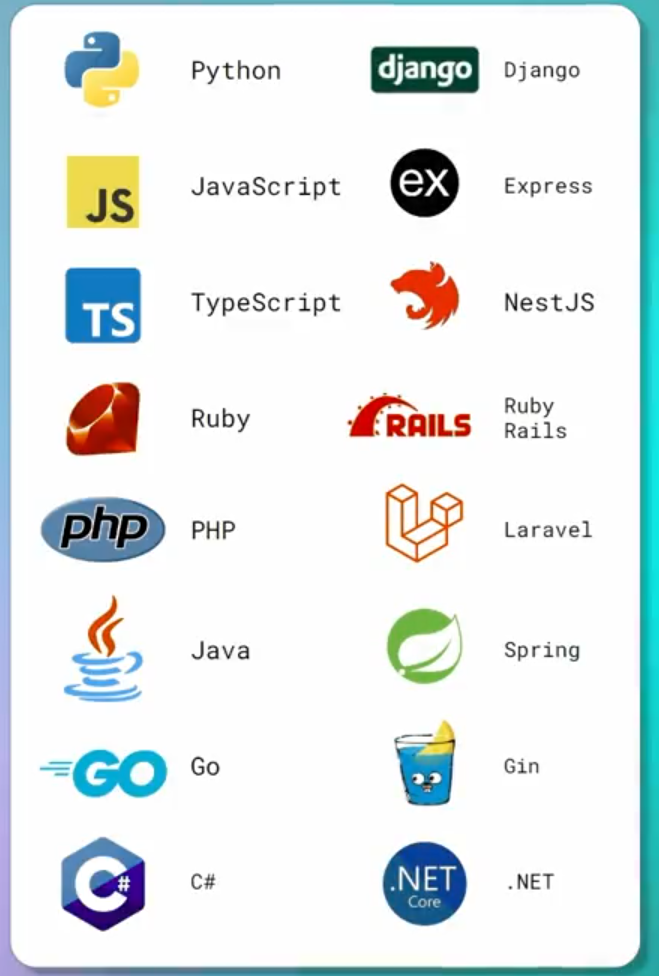
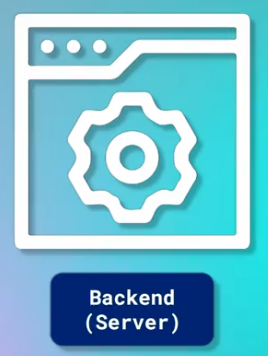
El rol principal del backend developer es el de escribir código que tenga que ver con reglas de negocio, ya sea validación, autorización de usuarios, conexión a bases de datos, manejo de datos, etc. El cual se montará sobre un servidor y será expuesto a millones de usuarios. Sin embargo, como backend también se puede adoptar los roles de:

* **DB Admin:** Este se encarga de gestionar una base de datos, tomando en cuenta sus políticas, disposición, seguridad, etc.
* **Server Admin:** Este se encarga de gestionar la seguridad de los servidores donde corre el código del backend.

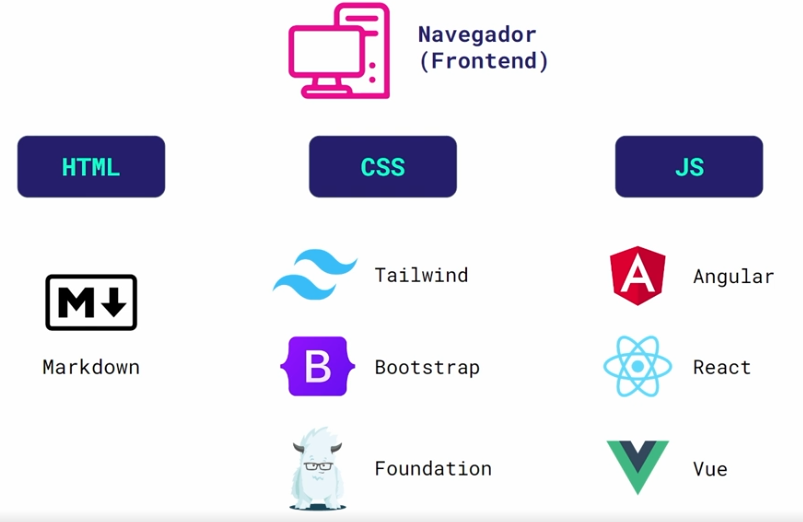
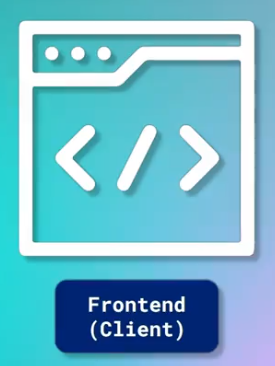


# Backend, Frontend y Fullstack Developer

* **Bakend Developer:** Desarrolla y pone a disposición **servicios** en los cuales los **clientes** **(plataformas frontend como páginas web, aplicaciones móviles, microcontroladores, etc.)** empiezan a conectarse para **extraer datos**.
  + **Los lenguajes más populares y sus frameworks para escribir servicios de backend en un servidor son**: Python (Django o FastAPI), JavaScript (Express de Node.js), TypeScript (NestJS), PHP (Laravel), Java (Spring), C# (.NET), Go (Gin), Ruby (Ruby on Rails), etc.



* **Frontend Developer:** Se enfoca en el área del **renderizado**, esto se refiere a la conexión con un servicio desplegado por el backend de una aplicación para poder **dibujar datos u otras cosas en pantalla**. Los diferentes clientes frontend pueden ser:
  + **Navegadores:** Es el más popular, ya que se encarga de conectarse a un servicio de backend que está montado en un servidor para mostrar los datos HTML que se despliegan (renderizan) en un sitio web. Los navegadores soportan 3 tipos de datos que puede devolver el servicio de un backend, los cuales son:
    - **HTML o Markdown:** Como backend developer se podrá crear un servicio que renderice un código HTML para que muestre diferentes datos en el cliente frontend o para hacer blogs se pueden renderizar de igual forma archivos markdown que sirven para creación de textos con formato.
    - **CSS:** Es código de estilo para la estructura HTML de un sitio web, este cuenta con librerías como Bootstrap, Tailwind, Foundation, etc. Para realizar de forma más sencilla el diseño de una página con código CSS.
    - **JavaScript:** Este se utiliza para proporcionar con efectos o dinamismo a un sitio web, los cuales son utilizados más que nada con frameworks como Angular, React.js, Vue.js, etc. Para solicitar información al backend y renderizarla.
  + **Smartphones:** Los cuales extraen datos para mostrar las aplicaciones móviles de los celulares. Los teléfonos soportan 3 tipos de datos que puede devolver el servicio de un backend, los cuales son:
    - **iOS Nativo:** Estos trabajan con código Swift u Objective C y solo pueden ser utilizados en sistemas operativos de Apple.
    - **Android Nativo:** Estos pueden recibir código Java o Kotlin y solo pueden ser utilizados en dispositivos con sistema operativo Android.
    - **Cross Platforms:** Estos pueden recibir ya sea código de React Native, Flutter o .NET MAUI, lo interesante de este método es que se puede utilizar en dispositivos con sistema operativo de Apple o Android, pero tiene algunas desventajas como el no poder realizar de forma tan sencilla el uso de sensores como GPS, Cámaras, etc.
  + **Microcontroladores:** Los cuales pueden crear dispositivos IoT que se conecten y manden o extraigan datos de algún servidor o base de datos.

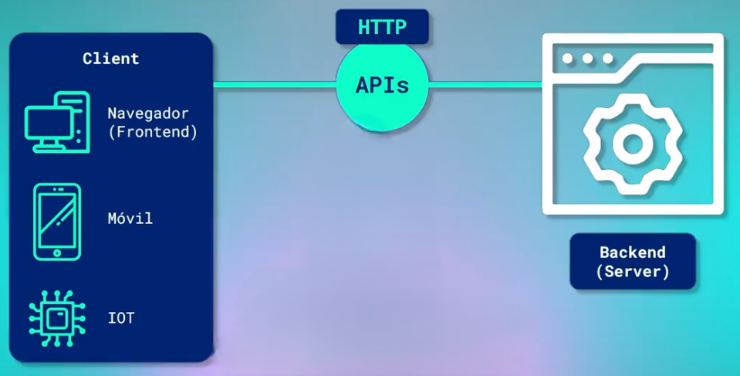
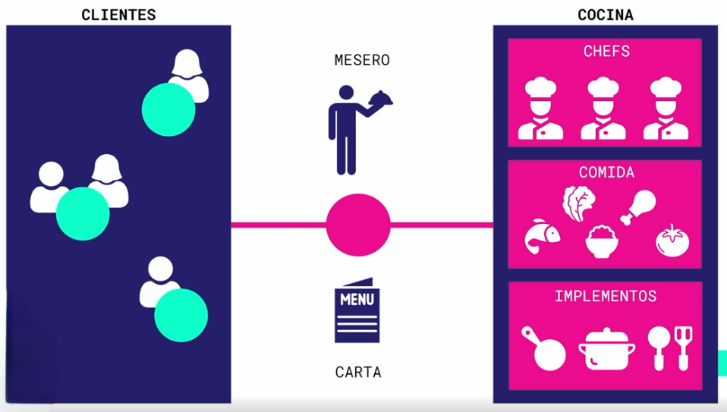


## Cómo se Construye el Backend: **Concepto de API**

Una vez que se tiene el cliente (frontend) y el servicio (backend), lo que se hace para conectarlos es crear una API (Application Programming Interface), la cual se comunica por algo llamado métodos HTTP.

Para explicar el funcionamiento de las APIs, se utiliza una analogía de un mesero, donde se cuenta con uno o varios comensales (cliente frontend) y la cocina (servicio backend) de un restaurante:

* **Comensales:** Estos representan cada navegador, teléfono o microcontrolador utilizado por un usuario, que simboliza un cliente frontend, el cual va a solicitar utilizar el servicio proporcionado por el backend (cocina).
* **Carta o Menú:** Esto representa la pantalla del sitio web, aplicación móvil o interfaz IoT con la cual el usuario (navegador, teléfono o microcontrolador) puede ver el listado de funciones que puede solicitar.
* **Mesero:** Este representa la API, que se encarga de recoger la solicitud del comensal ya que haya visto la carta y pedir que se prepare esto a la cocina (el backend).
* **Cocina:** Esto representa el backend completo del servicio, el cual utiliza las siguientes partes para realizar su función.
  + **Chefs:** Endpoints de desarrollo, estos son un listado de URLs que reciben y devuelven un tipo de dato en específico cada vez que sean utilizados.
  + **Almacén de Comida:** Esto representa las bases de datos, las cuales ya cuentan con datos que hayan sido recabados del frontend, sean llenadas de forma manual o se encuentren en un data warehouse para responder a las solicitudes de los clientes.
  + **Implementos, Cubiertos o Herramientas:** Estos representan las librerías con las que cuenta cada lenguaje de programación para permitirnos conectarnos a bases de datos, realizar autentificaciones de usuarios, etc.



En resumen, las APIs nos permiten comunicarnos con código backend comunicarnos con otra parte de un mismo sistema o con otro sistema con el protocolo HTTP haciendo uso de endpoints para recibir como valor de retorno un código que renderice el frontend o datos en formato JSON o XML.

### HTTP: HyperText Transfer Protocol

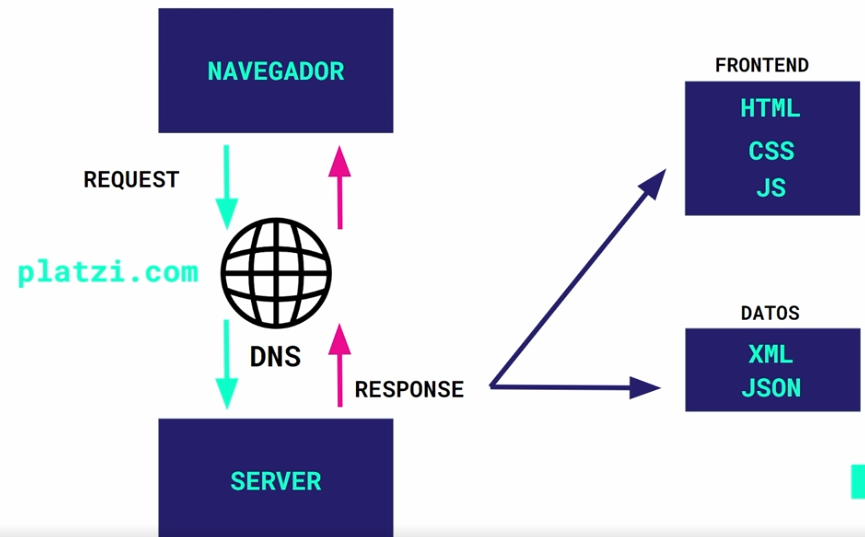
El protocolo HTTP se conforma de una URL que nos ayuda a explorar y exponer servicios web, es la forma en la que el frontend y el backend interactúan entre sí, el cual se conforma de las siguientes partes:

* **Protocolo:** Esta es la parte del URL que indica el protocolo que se está utilizando, ya sea ftp (para transmisión de archivos), http o https (que contiene una capa de seguridad).
* **Dominio:** Esta parte se compra a través de un servicio de host, que nos provee con un servidor DNS, el cual nos permite tener la propiedad de una dirección IP, la cual por motivos de facilidad se ve como una palabra en vez de números, llamado dominio y es como la gente podrá acceder a cada sitio web en específico.
* **Ruta o Endpoint:** Este va separado del dominio por medio de un símbolo de slash (/)
  + **Ruta:** Si nos estamos refiriendo a una ruta nos permitirá acceder a cada carpeta que contenga una imagen, archivo, etc.
  + **Endpoint:** Si nos referimos a un endpoint, este permite acceder a un dato en específico que es expuesto a través del backend en formato JSON o XML para que el frontend pueda ser renderizado o actualizado o a una nueva plantilla de frontend, la cual devolverá en su repuesta un código que pueda ser interpretado por el cliente, ya sea HTML, CSS, JavaScript, Java, Kotlin, Swift, etc.



* **Status HTTP:** Hay algo muy interesante de los protocolos HTTP y son los códigos de estado (status), estos nos dan información para saber el estado en el que se encuentra la petición que realizó el cliente hacia el servicio. El significado de estos se encuentra categorizado por rango, el cual será explicado a continuación:
  + **100-199:** Estos indican información de estado del servidor hacia el cliente, que puede ser como la 102 = Procesando datos.
  + **200-299:** Este es el rango más utilizado porque denota éxito en alguna acción del servidor, por ejemplo 200 = Ok, 201 = Algo fue creado de forma exitosa, 204 = La acción fue exitosa pero no retornó ningún dato, etc.
  + **300-399:** El rango de los 300 indica que algún recurso ha sido movido de lugar, por ejemplo 301 = El recurso que antes estaba aquí, fue movido de lugar, 307 = El recurso fue temporalmente movido, 308 = El recurso fue permanentemente movido, etc.
  + **400-499**: Representan errores de la parte del cliente, esto normalmente ocurre porque el cliente envió una solicitud de forma errónea, por ejemplo 401 = No autorizado, 404 = No encontrado, 409 = Existe conflicto porque los datos están siendo mandados de forma incorrecta.
  + **500-599:** Representan errores que ocurren de parte del servidor, del código que está ejecutándose, por ejemplo 500 = Error interno del servidor, 502 = Bad Gateway, 504 = El tiempo de espera del servidor se ha sobrepasado.

Para explicar todos estos tipos de respuesta del servidor en forma divertida con memes se cuenta con el sitio: [HTTP Cats](https://http.cat/)

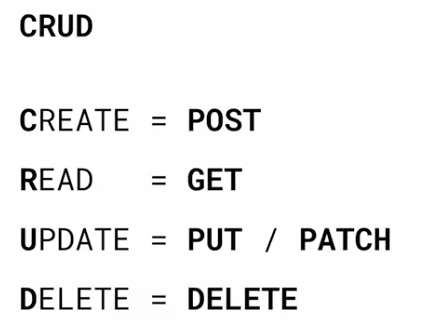


### Estructura REST: API con Protocolo HTTP

El estándar o estructura REST API se utiliza para desarrollar APIs que funcionen a través del protocolo HTTP, es por eso que es la más utilizada. La forma en la que se piden recursos o datos a los endpoints de las REST APIs es a través de su dominio y normalmente devuelven su resultado en formato JSON, que luego serán procesador por un cliente web, smartphone o un microcontrolador.

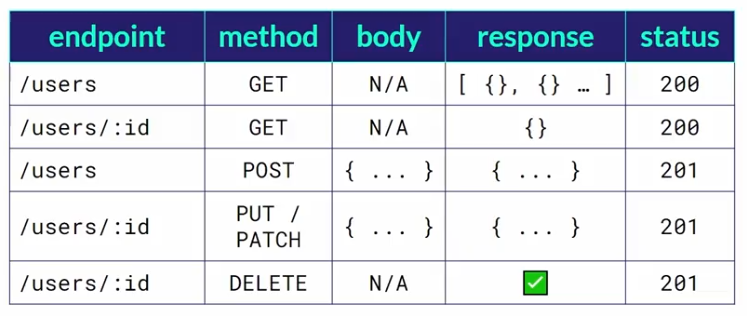
La forma en la que funcionan las REST API es proporcionando como mínimo métodos CRUD que permitan crear, leer, actualizar o borrar datos de un database a través de sus endpoints:

* **Create =** Método **Post** del protocolo HTTP.
* **Read =** Método **Get** del protocolo HTTP.
* **Update =** Métodos **Put/Patch** del protocolo HTTP.
* **Delete =** Método **Delete** del protocolo HTTP.

Cada que se cree un endpoint se debe indicar:

1. El nombre del endpoint.
2. Cuál es el método HTTP que se ejecuta al acceder a dicho endpoint
3. El body que contiene los datos que utilizará el método HTTP para realizar su función.
4. La respuesta que se obtendrá del servidor al utilizar el endpoint.
5. El número de status HTTP que se espera recibir.



## API Testing: **Insomnia y Postman**

Utilizaremos la herramienta de Insomnia y para testear una API de prueba de Platzi que se encuentra en el siguiente dominio: <https://fakeapi.platzi.com/>

Esta API lo que muestra es una tienda en línea de mentira, la cual tiene productos, imágenes de productos, precios, categorías, etc. y utiliza una arquitectura REST, para utilizarla veremos su documentación y utilizaremos algunos de sus endpoints para ver cómo funciona la herramienta:

En las documentaciones de APIs lo que podremos ver será:

* **Método HTTP que se utiliza +** el dominio **+** **el nombre del endpoint**.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

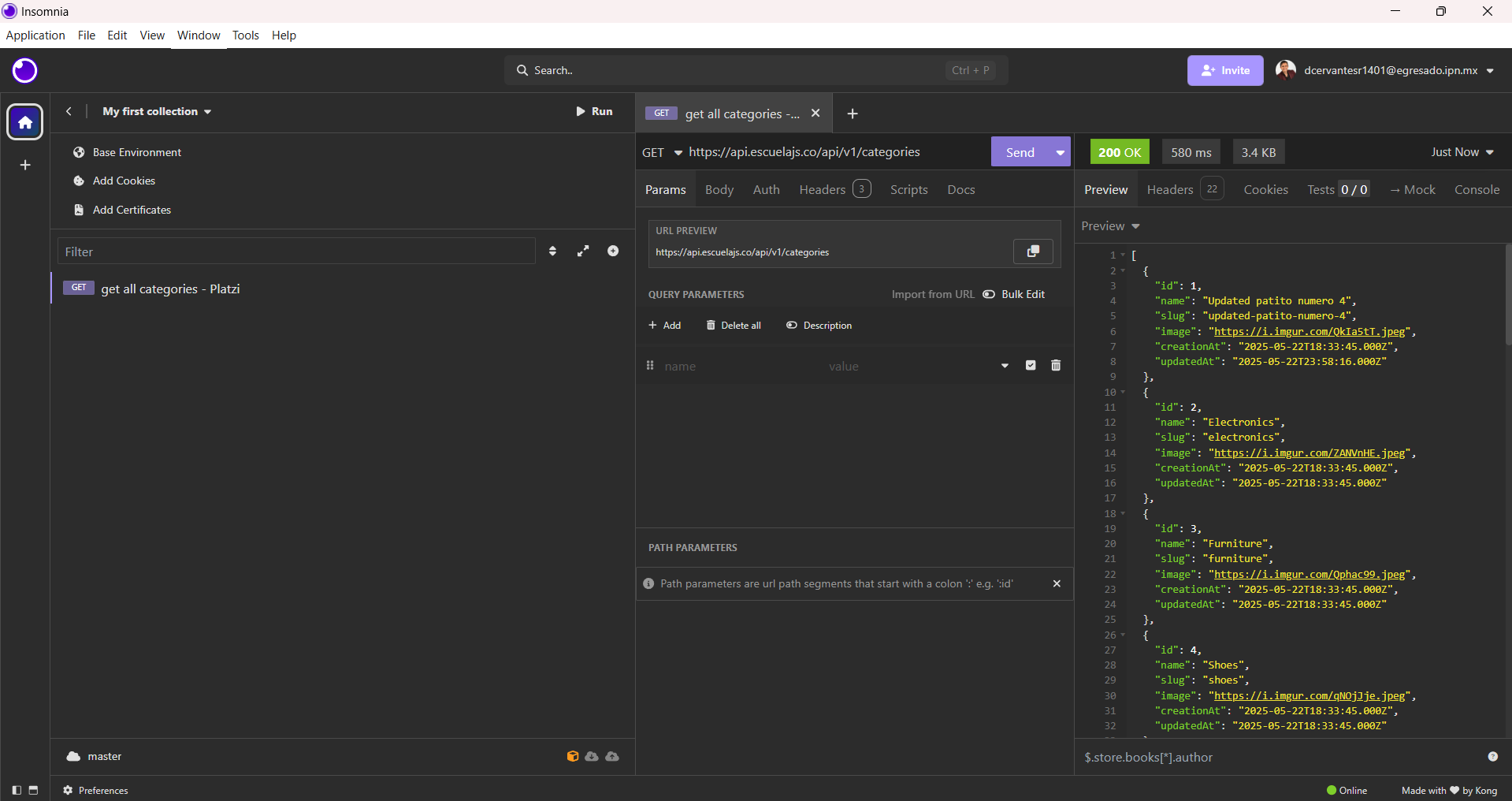
El software de Insomnia puede ser descargado del siguiente enlace: <https://insomnia.rest/download>

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En la interfaz de Insomnia se tiene lo siguiente:

* Del lado izquierdo superior podemos ver el environment, colección o documento en el que nos encontramos, lo cual se refiere a
  + Debajo y del lado izquierdo podemos ver toda la lista de endpoints a los que podemos acceder. Cada vez que utilicemos uno nuevo, podemos asignarle un nombre y guardarlo en nuestra colección de endpoints.
    - Los endpoints guardados indicarán el método HTTP que utilizan y su nombre asignado.
      * Para crear un nuevo request y por lo tanto, guardar un nuevo endpoint, debemos dar clic en el botón de + que se encuentra al lado derecho del filtro de requests.
    - De igual forma estos pueden ser categorizados en folders.
* En la parte media superior podemos introducir la siguiente estructura del endpoint:
  + **Método HTTP que se utiliza +** el dominio **+** **el nombre del endpoint**.
  + Abajo del endpoint se pueden introducir sus siguientes características:
    - **Params:** Se utiliza para ver la endpoint completa y si se quieren añadir elementos como un nombre y un valor a la URL.
    - **Body:** Es el cuerpo del mensaje que se quiere mandar, osea los datos que espera, esto se utiliza solamente cuando se ejecutan comandos HTTP POST, PUT, PATCH o DELETE.
      * En esta parte se puede escoger el formato del body, ya sea en formato JSON, XML, etc.
    - **Auth:** Es el método de autenticación, para validación de usuarios o seguridad.
    - **Headers:** Es el encabezado del body del mensaje que se manda a la API.
    - **Scripts:** Para poder escribir código backend que recabe datos.
    - **Docs:** Es la documentación del endpoint.
* En la parte derecha, podemos ver la respuesta de la solicitud que se realiza hacia cada endpoint de la API.
  + En la parte superior izquierda de la sección derecha podemos ver:
    - El HTTP status code que devolvió la solicitud.
    - El tiempo que se tardó en recuperarla.
    - El peso de la respuesta dada en bits de memoria.
  + Debajo de la franja de estado de la solicitud, se encuentran opciones donde se puede ver un preview de la respuesta, solo su header, las cookies que retornó y su timeline.

 Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto. Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de un celular

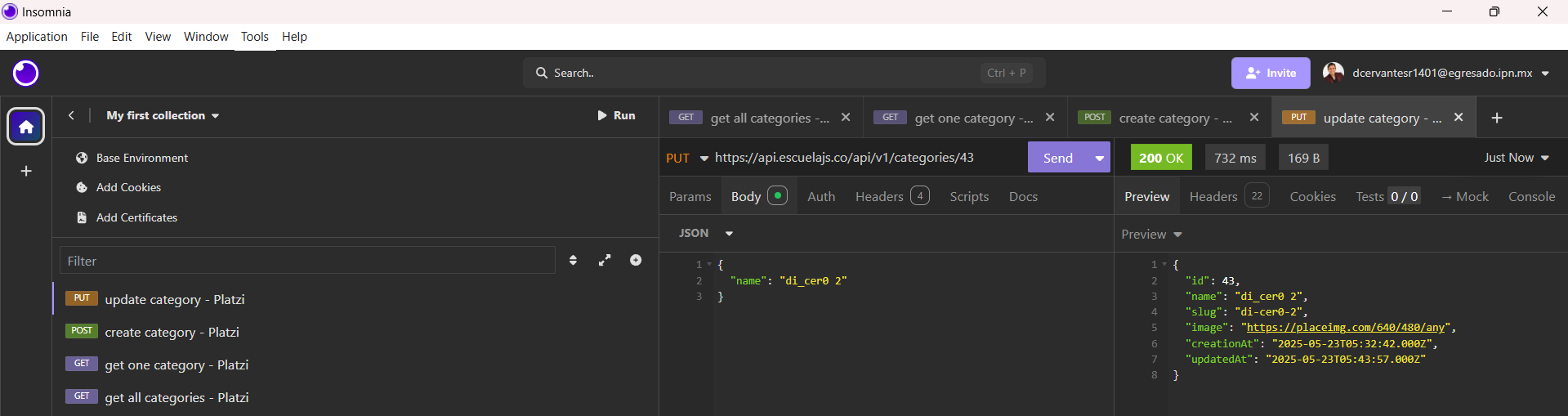
El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

 Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

La diferencia entre los métodos HTTP PUT y PATCH es que PATCH te permite cambiar solo uno de los datos del JSON, mientras que PUT te obliga a mencionarlos todos, aunque el valor sea el mismo en algunos y solo cambien otros.

# Referencias

Platzi, Nicolás Molina, “Curso de Introducción al Desarrollo Backend”, 2018 [Online], Available: https://platzi.com/home/clases/4656-backend/56005-los-roles-del-desarrollo-backend/