ACADEMY by NUMEN

FULL STACK

Comenzamos en unos minutos

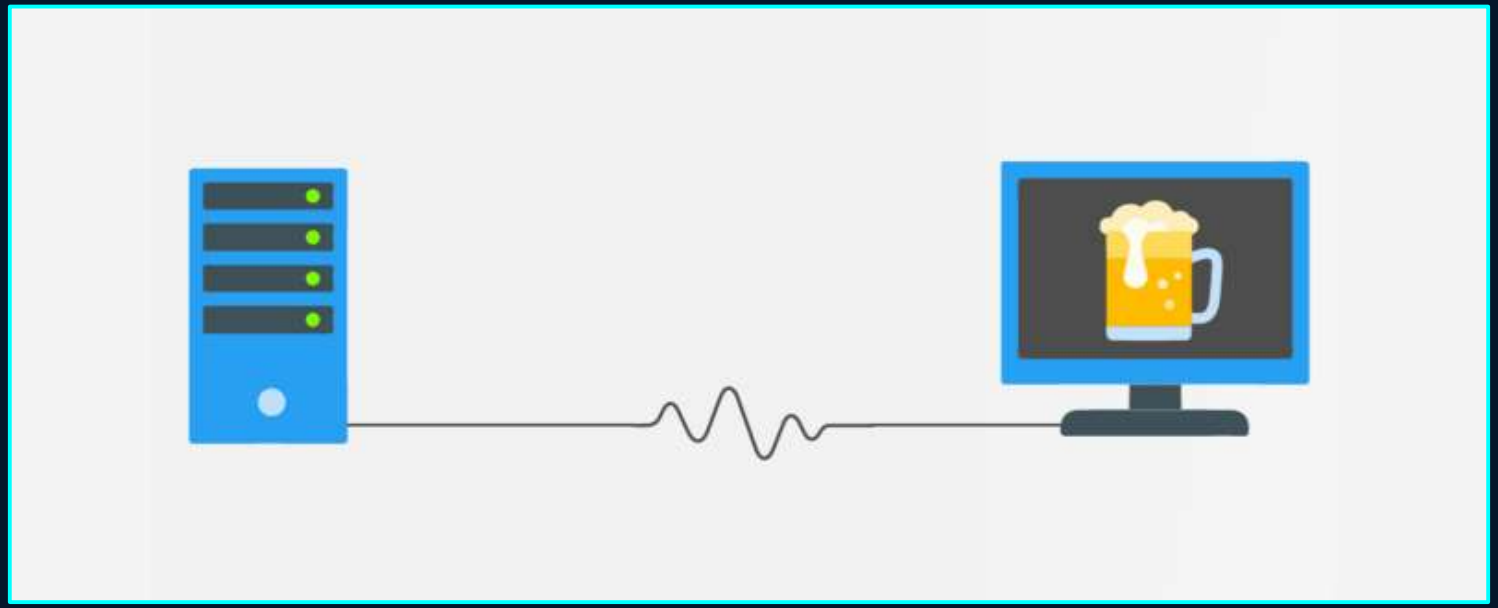

ACADEMY
by NUMEN

Javascript: Clase 8
API REST y CRUD

ACADEMY by NUMEN

¿Qué es una API?

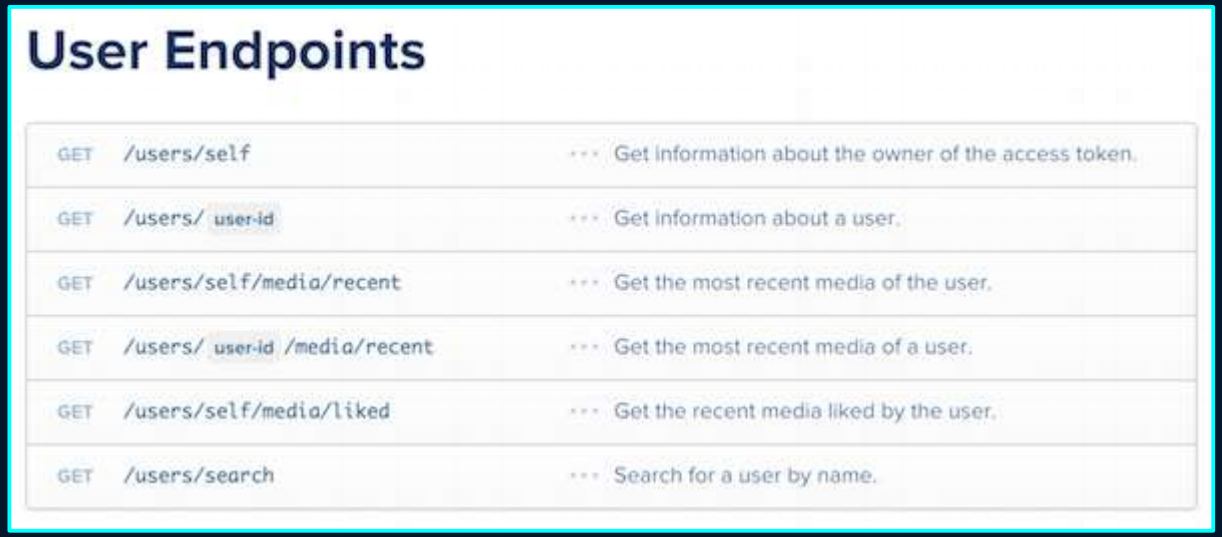
Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.



¿Que es un end point?

Un endpoint es cualquier dispositivo que sea físicamente la parte final de una red. Las computadoras de escritorio, las tablets, los smartphones, los dispositivos de oficina de red, como los routers, las impresoras y las cámaras de seguridad también son considerados endpoints.
Los servidores también pueden ser considerados endpoints porque también están conectados a la red.

Básicamente cualquier dispositivo final conectado a la red es un endpoint.

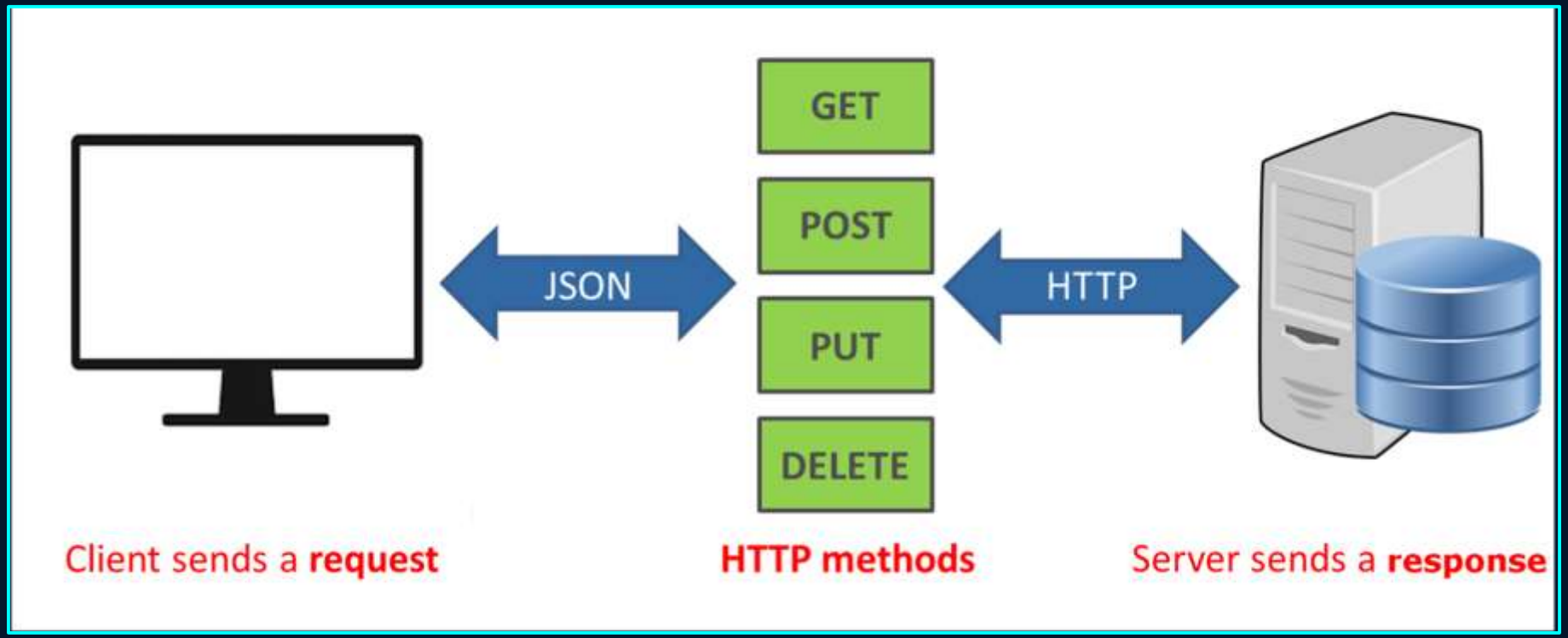


REST

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad.

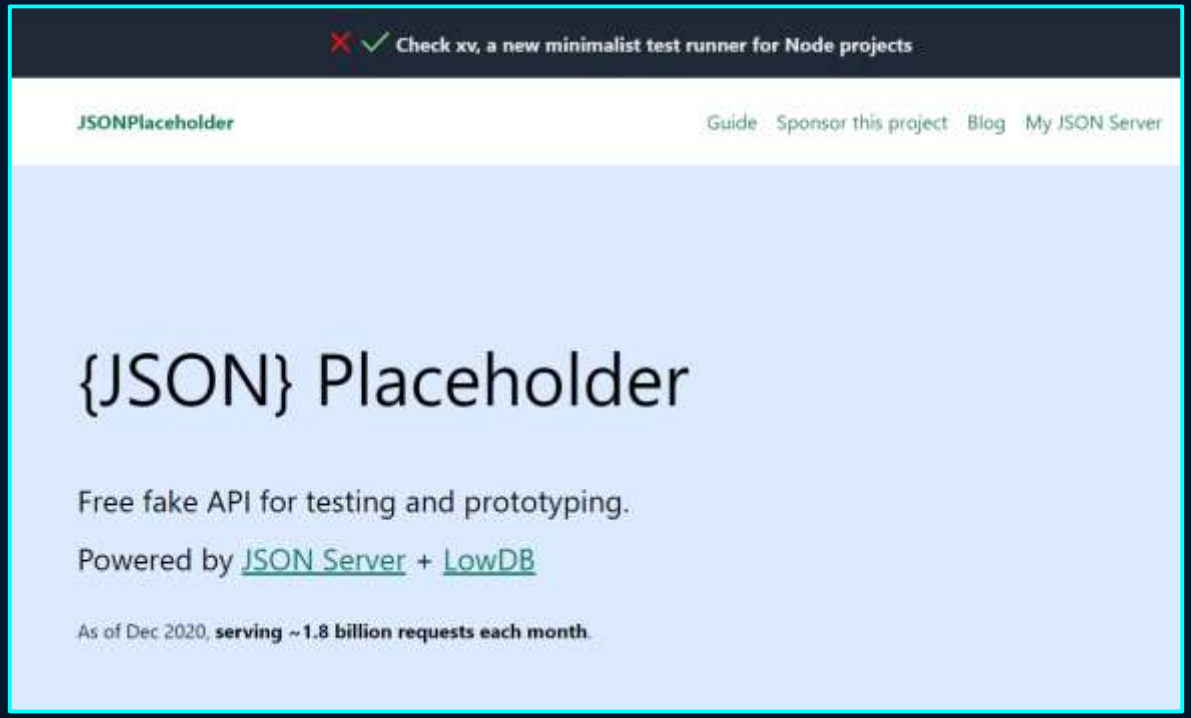
A veces es preferible **una solución más sencilla de manipulación de datos como REST**.

Arquitectura Rest



¿Que es CRUD?

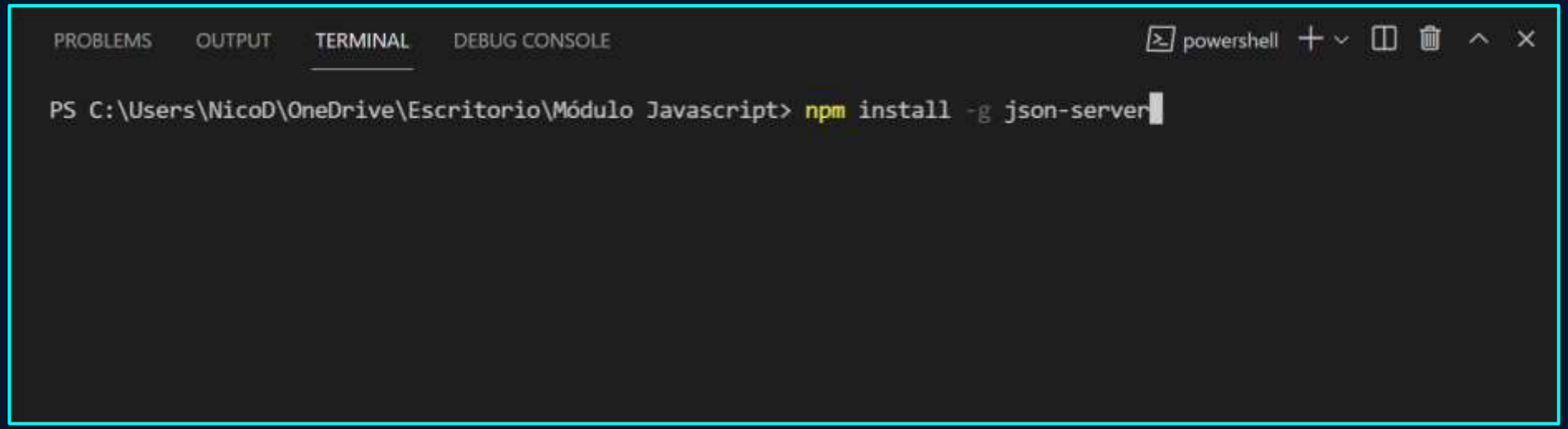
JSON Server



Enlace: https://github.com/typicode/json-server

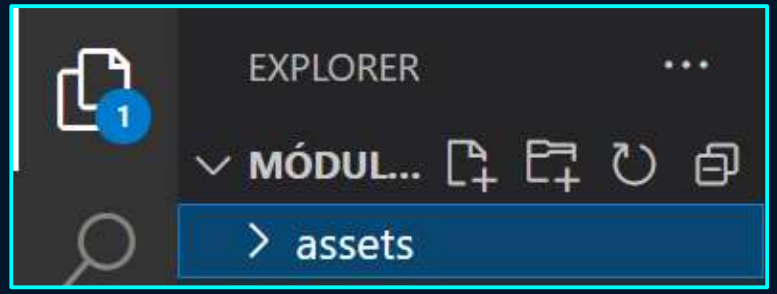
Instalando JSON Server

Teniendo instalado Node, ejecutamos el siguiente comando para instalar JSON Server en nuestro proyecto.

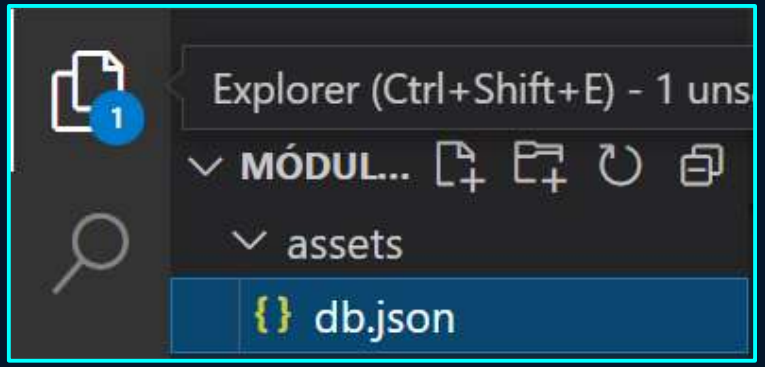


Conectando nuestra base de datos con JSON Server

Paso 1: Creamos una carpeta llamada assets



Paso 2: Dentro de assets creamos un archivo json llamado db.json con la información que deseamos manipular.



Paso 3: Abrimos la terminal y escribimos el siguiente comando

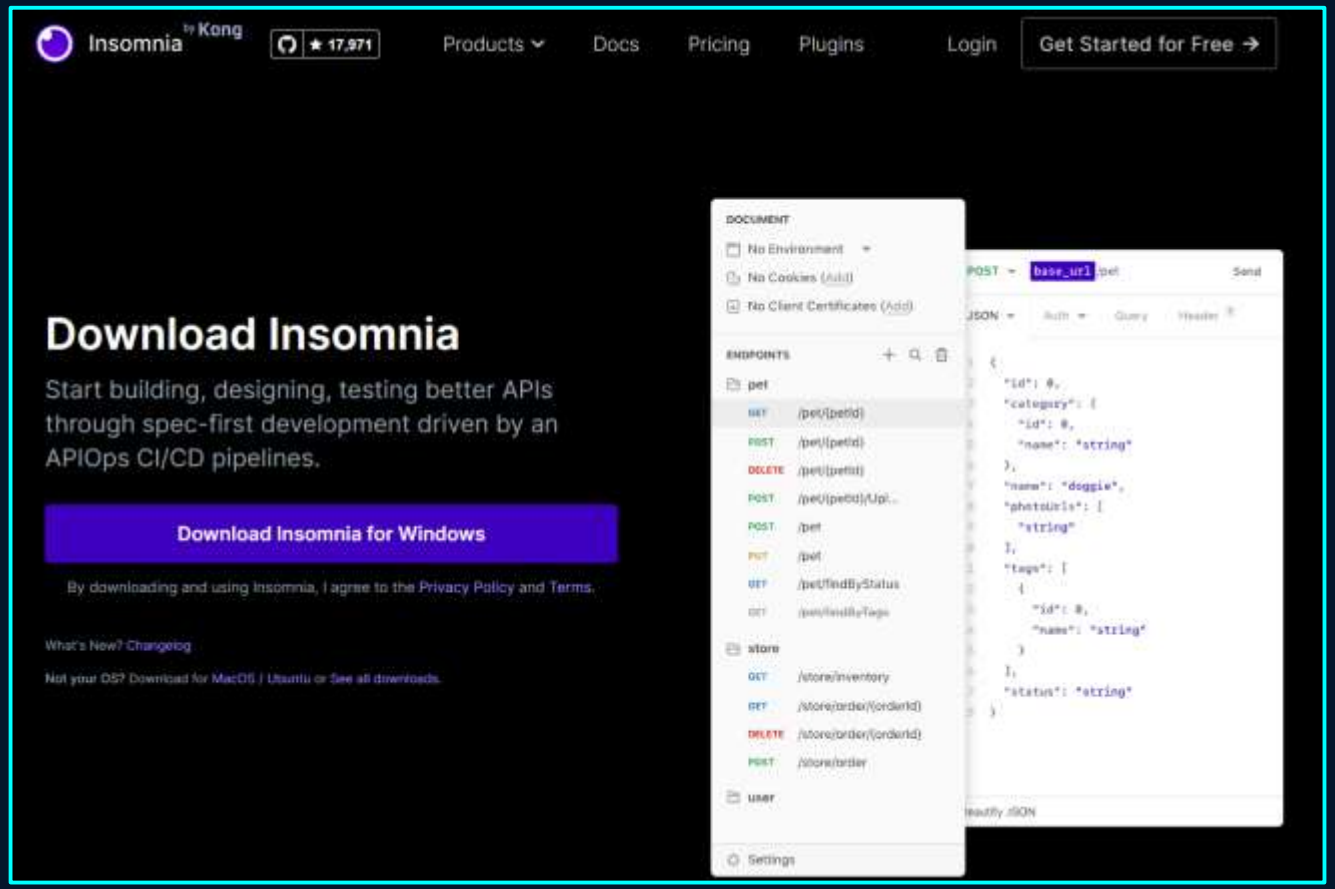
```
PS C:\Users\NicoD\OneDrive\Escritorio\Módulo Javascript> json-server --watch db.json
```

El Arreglo de objeto que utilizaremos en db.json

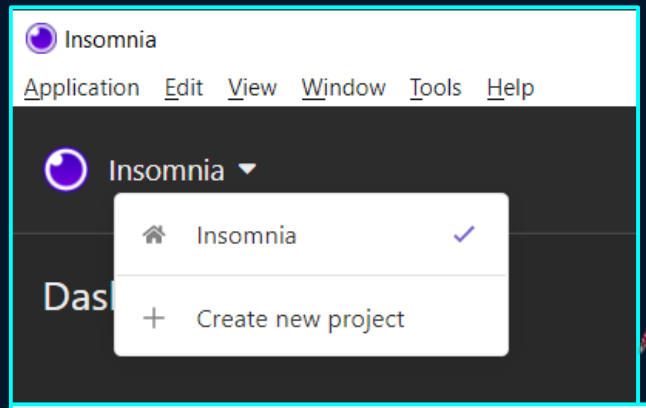
```
{
    "pokemons":[
        {
            "id": 1,
            "name": "Bulbasaur",
            "type": ["Grass", "Poison"]
        },
        {
            "id": 2,
            "name": "Ivysaur",
            "type": ["Grass", "Poison"]
        },
        {
            "id": 3,
            "name": "Venusaur",
            "type": ["Grass", "Poison"]
        },
        {
            "id": 4,
            "name": "Charmander",
            "type": ["Fire"]
        },
        {
            "id": 5,
            "name": "Charmeleon",
            "type": ["Fire"]
        },
        {
            "id": 6,
            "name": "Charizard",
            "type": ["Fire", "Flying"]
        },
        {
            "id": 7,
            "name": "Squirtle",
            "type": ["Water"]
        },
        {
            "id": 8,
            "name": "Wartortle",
            "type": ["Water"]
        },
        {
            "id": 9,
            "name": "Blastoise",
            "type": ["Water"]
        }

    ]
}
```

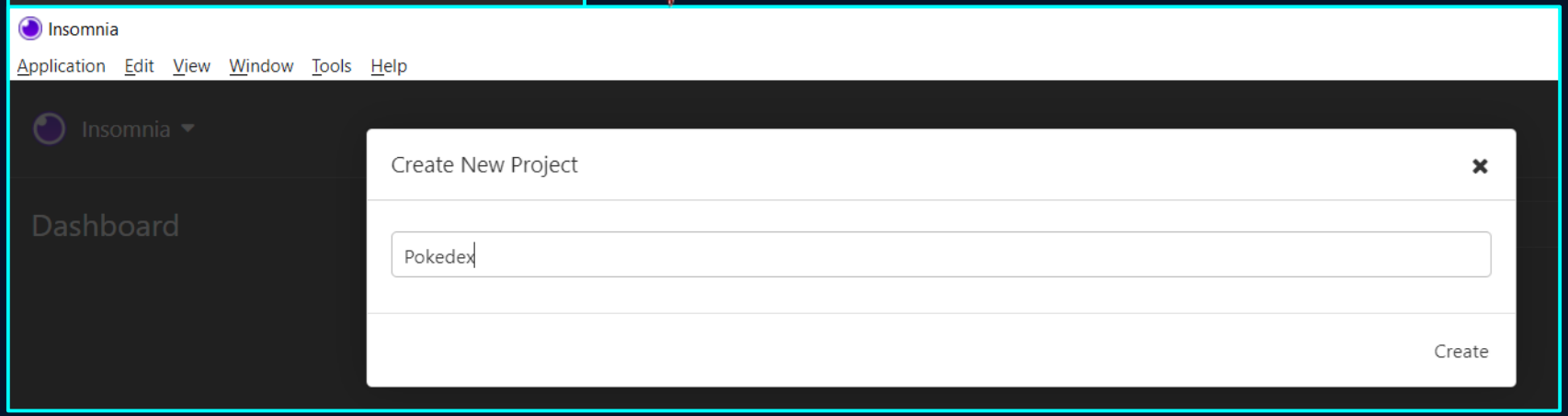
Descargando Insomnia Rest



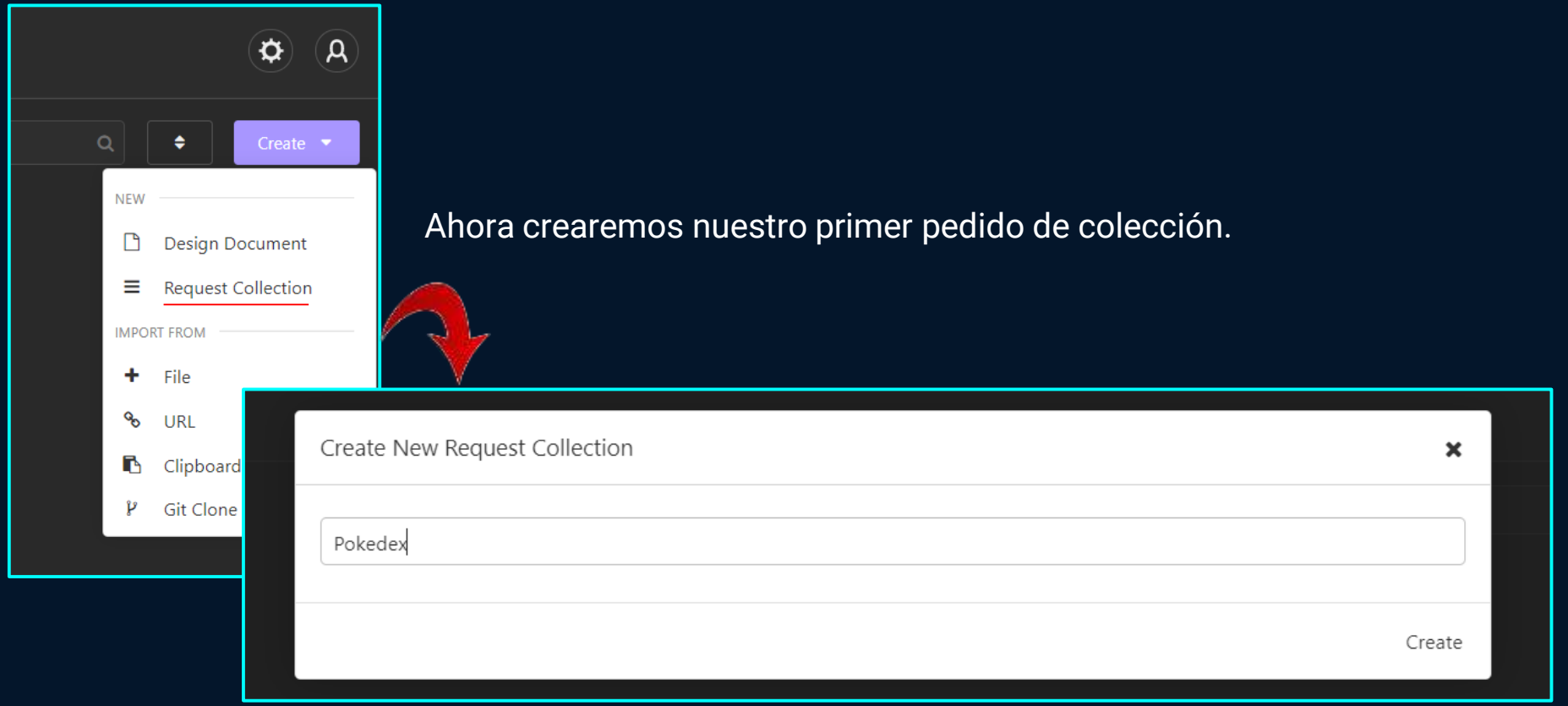
Creando nuevo proyecto



Iniciamos creando un nuevo proyecto donde tendremos nuestras colecciones pedidos de consumición a APIs.



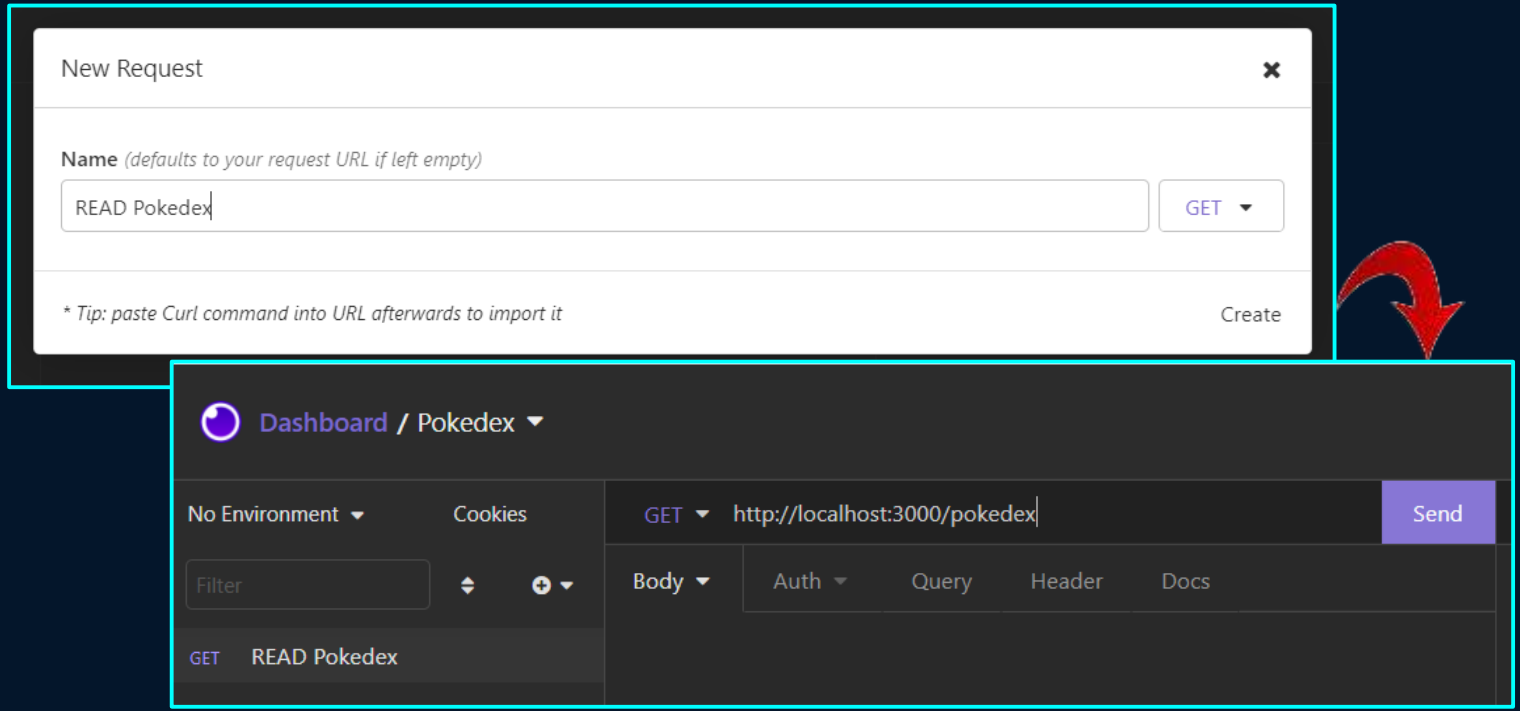
Creando una nueva colección



Ahora crearemos nuestro primer pedido de colección.

Creando una nueva colección

Ahora crearemos lo que será nuestro primer pedido de consumición a través del método GET. Una vez puesto el link de nuestro JSON Server estamos listos para consumir los datos del cliente.



ACADEMY by NUMEN

HTML de nuestro CRUD (Parte 1)

```
<h1>POKE CRUD</h1>
    <section class="crud">
        <article>
            <h2 class="crud-title">Agregar Pokemon</h2>
            <form class="crud-form">
                <input type="text" name="name" placeholder="name"
required>
                <br>
                <input type="text" name="type" placeholder="type"
required>
                <br>
                <input type="submit" value="Enviar">
                <input type="hidden" name="id">
            </form>
        </article>
```

HTML de nuestro CRUD (Parte 2 y 3)

```
        <article>
            <h2>Pokelista</h2>
            <table class="crud-table">
                <thead>
                    <tr>
                        <th>Nombre</th>
                        <th>Tipo</th>
                        <th>Acciones</th>
                    </tr>
                </thead>
                <tbody>

                </tbody>
            </table>
        </article>
    </section>
```

```
<template id="crud-template">
    <tr>
        <td class="name"></td>
        <td class="type"></td>
        <td>
            <button class="edit">Editar</button>
            <button class="delete">Eliminar</button>
        </td>
    </tr>
</template>
```

Conectado JS con nuestro documento HTML

```
// Referenciando selectores
const d = document,
      $table = d.querySelector(".crud-table"),
      $form = d.querySelector(".crud-form"),
      $title = d.querySelector(".crud-title"),
      $template = d.getElementById("crud-template").content,
      $fragment = d.createDocumentFragment();
```

Programando la consumición de datos

```
const ajax = (options) => {
    let { url, method, success, error, data } = options;
    const xhr = new XMLHttpRequest();

    xhr.addEventListener("readystatechange", e => {
    if (xhr.readyState !== 4) return;

    if (xhr.status >= 200 && xhr.status < 300) {
        let json = JSON.parse(xhr.responseText);
        success(json);
    } else {
        let message = xhr.statusText || "Ocurrió un error";
        error(`Error ${xhr.status}: ${message}`);
    }
    });

    xhr.open(method || "GET", url);
    xhr.setRequestHeader("Content-type", "application/json; charset=utf-8");
    xhr.send(JSON.stringify(data));
}
```

Programando la inserción en el DOM

```
const getAll = () => {
    ajax({
    url: "http://localhost:3000/pokemons",
    success: (res) => {
        console.log(res);

        res.forEach(el => {
        $template.querySelector(".name").textContent = el.name;
        $template.querySelector(".type").textContent = el.type;
        $template.querySelector(".edit").dataset.id = el.id;
        $template.querySelector(".edit").dataset.name = el.name;
        $template.querySelector(".edit").dataset.type = el.type;
        $template.querySelector(".delete").dataset.id = el.id;

        let $clone = d.importNode($template, true);
        $fragment.appendChild($clone);
        });

        $table.querySelector("tbody").appendChild($fragment);
    },
    error: (err) => {
        console.log(err);
        $table.insertAdjacentHTML("afterend", `<p><b>${err}</b></p>`);
    }
    })
}

d.addEventListener("DOMContentLoaded", getAll);
```

Programando el evento “submit” (parte 1 - POST)

```
d.addEventListener("submit", e => {
    if (e.target === $form) {
    e.preventDefault();

    if (!e.target.id.value) {
        //Create - POST
        ajax({
        url: "http://localhost:3000/pokemons",
        method: "POST",
        success: (res) => location.reload(),
        error: (err) => $form.insertAdjacentHTML("afterend", `<p><b>${err}</b></p>`),
        data: {
            name: e.target.name.value,
            type: e.target.type.value
        }
        });
```

Programando el evento “submit” (parte 2 - PUT)

```
    } else {
        //Update - PUT
        ajax({
        url: `http://localhost:3000/pokemons/${e.target.id.value}`,
        method: "PUT",
        success: (res) => location.reload(),
        error: (err) => $form.insertAdjacentHTML("afterend", `<p><b>${err}</b></p>`),
        data: {
            name: e.target.name.value,
            type: e.target.type.value
        }
        });
    }
    }
});
```

Programando botones de UPDATE y DELETE (parte 1)

```
d.addEventListener("click", e => {
    if (e.target.matches(".edit")) {
    $title.textContent = "Editar Pokemon";
    $form.name.value = e.target.dataset.name;
    $form.type.value = e.target.dataset.type;
    $form.id.value = e.target.dataset.id;
    }

    if (e.target.matches(".delete")) {
    let isDelete = confirm(`¿Estás seguro de eliminar el id ${e.target.dataset.id}?`);
```

Programando botones de UPDATE y DELETE (parte 2)

```
    if (isDelete) {
        //Delete - DELETE
        ajax({
        url: `http://localhost:3000/pokemons/${e.target.dataset.id}`,
        method: "DELETE",
        success: (res) => location.reload(),
        error: (err) => alert(err)
        });
    }
    }
})
```

ACADEMY
by NUMEN