

# Apuntes de DQ II y Tarea.

## ¿Que es Node.js y Express.js?

Node.js Es un entorno de ejecucion, que nos permite correr javascript fuera del navegador, esto nos permite crear microservicios e incluso Raspberry Pi, node es ideal para crear aplicaciones escalables y de alto rendimiento.

Express.js es el framework mas popular para el diseño y construccion de aplicaciones y Apis(requests), esto nos proporciona un conjunto de características para aplicaciones webs y moviles, y nos ayuda a construir del lado del servidor de una manera mucho mas facil.

- Te permite manejar **rutas** ( /login , /register , /productos , etc.).
- Facilita trabajar con **middlewares** (funciones que procesan la petición antes de llegar a la respuesta final, por ejemplo para autenticar usuarios).
- Soporta distintos métodos HTTP: GET , POST , PUT , DELETE .
- Se integra fácil con bases de datos, motores de plantillas y librerías de frontend.
- Es **ligero y flexible**, no te obliga a seguir una estructura estricta como otros frameworks más grandes (por ejemplo, Django en Python).


```
const express = require("express"); // estamos creando una constante
const app = express();
const PORT = 3000;
// Esta parte del codigo es la que hace la creacion del servidor y definimos su puerto.
// Ruta de ejemplo
app.get("/", (req, res) => {
  res.send("¡Hola Mundo!");
});

// Arrancar el servidor
app.listen(PORT, () => {
  console.log(Servidor escuchando en http://localhost:${PORT});
});
```

Los usos que tienen Node y express.js son bastante amplios, comencemos con Node.js, muchos de los servidores en tiempo real y aplicaciones de mensajería utilizan node para manejar en la parte del backend, miles de mensajes en tiempo real con websocket (Este es un protocolo utilizado para permitir la comunicación bidireccional y en tiempo real entre un cliente y un servidor), también se utiliza en los servicios de streaming para manejar gran parte de su infraestructura y su transmisión de datos, también suele utilizarse en scripts o automatización como por ejemplo los comandos de `npm run dev` , básicamente debajo de este pedazo de

codigo estamos automatizando procesos mas largos, y por ultimo en placas o sensores como por ejemplo Raspberry Pi.

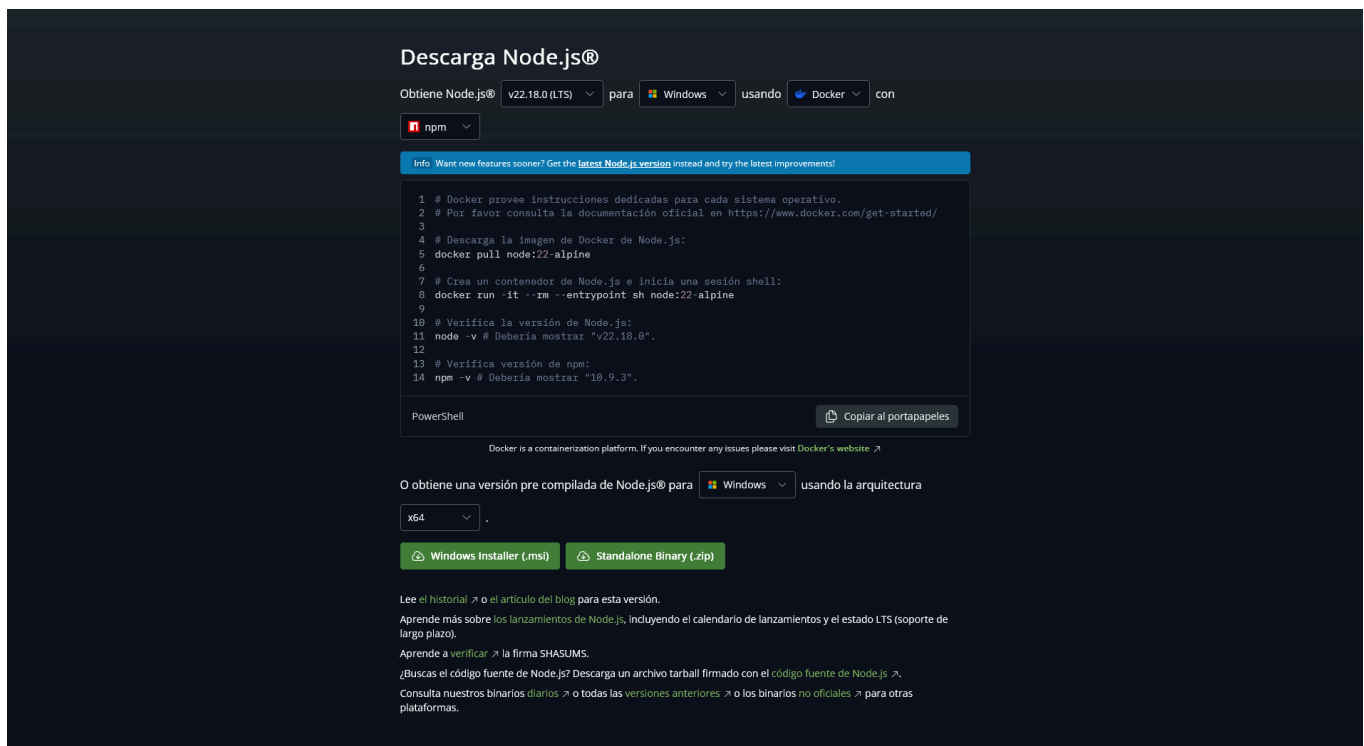
Por otro lado Express.js aparece cuando queremos crear APIs o servidores webs facilmente, express nos permite definir rutas y devolver json que va a consumir nuestro fronted como por ejemplo en este segmento:



```
1 // GET → Listar todas las tareas
2 app.get("/api/tareas", (req, res) => {
3     res.json(tareas);
4 });
```

Estamos definiendo la ruta para llamar a la API get (La cual nos permitira ver la lista de tareas desde el backend, en este caso utilicé POSTMAN para hacer los request)

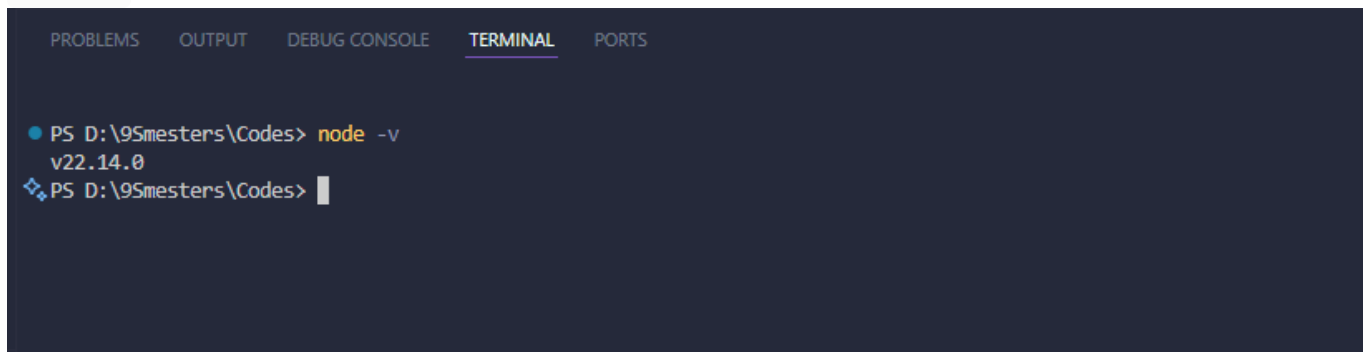
## Tutorial de instalación para Node.js y Express.js.



Descargamos e instalamos el instalador de node.js, le damos siguiente a todo y (Modificamos la ruta si lo deseamos)

luego de esto en nuestro entorno de programacion (En mi caso vscode) utilizamos el siguiente comando para verificar si se instaló correctamente.

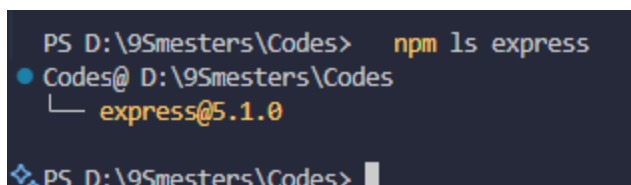
```
node -v
```



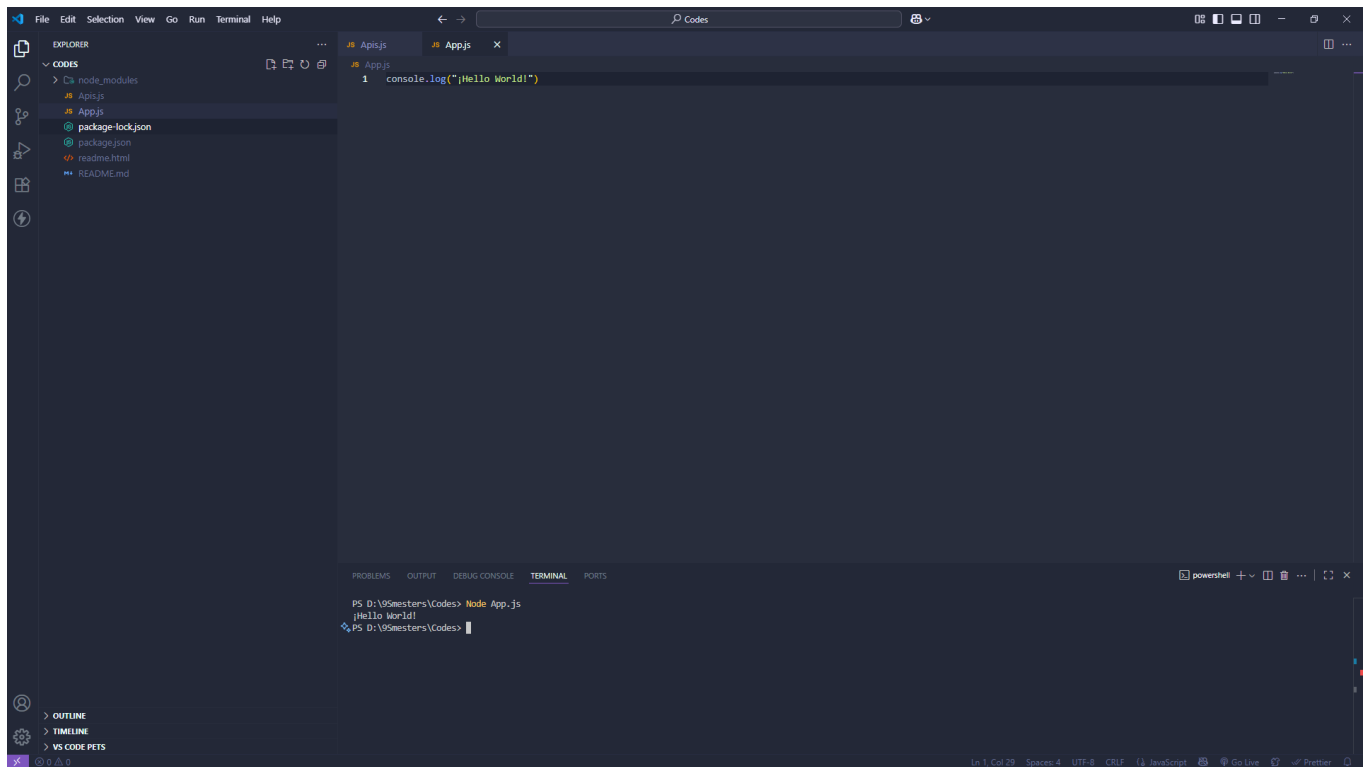
hay varias maneras de instalar express, la que yo usé es utilizando los codigos que te proporciona node.js (NPM)

```
npm install express
```

Ya con esto tendremos instalado Express, para verificar si tenemos express en la carpeta o el path utilizamos `npm ls express`



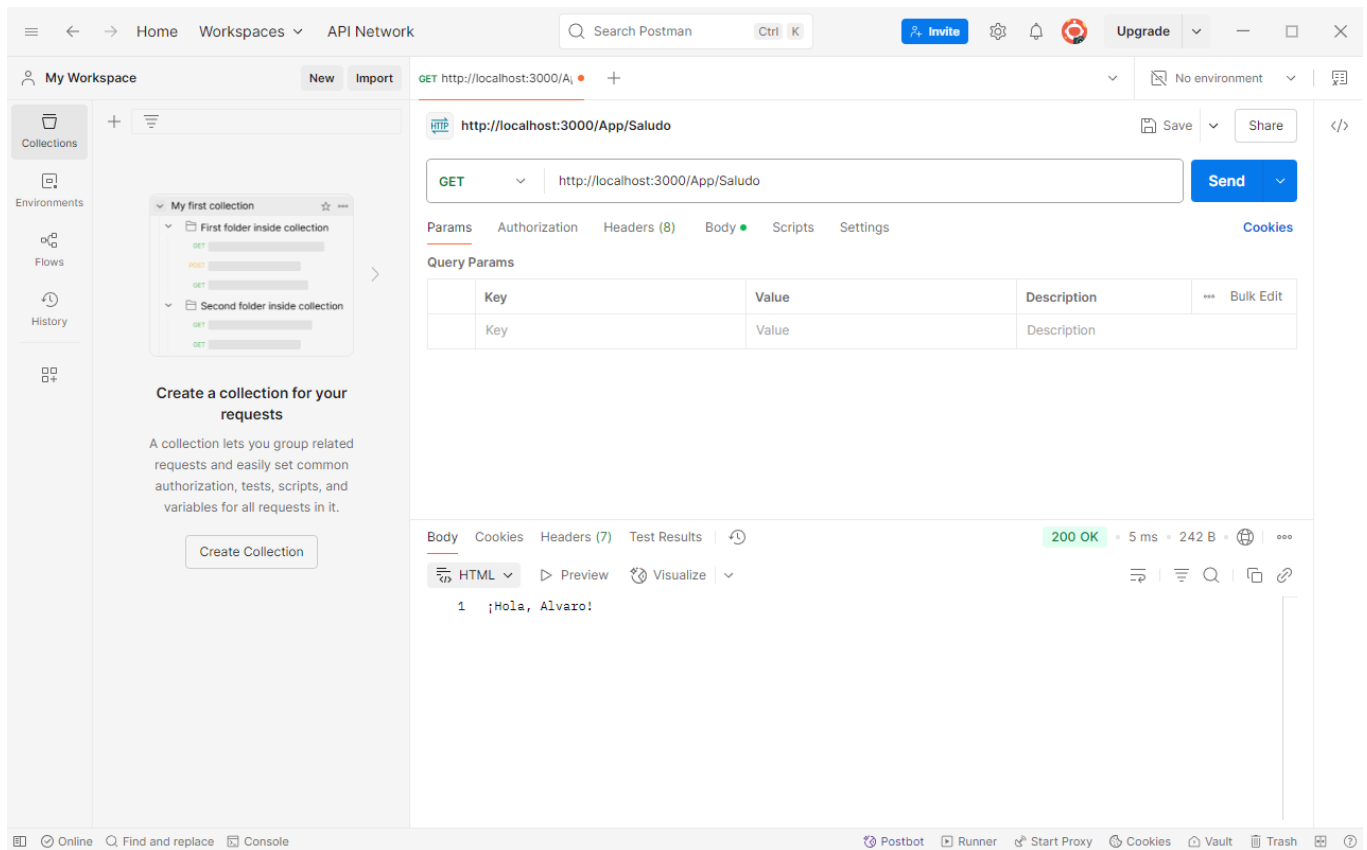
Hacemos nuestro primer hola mundo en node para verificar funcionamiento, utilizamos `Node %El nombre del archivo%` para correr el programa.



Ahora con express vamos a realizar una pequeña prueba para ver que este todo funcionando:

```
1 // Inicializamos express y colocamos por defecto el puerto 3000
2 const express = require("express");
3 const app = express();
4 const PORT = 3000;
5
6 app.get("/App/Saludo", (req, res) => {
7     res.send("¡Hola, Alvaro!")
8 })
9
10
11 //Arrancar servidor
12 app.listen(PORT, () => {
13     console.log(`Servidor escuchando en http://localhost:${PORT}`);
14 });
```

Con este pedazo de código podemos mandar un saludo y obtenerlo mediante POSTMAN (El cual es una plataforma que nos permite enviar requests mediante solicitudes HTTP, y sus respuesta.)



En este entorno podemos ver que realizamos la solicitud mediante la ruta /App/Saludo y recibimos la respuesta ¡Hola, Alvaro!

Ahora vamos a realizar un...

## Mini Proyecto.

Se nos pide realizar 3 endpoints:

- Metodo GET /tarefas -> Listar tareas
- Metodo POST /tarefas -> crear tareas
- Metodo PUT /tarefas/:id -> actualizar tareas (Lo manejé para que sea necesario con este metodo modificar tanto el string(Titulo) como el booleano(Completado).)  
Adicional a estos agregué algunos más para mejorar un poco el mini proyecto.
- Metodo Delete Delete/tarefas/:id
- Metodo Patch (Modificación parcial, nos permite poder mofidicar solo Completado o Titulo, tmb se pueden modificar ambos aqui, basicamente es una version mas completa)



```
1 // Inicializamos express y colocamos por defecto el puerto 3000
2 const express = require("express");
3 const app = express();
4 const PORT = 3000;
5
6 // Middleware para leer JSON en el body
7 app.use(express.json());
8
9 // "Base de datos" temporal en memoria
10 let tareas = [
11   { id: 1, titulo: "Estudiar Express", completada: false },
12   { id: 2, titulo: "Avanzar en UniSport", completada: true }
13 ];
```

En este pedazo de código inicializamos el servidor en el puerto 3000, y creamos una lista con dos tareas por defecto.



```
1  // GET → Listar todas las tareas
2  app.get("/api/tareas", (req, res) => {
3      res.json(tareas);
4  });
5
6  // POST → Crear una nueva tarea
7  app.post("/api/tareas", (req, res) => {
8      const { titulo } = req.body;
9      const nuevaTarea = {
10         id: tareas.length + 1,
11         titulo,
12         completada: false
13     };
14     tareas.push(nuevaTarea);
15     res.status(201).json(nuevaTarea);
16 });
17
18 // DELETE → Eliminar una tarea por id
19 app.delete("/api/tareas/:id", (req, res) => {
20     const id = parseInt(req.params.id);
21     tareas = tareas.filter(t => t.id !== id);
22     res.json({ mensaje: `Tarea ${id} eliminada` });
23 });
```

En este pedazo de código tenemos los tres métodos, GET (El cual nos muestra la lista), POST (El cual crea nuevas tareas que agregar a la lista), y DELETE (Borra tareas de la lista), además definimos el ID al momento de crear una nueva tarea con `length + 1`. (Esto hará que cada nueva tarea, tenga un número consecutivo.)

```

1 // PUT /api/tareas/:id -> reemplaza completamente (requiere 'titulo' y 'completada')
2 app.put("/api/tareas/:id", (req, res) => {
3   const id = Number(req.params.id);
4   if (!Number.isInteger(id)) {
5     return res.status(400).json({ error: "El parámetro :id debe ser un número entero" });
6   }
7
8   const { titulo, completada } = req.body || {};
9   if (typeof titulo !== "string" || !titulo.trim()) {
10    return res.status(400).json({ error: "El campo 'titulo' es requerido y no puede estar vacío" });
11  }
12  if (typeof completada !== "boolean") {
13    return res.status(400).json({ error: "El campo 'completada' es requerido y debe ser boolean" });
14  }
15
16  const idx = tareas.findIndex(t => Number(t.id) === id);
17  if (idx === -1) {
18    return res.status(404).json({ error: `No existe la tarea ${id}` });
19  }
20
21  //Evitar títulos duplicados (ignorando mayúsculas)
22  const dup = tareas.some(t => Number(t.id) !== id && t.titulo.toLowerCase() === titulo.trim().toLowerCase());
23  if (dup) {
24    return res.status(409).json({ error: "Ya existe una tarea con ese título" });
25  }
26
27  tareas[idx] = { id, titulo: titulo.trim(), completada };
28  return res.json(tareas[idx]);
29 });

```

este metodo utiliza PUT.

Con este pedazo de codigo lo que hacemos es reemplazar alguna tarea, con la condicion de que debemos reemplazar ambos parametros, (Tanto titulo como Completado), ademas de algunas verificaciones como por ejemplo:

1. No colocar titulos vacios ni el Booleano Completado vacio.
2. Si al momento de hacer el request no existe el ID saldrá el estado 404 (No existe la tarea)
3. Para titulos repetidos saldrá el estado 409.

Todas estas verificaciones se repiten para los siguientes metodos de modificación.

para los siguientes metodos usaremos PATCH.



```

1 // PATCH /api/tareas/:id -> actualización parcial (puedes enviar 'titulo' y/o 'completada')
2 app.patch("/api/tareas/:id", (req, res) => {
3   const id = Number(req.params.id);
4   if (!Number.isInteger(id)) {
5     return res.status(400).json({ error: "El parámetro :id debe ser un número entero" });
6   }
7
8   const idx = tareas.findIndex(t => Number(t.id) === id);
9   if (idx === -1) {
10    return res.status(404).json({ error: `No existe la tarea ${id}` });
11  }
12
13  let { titulo, completada } = req.body || {};
14  const tarea = { ...tareas[idx] };
15
16  if (titulo !== undefined) {
17    if (typeof titulo !== "string" || !titulo.trim()) {
18      return res.status(400).json({ error: "Si envías 'titulo', debe ser string no vacío" });
19    }
20    const dup = tareas.some(t => Number(t.id) !== id && t.titulo.toLowerCase() === titulo.trim().toLowerCase());
21    if (dup) {
22      return res.status(409).json({ error: "Ya existe una tarea con ese título" });
23    }
24    tarea.titulo = titulo.trim();
25  }
26
27  if (completada !== undefined) {
28    if (typeof completada !== "boolean") {
29      return res.status(400).json({ error: "Si envías 'completada', debe ser boolean" });
30    }
31    tarea.completada = completada;
32  }
33
34  tareas[idx] = tarea;
35  return res.json(tarea);
36 });

```

En esta parte del código hacemos las verificaciones y además permitimos modificar alguna parte del cuerpo, (Puede ser solamente el título o Completado)

```

1 //PATCH rápido para alternar completada
2 app.patch("/api/tareas/:id/toggle", (req, res) => {
3   const id = Number(req.params.id);
4   if (!Number.isInteger(id)) {
5     return res.status(400).json({ error: "El parámetro :id debe ser un número entero" });
6   }
7   const idx = tareas.findIndex(t => Number(t.id) === id);
8   if (idx === -1) {
9     return res.status(404).json({ error: `No existe la tarea ${id}` });
10  }
11  tareas[idx].completada = !tareas[idx].completada;
12  res.json(tareas[idx]);
13 });

```

Con este pedazo de código al colocar la ruta `/api/tareas/:id/toggle` podemos atornar rápidamente el booleano, si se encuentra en `false` pasara a `true`, y viceversa.

Vamos con las peticiones.

Podemos ver nuestra lista inicial, la pedimos usando el api GET.

The screenshot shows the Postman interface with a GET request to `http://localhost:3000/api/tareas` executed successfully. The response is a JSON array of two task objects.

**Request Details:**

- Method: GET
- URL: `http://localhost:3000/api/tareas`
- Query Params: (Empty table)

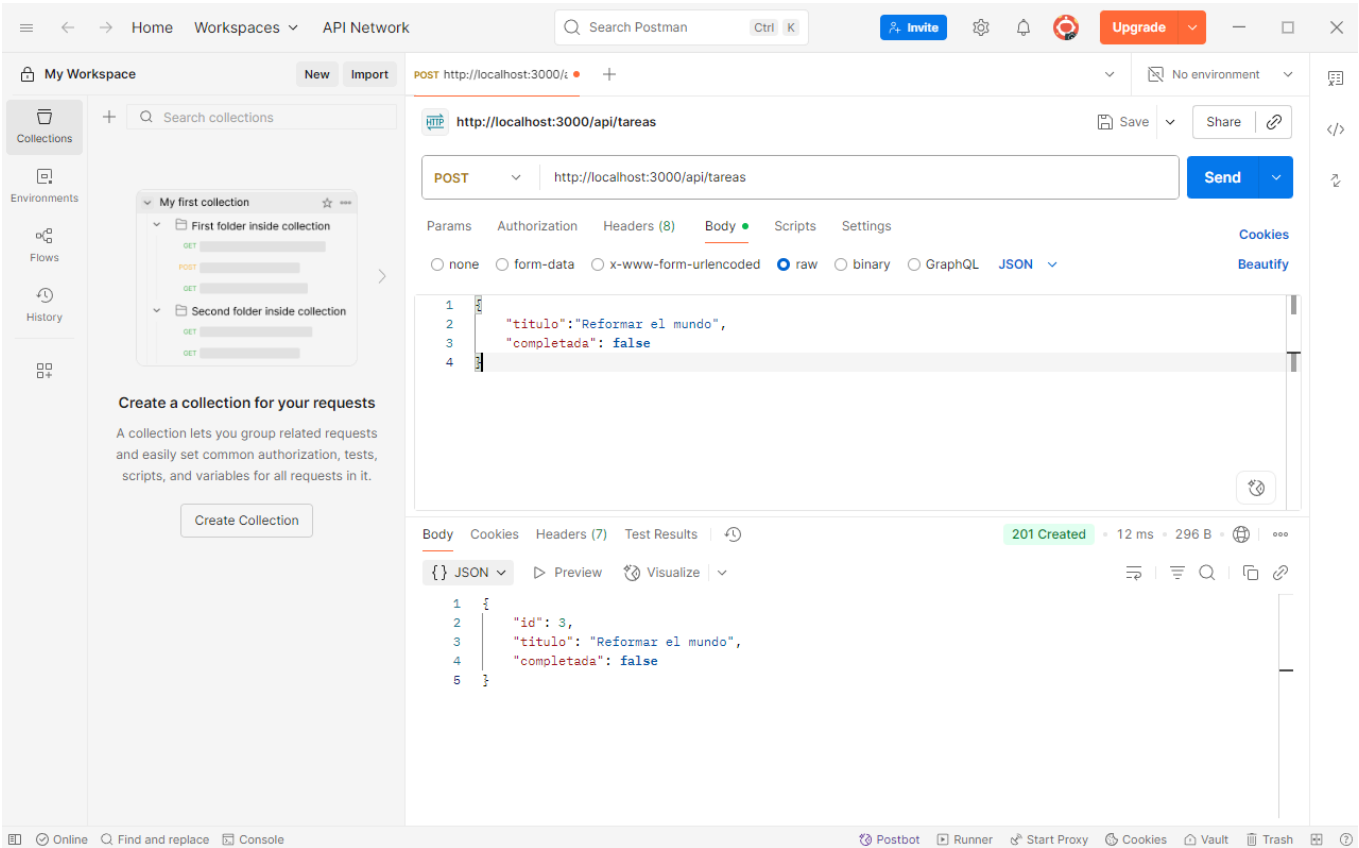
**Response Details:**

- Status: 200 OK
- Time: 172 ms
- Size: 351 B

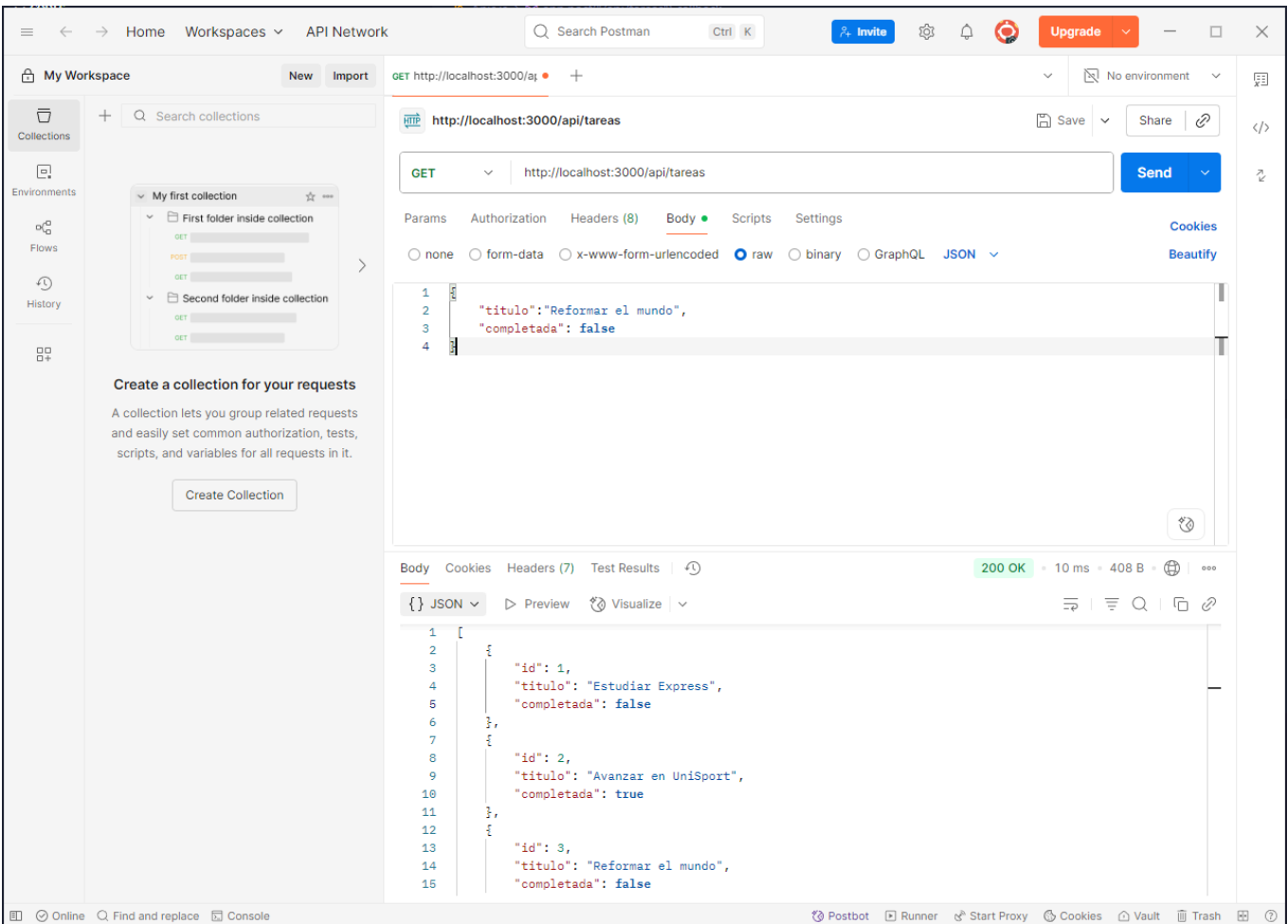
**JSON Response:**

```
1 {
2   {
3     "id": 1,
4     "titulo": "Estudiar Express",
5     "completada": false
6   },
7   {
8     "id": 2,
9     "titulo": "Avanzar en UniSport",
10    "completada": true
11  }
12 }
```

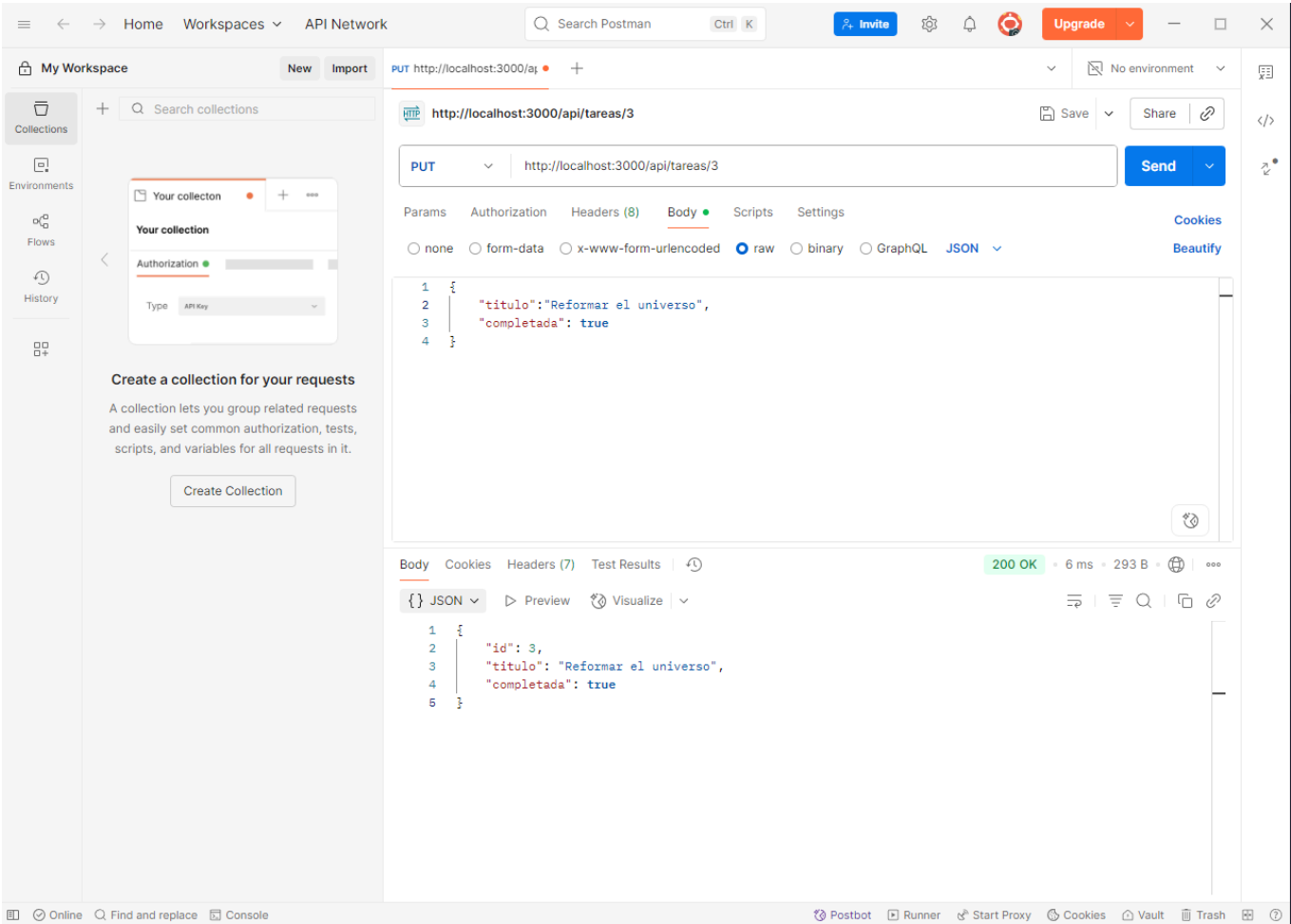
Aqui agregamos una nueva tarea a la lista con la request POST.



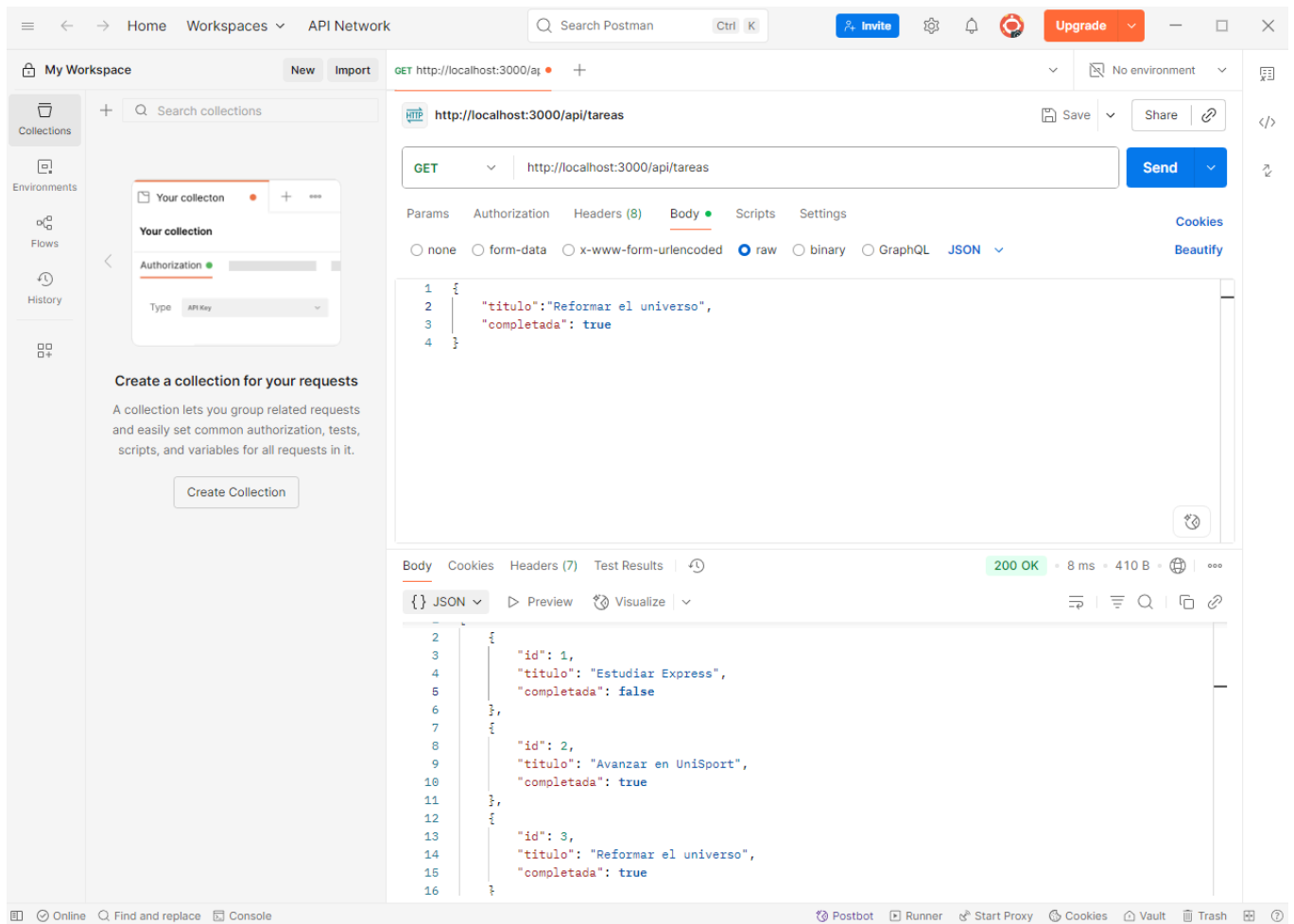
Lista actualizada.



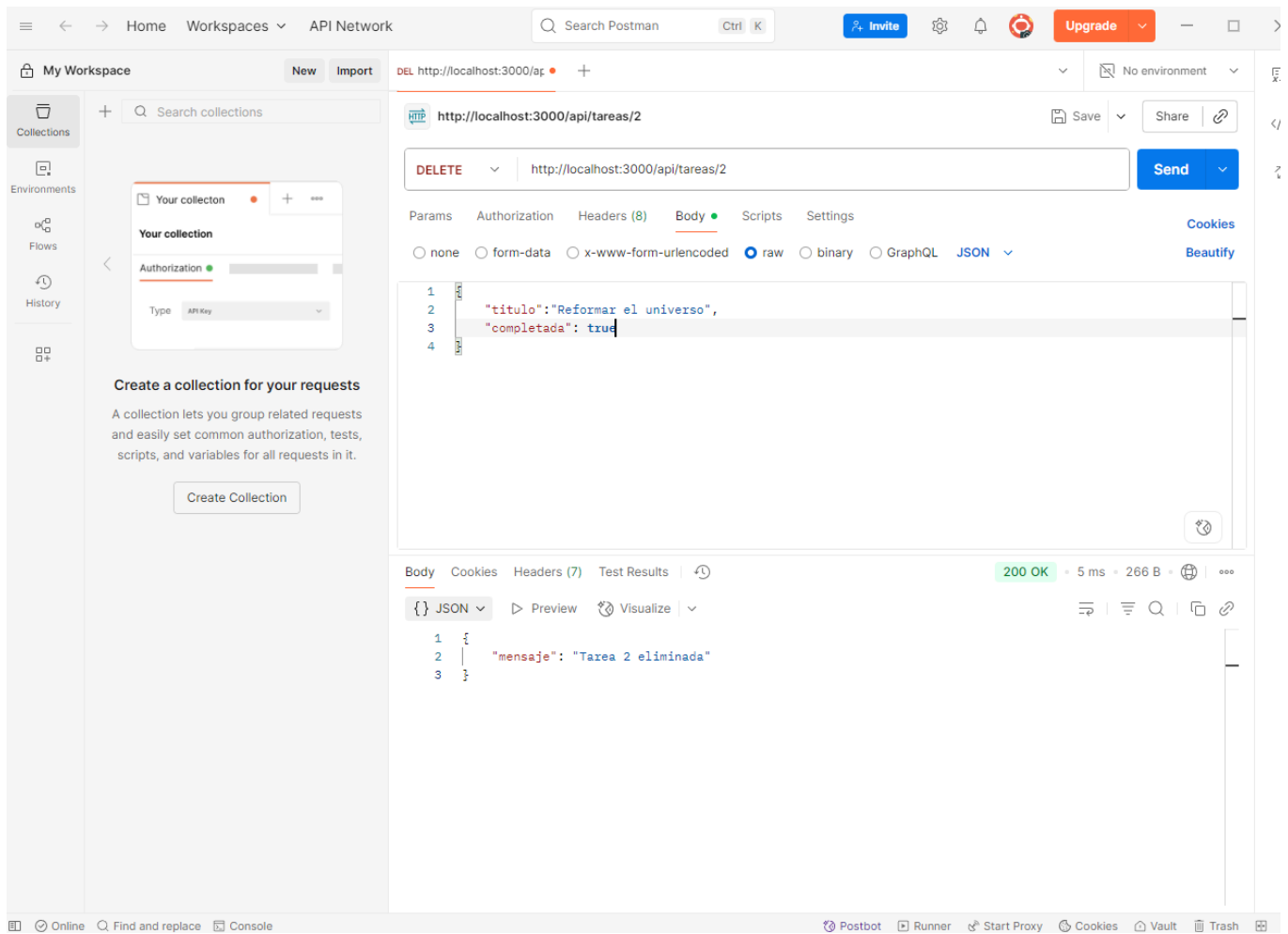
Ahora vamos a modifiamos utilizando esta vez el PUT.



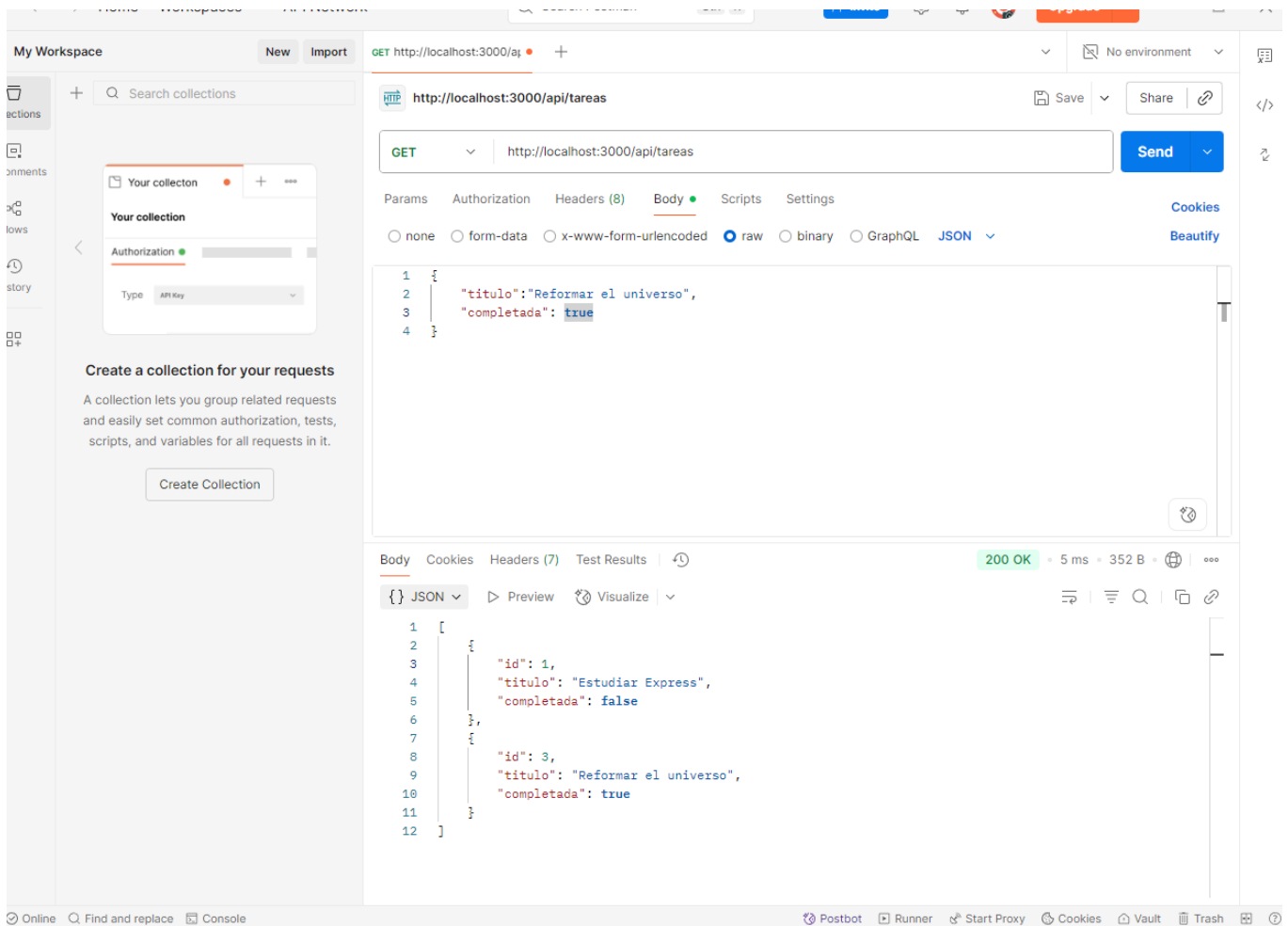
Lista actualizada



Eliminamos la segunda tarea.



Esta seria la lista actualizada



Link del repositorio: [https://github.com/Alvxro12/Apis-Desarrollo\\_web\\_II-MiniProyecto](https://github.com/Alvxro12/Apis-Desarrollo_web_II-MiniProyecto)