

Queremos desarrollar un backend para un ayuntamiento que cuente con un servicio de alquiler de bicicletas. Vamos a desarrollar la API de un proyecto siguiendo la estructura que express-generator nos proporcionó en la tarea anterior.

URL DE RENDER: <https://bicicletas.onrender.com/api/bicicletas>

Para ello vamos a crear un nuevo proyecto con express generator como en la practica anterior.

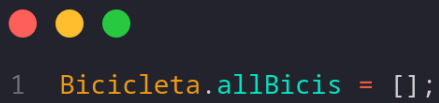
### Ejercicio 1. Modelo Bicicleta

En la estructura de proyecto generada en la práctica anterior, en la carpeta raíz crearemos una carpeta llamada `models` y, dentro de ella, un archivo `Bicicleta.js` que contendrá toda la lógica del modelo Bicicleta. Es una buena práctica escribir la primera letra de los modelos en mayúscula y nombrarlos en singular.

A continuación, crea un constructor para el modelo con sus atributos:

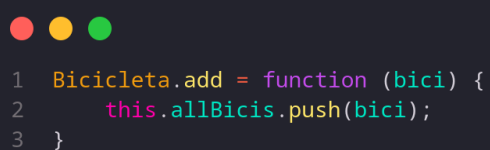
```
1  let Bicicleta = function (id, color, modelo, ubicacion) {  
2    this.id = id;  
3    this.color = color;  
4    this.modelo = modelo;  
5    this.ubicacion = ubicacion;  
6  }
```

Crea un array llamado `allBicis` que contendrá objetos de tipo Bicicleta. Cuando utilicemos MongoDB para almacenar los modelos, no necesitaremos este arreglo. Por ahora, lo usaremos para simplificar:



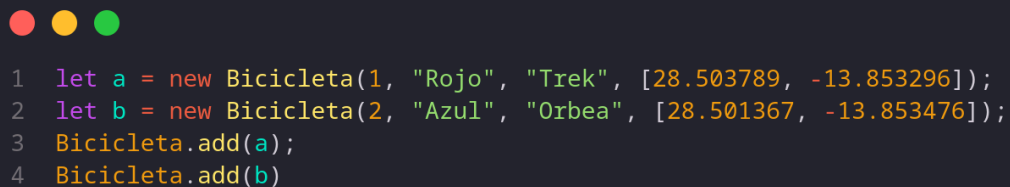
```
1  Bicicleta.allBicis = [];
```

**Crea un método para añadir una bicicleta al final del array:**



```
1  Bicicleta.add = function (bici) {  
2    this.allBicis.push(bici);  
3  }
```


**Agreguemos un par de bicicletas al array de bicicletas en memoria. Estos datos están “hardcodeados”, lo cual no es la mejor opción, pero lo hacemos así para simplificar.**



```
1  let a = new Bicicleta(1, "Rojo", "Trek", [28.503789, -13.853296]);  
2  let b = new Bicicleta(2, "Azul", "Orbea", [28.501367, -13.853476]);  
3  Bicicleta.add(a);  
4  Bicicleta.add(b)
```

**Exporta el módulo para que sea accesible en toda tu app.**

Primero creamos una carpeta controllers en el raíz del proyecto y dentro una carpeta llamada api, dentro de ella, un archivo BicicletaControllerAPI.js:



```

1 let Bicicleta = require("../models/Bicicleta");
2
3 exports.bicicleta_list = function(req, res) {
4     res.status(200).json ({
5         bicicletas: Bicicleta.allBicis
6     });
7 };

```

Ahora tenemos que crear la ruta para ese controlador. Creamos una carpeta api dentro de routes y, dentro de ella, un archivo bicicletas.js:




```

1 let express = require('express');
2 let router = express.Router();
3 let BicicletaControllerAPI = require("../controllers/api/BicicletaControllerAPI");
4
5 router.get("/", BicicletaControllerAPI.bicicleta_list);
6
7 module.exports = router;

```

Tenemos que modificar app.js para agregar el router que se encargará de la API (coloca el código en las líneas correspondientes):

Esto con el resto de variables



```

1 var bicicletasAPIRouter = require('./routes/api/bicicletas');

```

y esto :

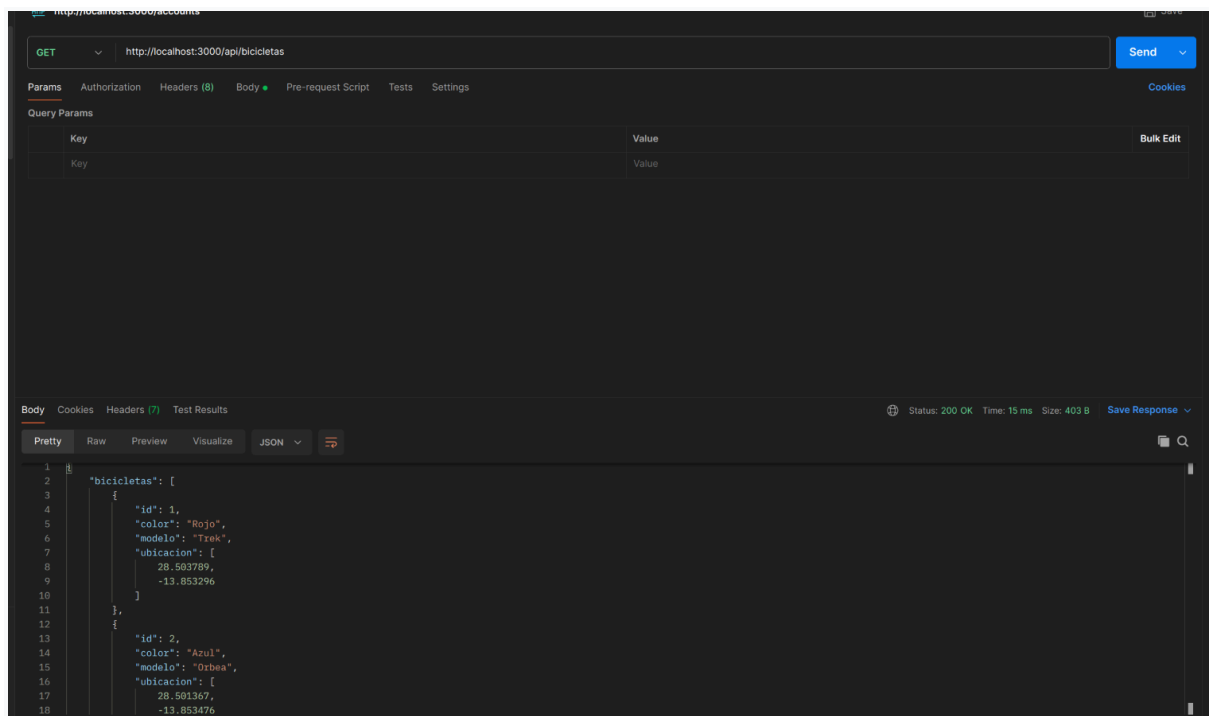


```
1 app.use('/api/bicicletas', bicicletasAPIRouter);
```

## como ruta

Vamos a hacer las pruebas con postman, primero levantamos el servidor con el comando : ***npm start***

Y probamos con el postman :



como vemos nos devuelve las bicicletas

## Ejercicio 3. POST

Ahora implementemos el método POST. Vamos al controlador de la API y exportamos el método `bicicleta_create`:

```

1  exports.bicicleta_create = function(req,res){
2    let bici = new Bicicleta(req.body.id, req.body.color, req.body.modelo);
3    bici.ubicacion = [req.body.latitud, req.body.longitud];
4
5    Bicicleta.add(bici);
6
7    res.status(201).json({
8      bicicleta: bici
9    })
10 }

```

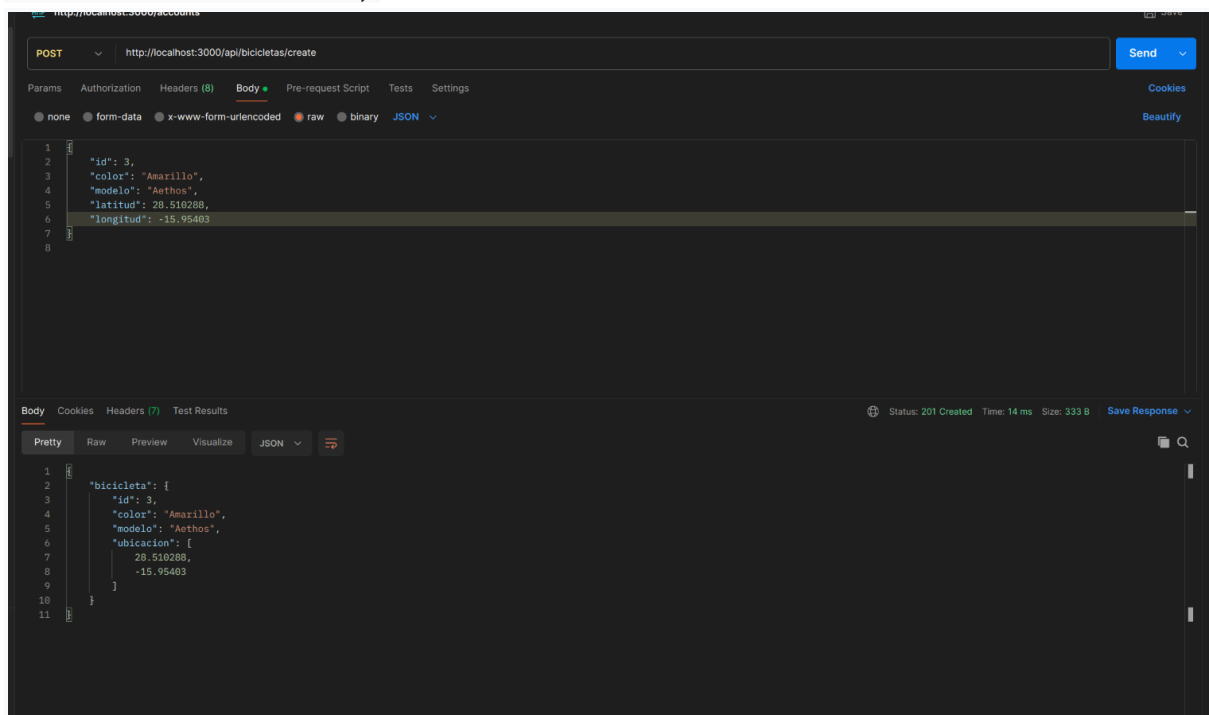
Ahora tenemos que añadir la ruta en routes api:

```

1  router.get("/", BicicletaControllerAPI.bicicleta_list);
2  router.post("/create", BicicletaControllerAPI.bicicleta_create);

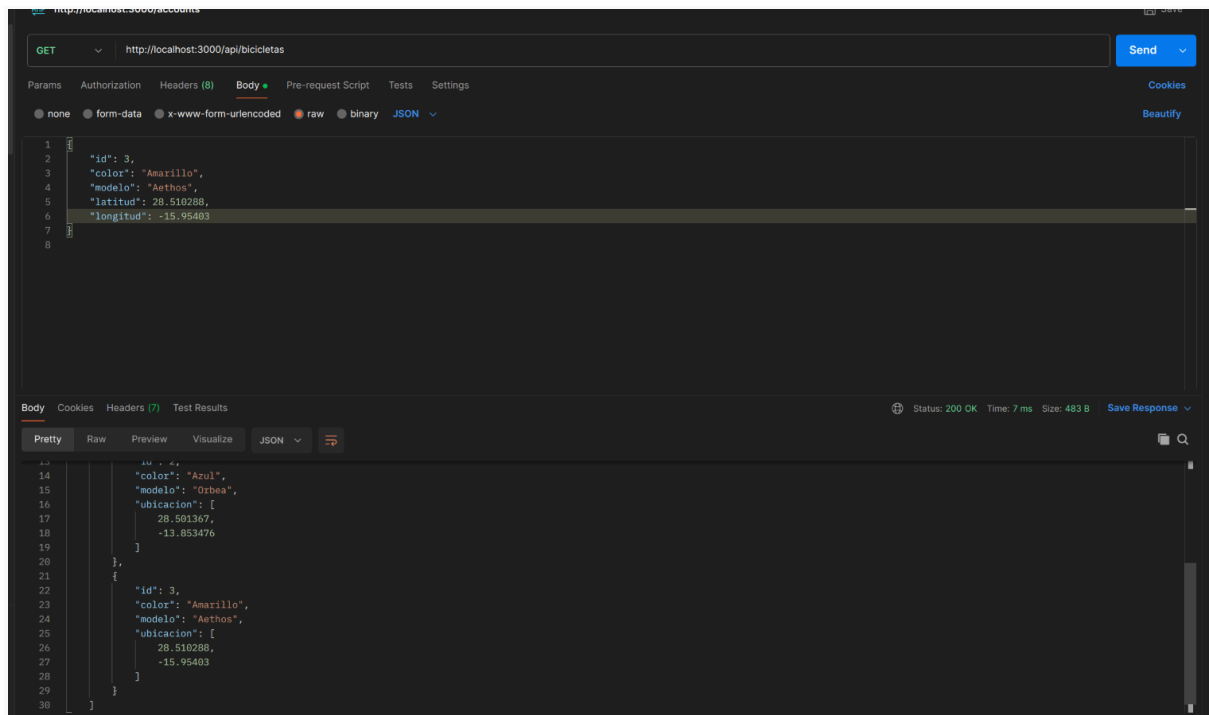
```

Y en el postman probamos el método create (hay que acordarse de reiniciar el servidor cuando se hacen cambios).



como vemos se crea perfectamente

Ahora podemos probar el metodo get a ver si se ve con el resto de bicicletas:



Si, sale con el resto de bicicletas.

## Ejercicio 4. DELETE

Tendrás que implementar el método `removeById` en el controlador Bicicleta:



Y en el modelo estableceremos la función tal cual.

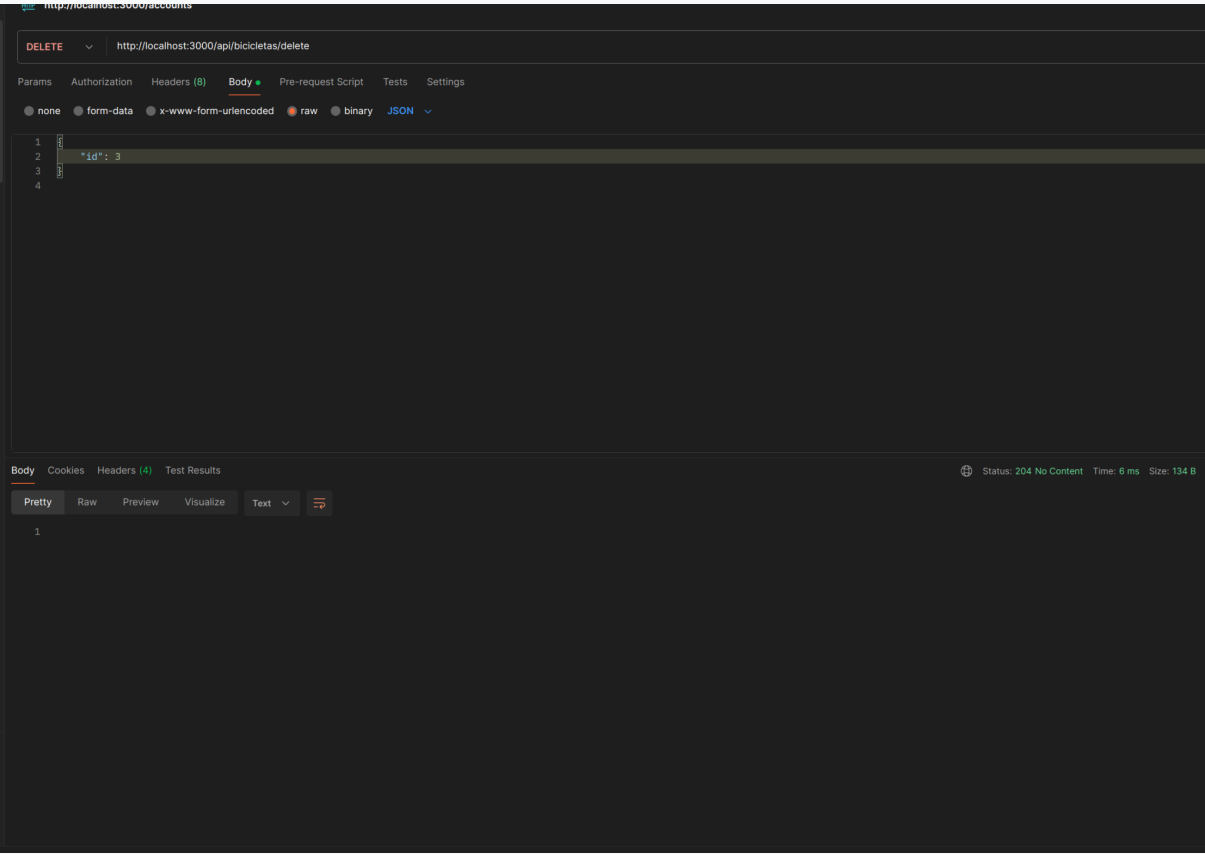
.

```
1 //Eliminar Bicicleta
2 Bicicleta.removeById = function (idbicicleta){
3     const bicicletaelegida = this.allBicis.findIndex(bici => bici.id === idbicicleta);
4
5     if(bicicletaelegida !== -1){
6         this.allBicis.splice(bicicletaelegida,1);
7         console.log(`La bicicleta con ID ${idbicicleta} eliminada exitosamente`);
8     }
9     else{
10         console.log(`La bicicleta con ID ${idbicicleta} no existe`);
11     }
12 }
```

La ruta será la siguiente:

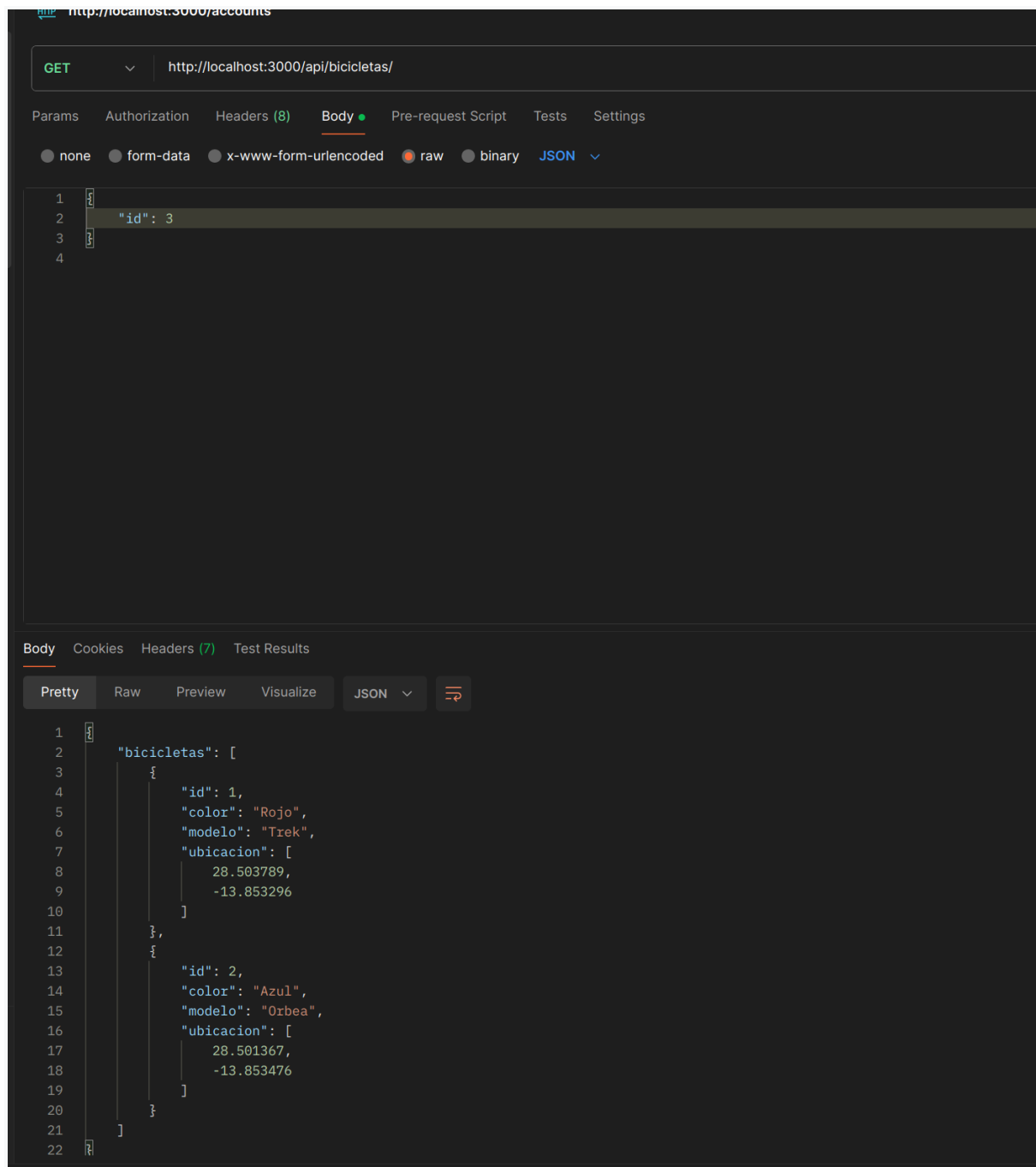
```
1 router.delete("/delete", BicicletaControllerAPI.bicicleta_delete);
```

Ahora podemos probar con Postman si funciona:



Como vemos se ha borrado la bicicleta:





## Ejercicio 5. PUT (update)

Ahora lo que vamos a hacer es una función que modifique el objeto de la bicicleta entero:

Primero hacemos la función en el modelo :

```

1  Bicicleta.update = function (id, color, modelo, latitud , longitud) {
2      const bicicletaelegida = this.allBicis.findIndex(bici => bici.id === id);
3
4      if(bicicletaelegida !== -1){
5          this.allBicis[bicicletaelegida].color = color;
6          this.allBicis[bicicletaelegida].modelo = modelo;
7          this.allBicis[bicicletaelegida].ubicacion = [latitud , longitud];
8
9          console.log(`La bicicleta con el ${id} actualizada correctamente`);
10     }
11     else{
12         console.log(`La bicicleta con el ${id} no existe`);
13     }
14 }

```

Ahora tenemos que llamar a la función en el controlador, como con las otras funciones :

```

1  exports.bicicleta_update = function(req,res){
2      Bicicleta.update(req.body.id,req.body.color,req.body.modelo,req.body.latitud,req.body.longitud);
3      res.status(200).send();
4  }

```

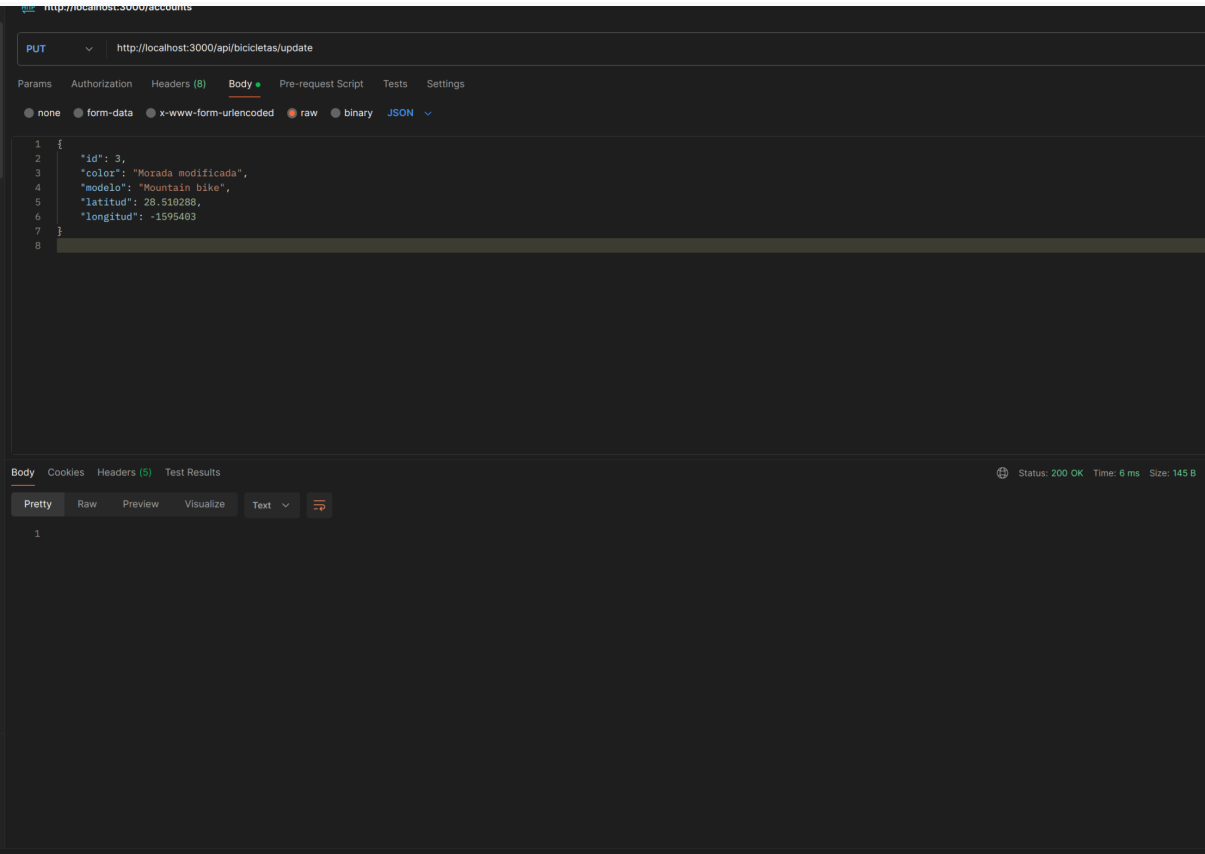
Por último dejar la ruta definida :

```

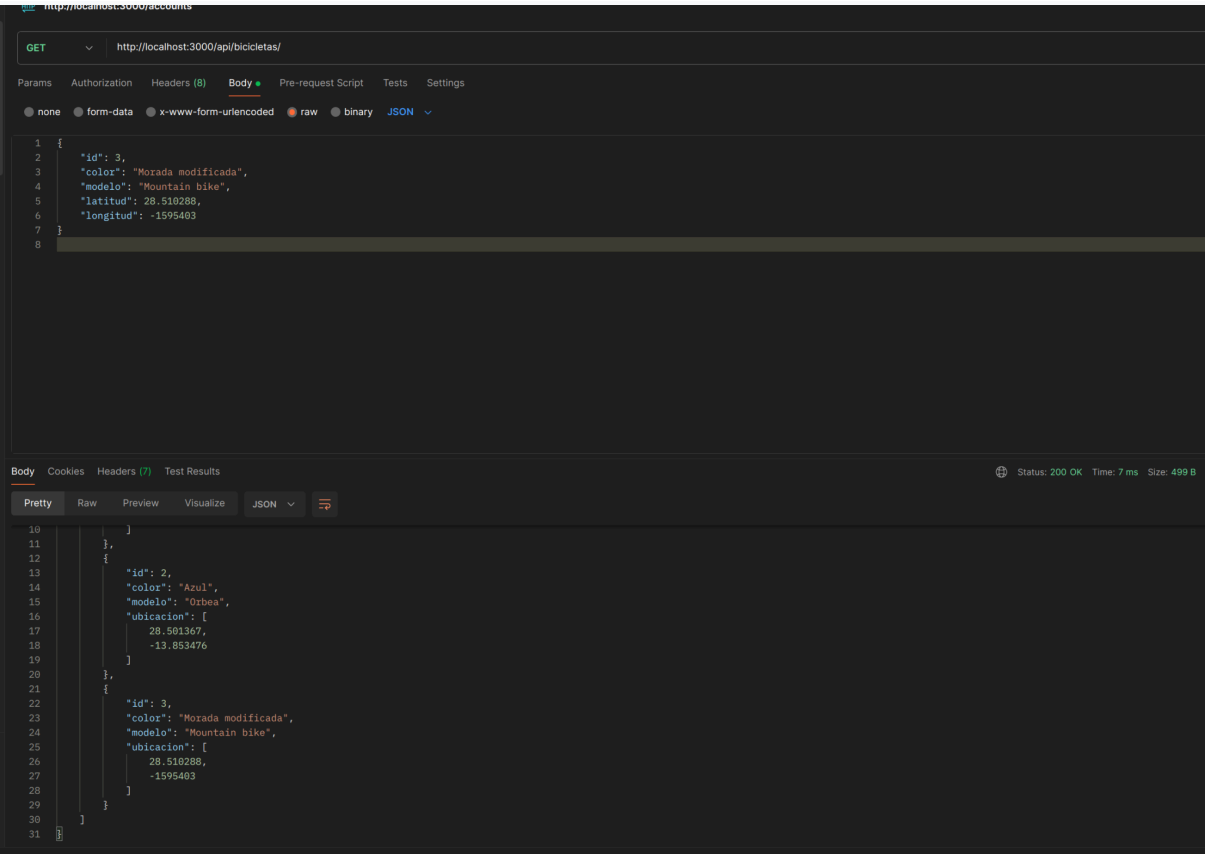
1  router.get("/", BicicletaControllerAPI.bicicleta_list);
2  router.post("/create", BicicletaControllerAPI.bicicleta_create);
3  router.delete("/delete", BicicletaControllerAPI.bicicleta_delete);
4  router.put("/update", BicicletaControllerAPI.bicicleta_update);

```

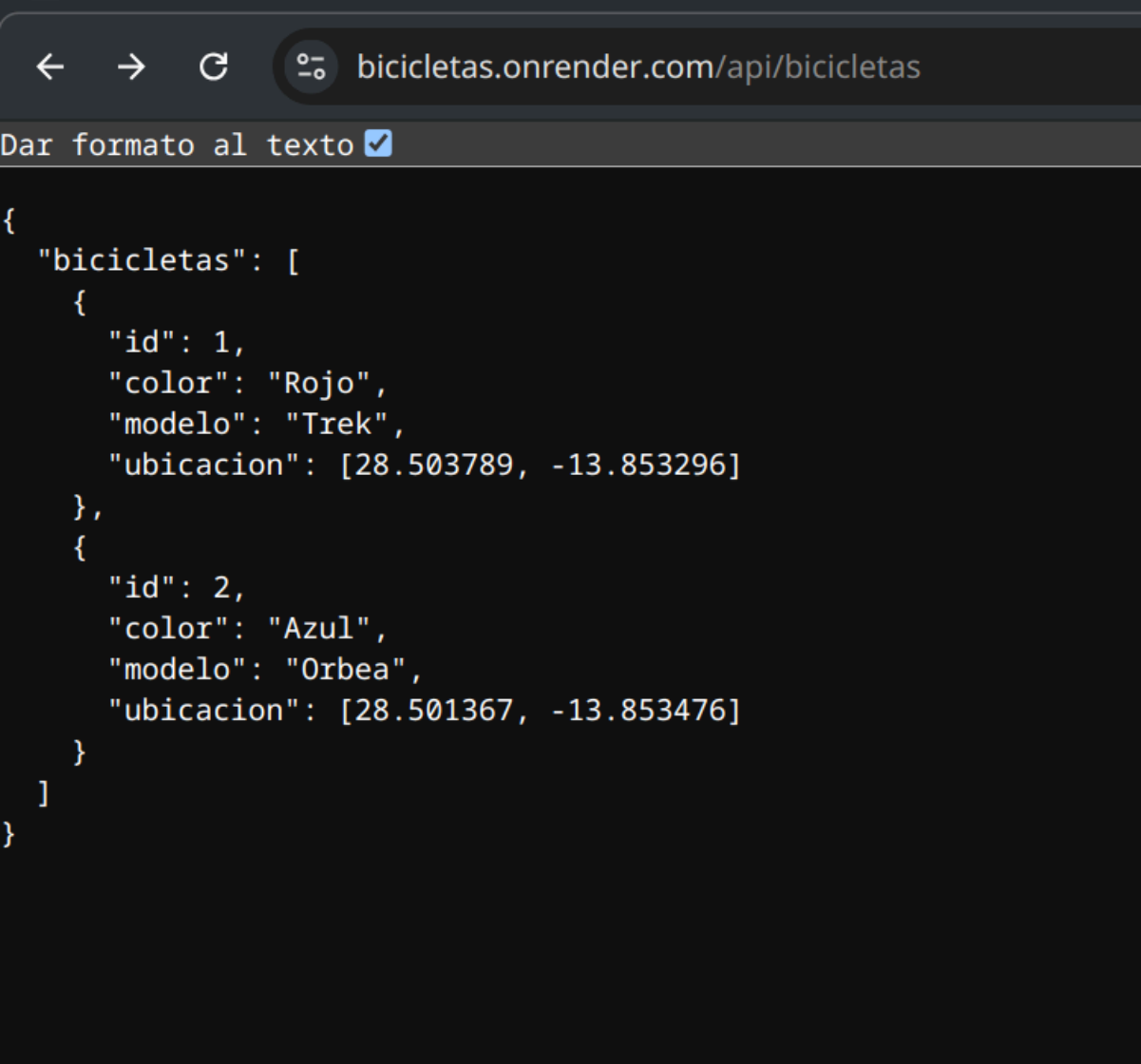
Ahora podemos en el postman:



Como vemos ha sido modificada con éxito :

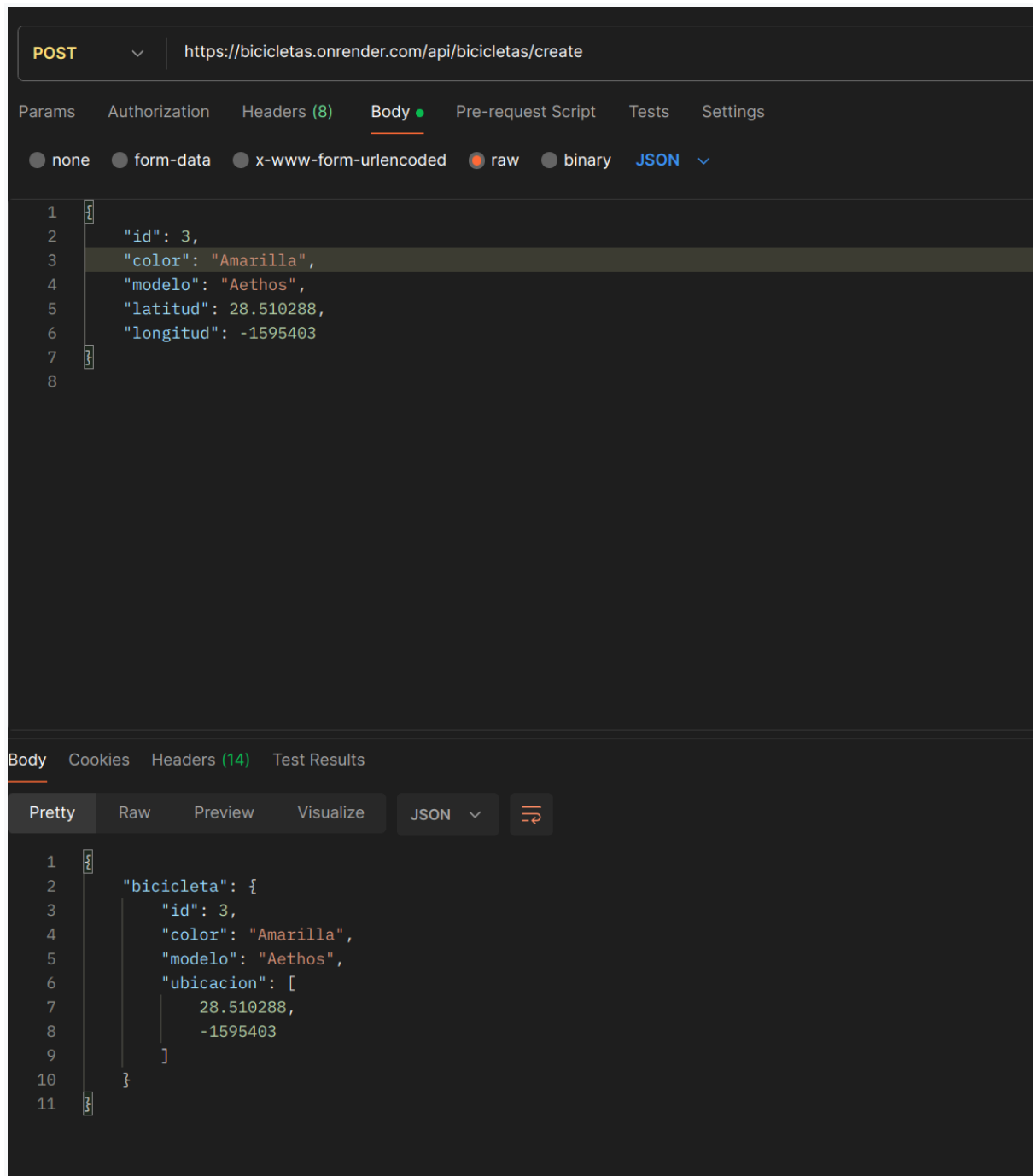


Vamos a subirlo a Render y probarlo todo, para hacerlo tenemos que subir el repositorio a github (no olvidar el gitignore).

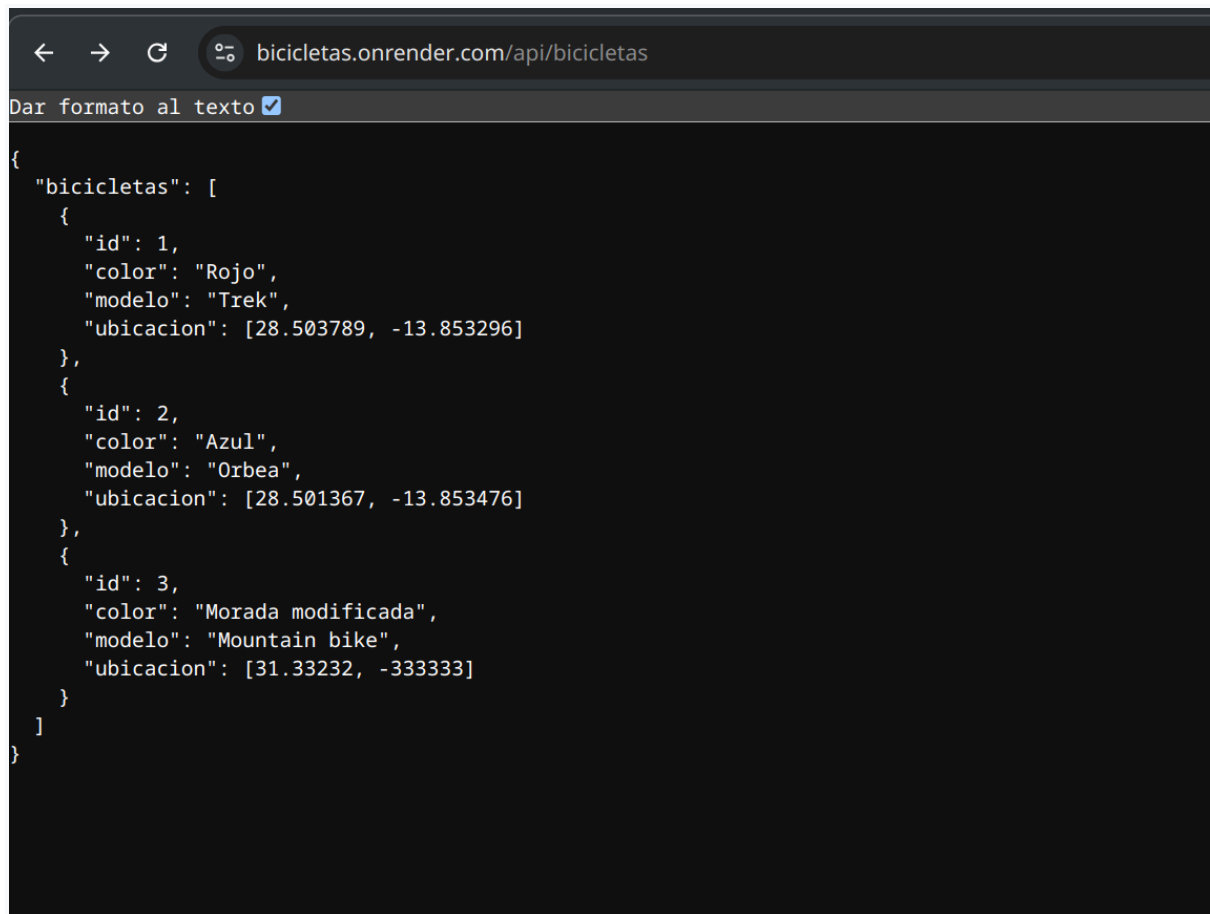


```
{
  "bicicletas": [
    {
      "id": 1,
      "color": "Rojo",
      "modelo": "Trek",
      "ubicacion": [28.503789, -13.853296]
    },
    {
      "id": 2,
      "color": "Azul",
      "modelo": "Orbea",
      "ubicacion": [28.501367, -13.853476]
    }
  ]
}
```

Como vemos está funcionando correctamente, vamos a ir probando los métodos por orden, primero creare una bicicleta, después la modificare y por último la borraré.



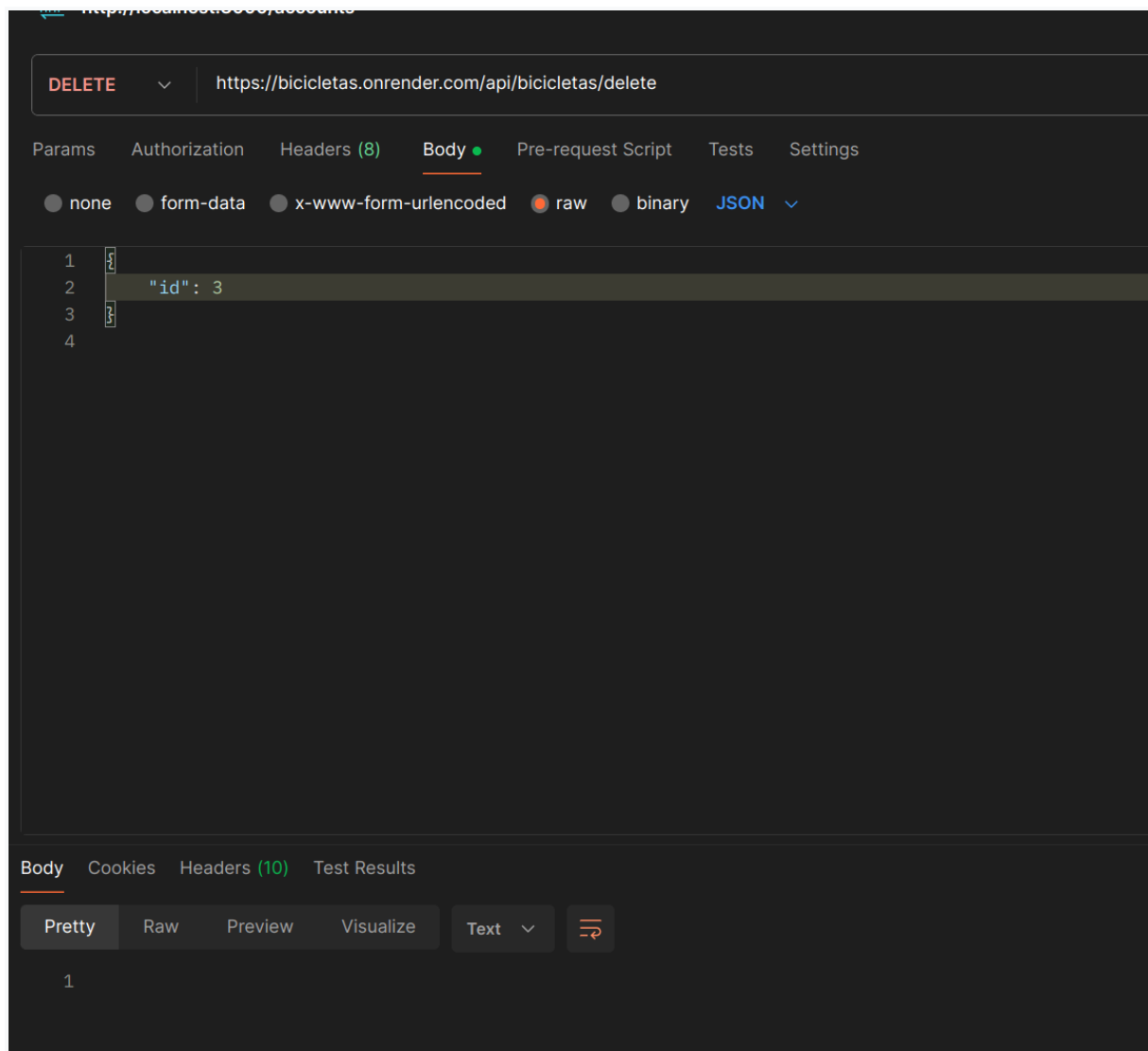
Como vemos funciona al crear una Bicicleta



A screenshot of a web browser window. The address bar shows the URL `bicicletas.onrender.com/api/bicicletas`. Below the address bar, there is a checkbox labeled "Dar formato al texto" which is checked. The main content area of the browser displays a JSON array of three bicycle objects. The JSON is formatted with syntax highlighting: strings are in quotes, numbers are in plain text, and arrays are in brackets. The objects contain fields for id, color, modelo, and ubicacion.

```
{
  "bicicletas": [
    {
      "id": 1,
      "color": "Rojo",
      "modelo": "Trek",
      "ubicacion": [28.503789, -13.853296]
    },
    {
      "id": 2,
      "color": "Azul",
      "modelo": "Orbea",
      "ubicacion": [28.501367, -13.853476]
    },
    {
      "id": 3,
      "color": "Morada modificada",
      "modelo": "Mountain bike",
      "ubicacion": [31.33232, -333333]
    }
  ]
}
```

Modificarlas igual



Y borrarla igual

```
← → ↻ 🌐 bicicletas.onrender.com/api/bicicletas
Dar formato al texto ☒
{
  "bicicletas": [
    {
      "id": 1,
      "color": "Rojo",
      "modelo": "Trek",
      "ubicacion": [28.503789, -13.853296]
    },
    {
      "id": 2,
      "color": "Azul",
      "modelo": "Orbea",
      "ubicacion": [28.501367, -13.853476]
    }
  ]
}
```