

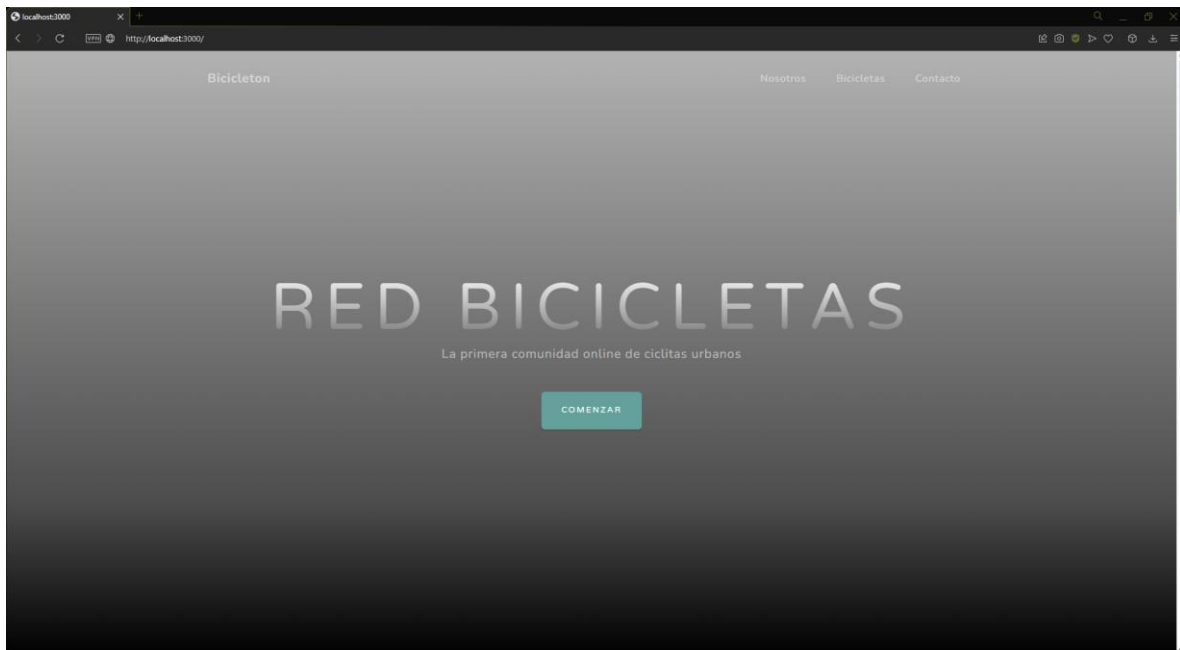
Documentación API

RUN API

Ponemos a correr la API desde la terminal con el comando `npm run devstart`

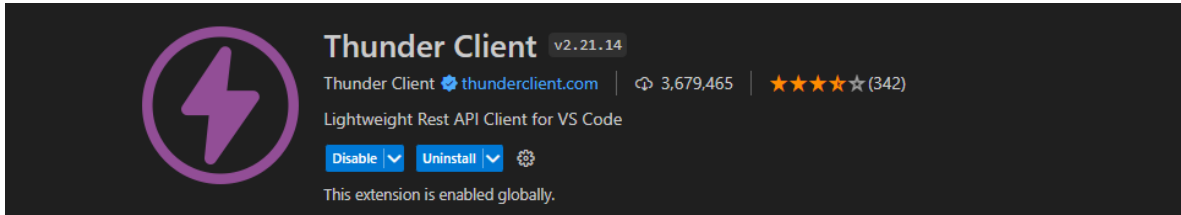
```
PS D:\C11D Gray Col\Documents\JavaScript\red_bicicletas> npm run devstart  
  
> red-bicicletas@0.0.1 devstart  
> nodemon ./bin/www  
  
[nodemon] 3.1.0  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node ./bin/www`
```

Verificamos desde el navegador con la ruta <http://localhost:3000/>

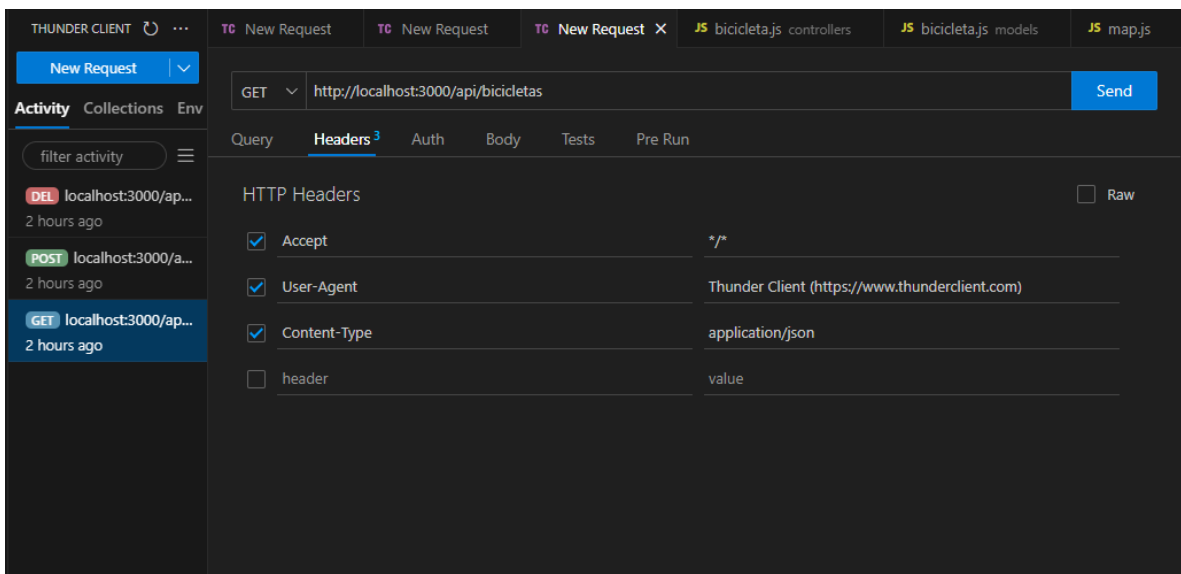


METODOS

Utilizando el plugin de Visual Studio Code llamado "Thunder Client" haremos las pruebas correspondientes para corroborar estos métodos



Iniciamos una nueva solicitud (new request) y seleccionamos el método GET y escribimos el url de donde queremos mostrar los datos, en este caso <http://localhost:3000/api/bicicletas>



Ya con esto enviamos la solicitud, en la parte derecha en response nos dará los datos que tiene en formato .json en la pestaña Response

```
Status: 200 OK Size: 204 Bytes Time: 4 ms

Response Headers Cookies Results Docs {} ≡

1 {
2   "bicicletas": [
3     {
4       "id": 1,
5       "color": "rojo",
6       "modelo": "urbana",
7       "ubicacion": [
8         4.587067826932398,
9         -74.16080334293196
10      ]
11    },
12    {
13      "id": 2,
14      "color": "blanca",
15      "modelo": "urbana",
16      "ubicacion": [
17        4.580993341635696,
18        -74.15413000710157
19      ]
20    }
21  ]
22 }
```

POST

Iniciamos una nueva solicitud (new request) y seleccionamos el método POST y escribimos el url de donde queremos insertar el dato, en este caso <http://localhost:3000/api/bicicletas/create>

Nos dirigimos a la pestaña de Body e ingresamos el código en .json del dato que queremos ingresar, en este caso ingresaremos este dato

```
{
  "id": 5,
  "color": "rojo",
  "modelo": "urbana",
  "lat": 4.5804644161547,
  "lng": -74.1574237570102
}
```

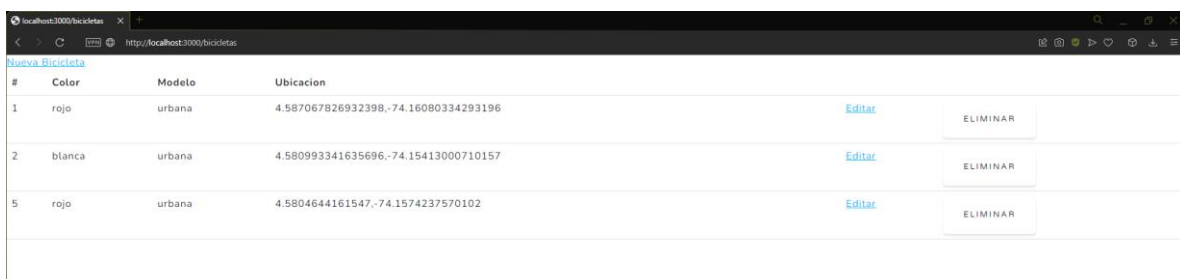
Ya teniendo esto enviamos la solicitud, en la parte derecha en la pestaña Response nos mostrara el nuevo dato que es ingresado

```
Status: 200 OK Size: 103 Bytes Time: 9 ms

Response Headers 6 Cookies Results Docs {} ≡

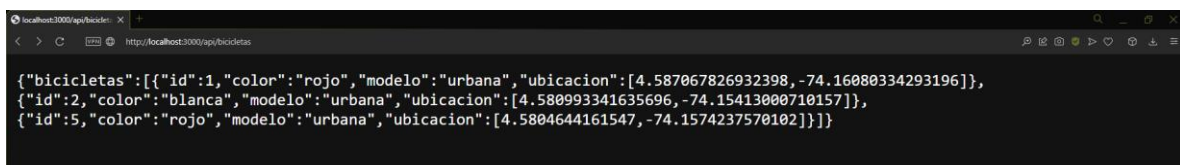
1 {
2   "bicicleta": {
3     "id": 5,
4     "color": "rojo",
5     "modelo": "urbana",
6     "ubicacion": [
7       4.5804644161547,
8       -74.1574237570102
9     ]
10  }
11 }
```

Nos vamos al link <http://localhost:3000/bicicletas> donde se muestran los datos y vemos la nueva bicicleta agregada



#	Color	Modelo	Ubicación	
1	rojo	urbana	4.587067826932398,-74.16080334293196	Editar <button>ELIMINAR</button>
2	blanca	urbana	4.580993341635696,-74.15413000710157	Editar <button>ELIMINAR</button>
5	rojo	urbana	4.5804644161547,-74.1574237570102	Editar <button>ELIMINAR</button>

Igualmente, en el link <http://localhost:3000/api/bicicletas> donde se ven los datos en texto plano se puede apreciar también la nueva bicicleta agregada



```
{
  "bicicletas": [
    {
      "id": 1,
      "color": "rojo",
      "modelo": "urbana",
      "ubicacion": [
        4.587067826932398,
        -74.16080334293196
      ]
    },
    {
      "id": 2,
      "color": "blanca",
      "modelo": "urbana",
      "ubicacion": [
        4.580993341635696,
        -74.15413000710157
      ]
    },
    {
      "id": 5,
      "color": "rojo",
      "modelo": "urbana",
      "ubicacion": [
        4.5804644161547,
        -74.1574237570102
      ]
    }
  ]
}
```

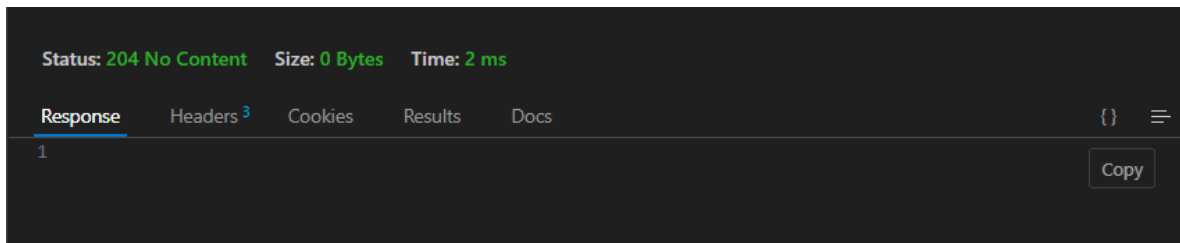
DELETE

Iniciamos una nueva solicitud (new request) y seleccionamos el método DELETE y escribimos el url de donde queremos eliminar el dato, en este caso <http://localhost:3000/api/bicicletas/delete>

Nos dirigimos a la pestaña de Body e ingresamos el código en .json del dato que queremos eliminar, en este caso ingresaremos este dato

```
{
  "id": 1
}
```

Ya con esto enviamos la solicitud, en la parte derecha en response no nos mostrara nada, pero la acción ya se encuentra hecha (el Status nos muestra 204 el cual fue el que se indicó en el código)

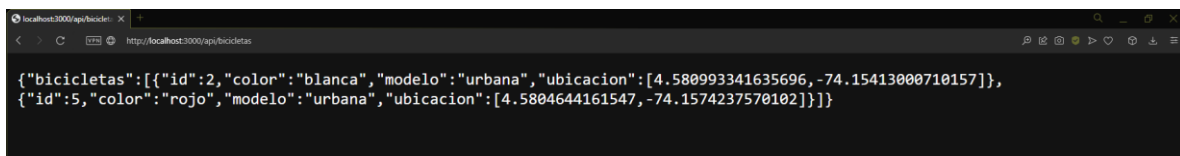


Nos vamos al link <http://localhost:3000/bicicletas> donde se muestran los datos y vemos la bicicleta eliminada

The screenshot shows a web browser displaying a list of bicycles. The table has columns for '#', 'Color', 'Modelo', and 'Ubicación'. There are two rows of data, each with an 'Editar' link and an 'ELIMINAR' button.

#	Color	Modelo	Ubicación		
2	blanca	urbana	4.580993341635696,-74.15413000710157	Editar	ELIMINAR
5	rojo	urbana	4.5804644161547,-74.1574237570102	Editar	ELIMINAR

Igualmente, en el link <http://localhost:3000/api/bicicletas> donde se ven los datos en texto plano se puede apreciar también la bicicleta eliminada



UPDATE

La actualización de datos la hacemos directamente desde la pagina en el link <http://localhost:3000/api/bicicletas> donde se ven las bicicletas y dan la opción de modificar como se muestra en la imagen

The screenshot shows a web browser displaying a list of bicycles. The table has columns for '#', 'Color', 'Modelo', and 'Ubicación'. There are two rows of data, each with an 'Editar' link and an 'ELIMINAR' button. The 'Editar' links are circled in red.

#	Color	Modelo	Ubicación		
2	blanca	urbana	4.580993341635696,-74.15413000710157	Editar	ELIMINAR
5	rojo	urbana	4.5804644161547,-74.1574237570102	Editar	ELIMINAR

Al darle a esa opción nos da una ventana con un formulario para poder actualizarla

[volver](#)

Actualizando Bicicleta

ID:

Color:

Modelo:

Latitud:

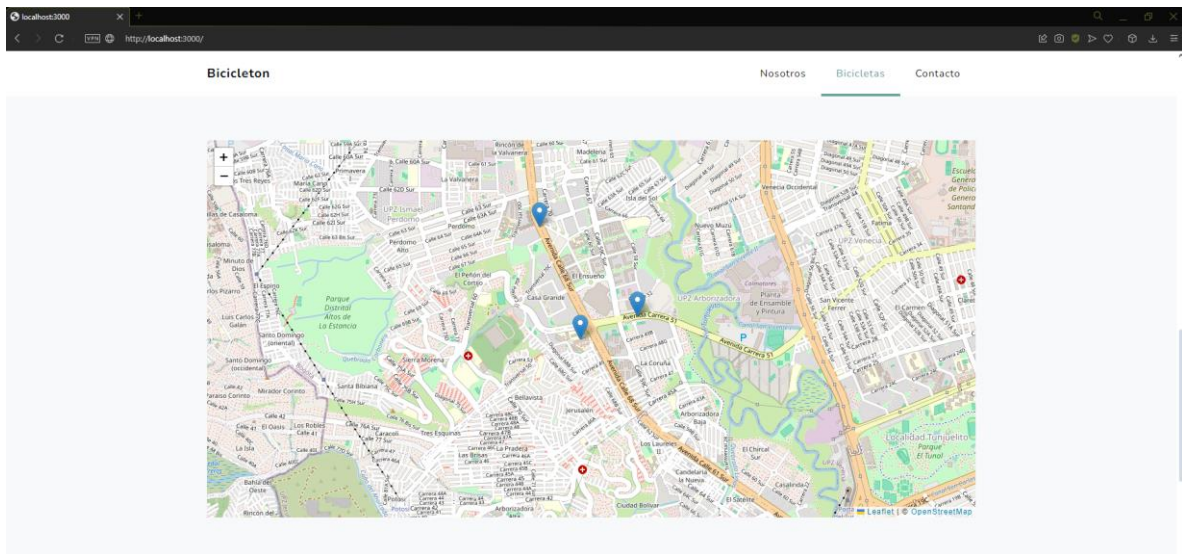
Longitud:

Y cambiando estos datos se actualizarán, como se muestra en la siguiente imagen la cual se cambio en los textos a naranja el color y montana el modelo

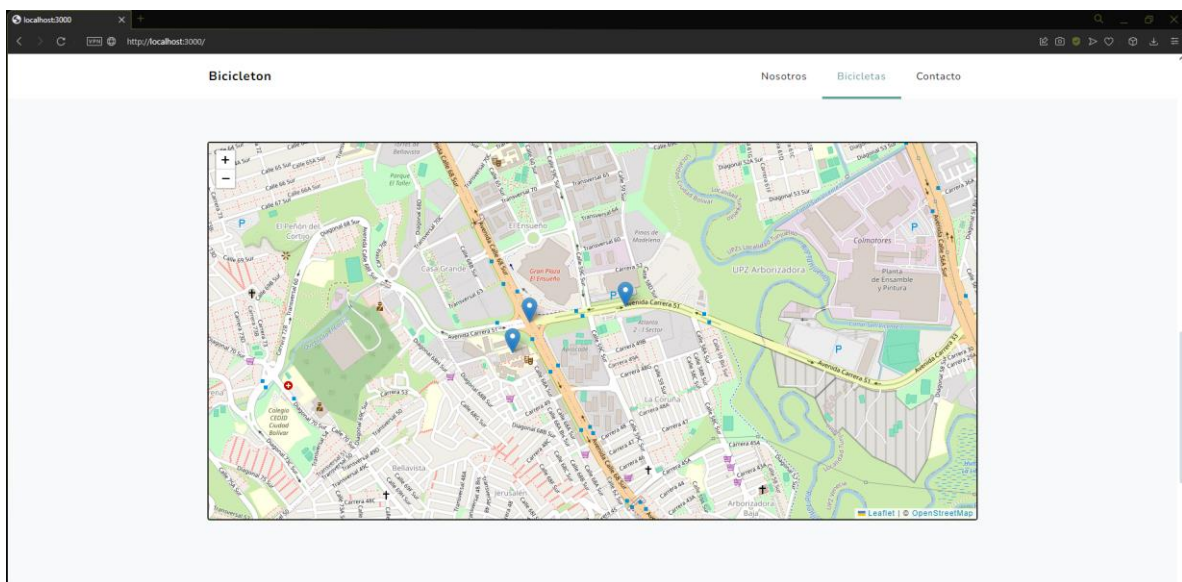
#	Color	Modelo	Ubicacion		
2	blanca	urbana	4.580993341635696,-74.15413000710157	Editar	<input type="button" value="ELIMINAR"/>
5	naranja	montana	4.5804644161547,-74.1574237570102	Editar	<input type="button" value="ELIMINAR"/>

Igualmente, esta actualización y el agregar también queda reflejado en el mapa de la página principal

Aquí la primera imagen antes de ingresar y actualizar



Y aquí luego de agregar y actualizar



CONCLUSIONES

- Se ha desarrollado una API funcional utilizando Node.js y Express, capaz de manejar solicitudes GET, POST, PUT y DELETE para manipular datos de las bicicletas.
- Se ha implementado un enrutador en Express para manejar las distintas rutas relacionadas con las bicicletas.
- Estructurar y organizar mucho mejor los documentos y archivos para que se entienda mejor el código.
- Se ha demostrado cómo interactuar con la API utilizando herramientas como un navegador web con el localhost y un plugin de Visual Studio Code "Thunder Client". Esto facilita las pruebas y el seguimiento del comportamiento de la API.
- Se conoció una nueva forma de generar vistas de paginas web en este caso remplazar el HTML y manejar PUG
- La API acepta y devuelve datos en formato JSON, lo cual es estándar y ampliamente utilizado en aplicaciones web modernas debido a su simplicidad y facilidad de uso.
- La implementación de una API de mapas en este caso Leaflet la cual facilita y ayuda mucho la creación y visualización de mapas de casi todo el mundo.
- La API implementa las operaciones básicas de CRUD (Crear, Leer, Actualizar, Eliminar) sobre las bicicletas, lo que permite una manipulación completa de la información.
- La combinación de Express, una estructura modular, una documentación clara, compilador como Visual Studio Code y herramientas de prueba como Thunder Client o PostMan hace que tanto la creación como el uso de una API sean procesos bastante accesibles y directos para los desarrolladores.