wawiwa

CONSTRUIR UN SERVIDOR USANDO EXPRESS NODE.JS

wawiwa

Antes de empezar, descarga Postman - Registrarse

Folder : chap9-express-introduction

Postman es un cliente API que permite a los usuarios fácilmente crear y guardar solicitudes HTTPs simples y complejas, así como leer sus respectivas respuestas.

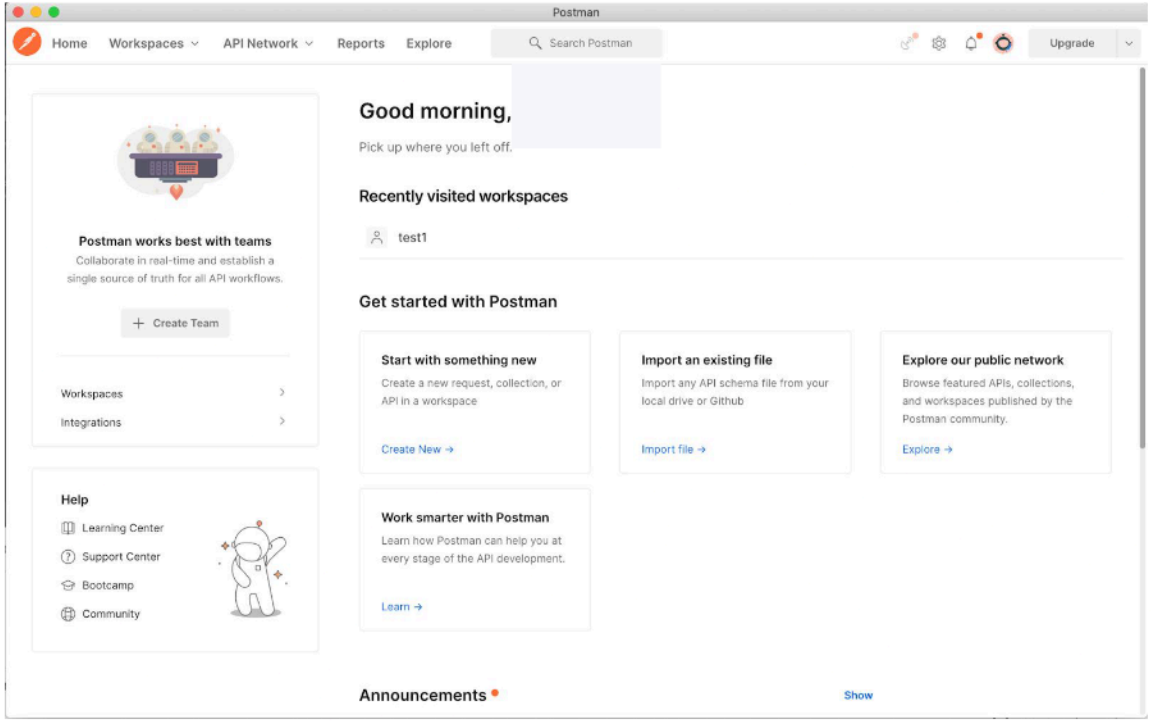
https://www.postman.com/

Descarga la aplicación:

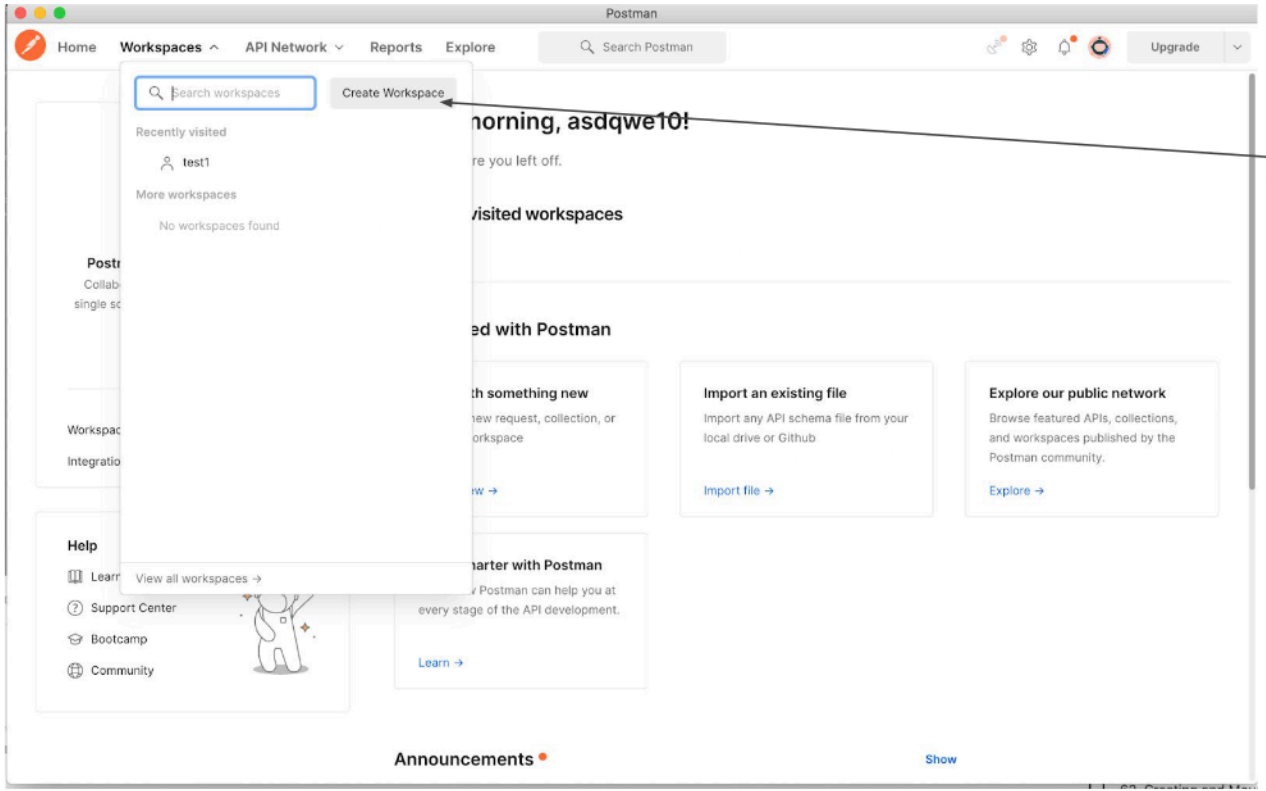
https://www.postman.com/downloads/

wawiwa

Antes de empezar, descarga Postman - Registrarse



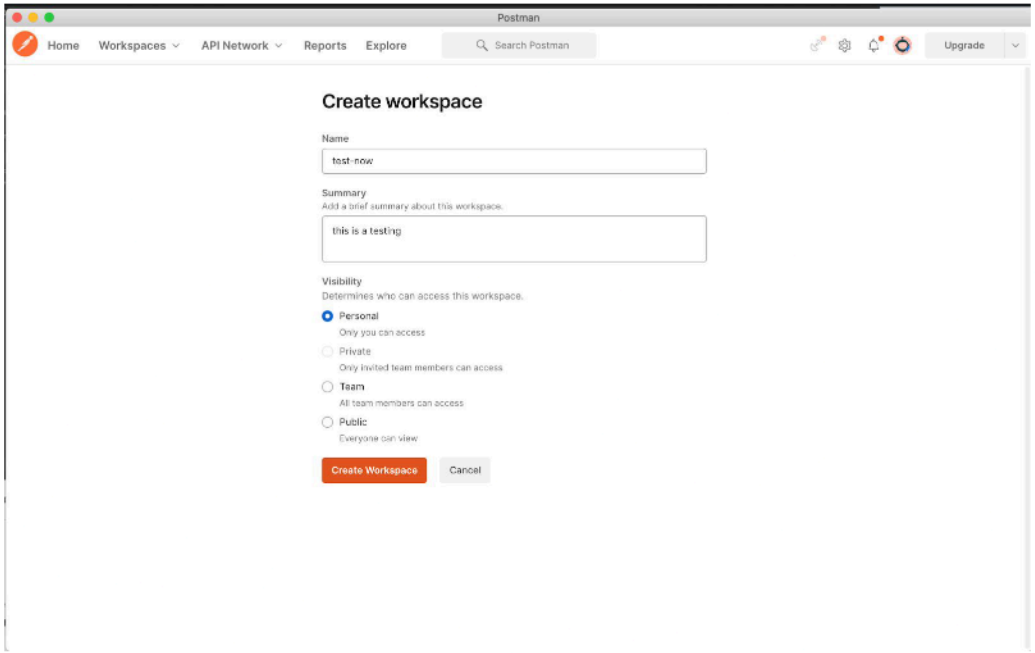
Antes de empezar, descarga Postman - Registrarse



Create workspace

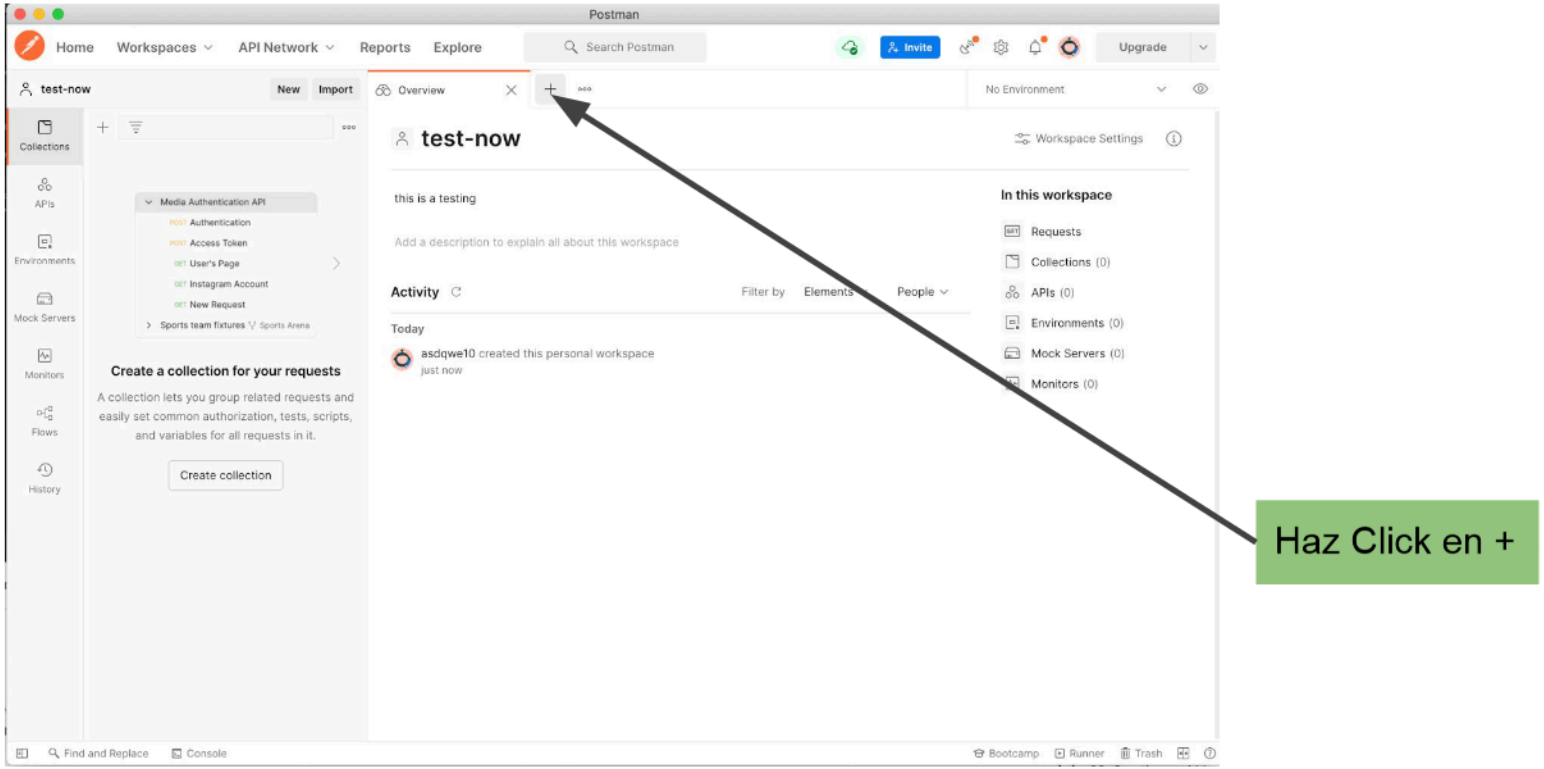
wawiwa

Antes de empezar, descarga Postman - Registrarse



Create workspace

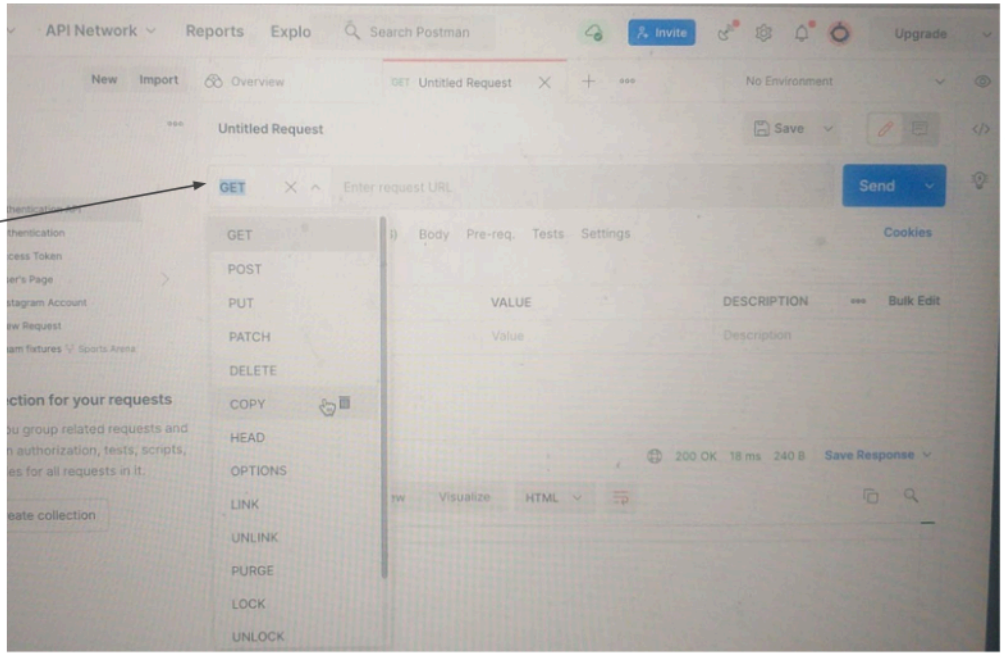
Antes de empezar, descarga Postman - Registrarse



wawiwa

Antes de empezar, descarga Postman - Registrarse

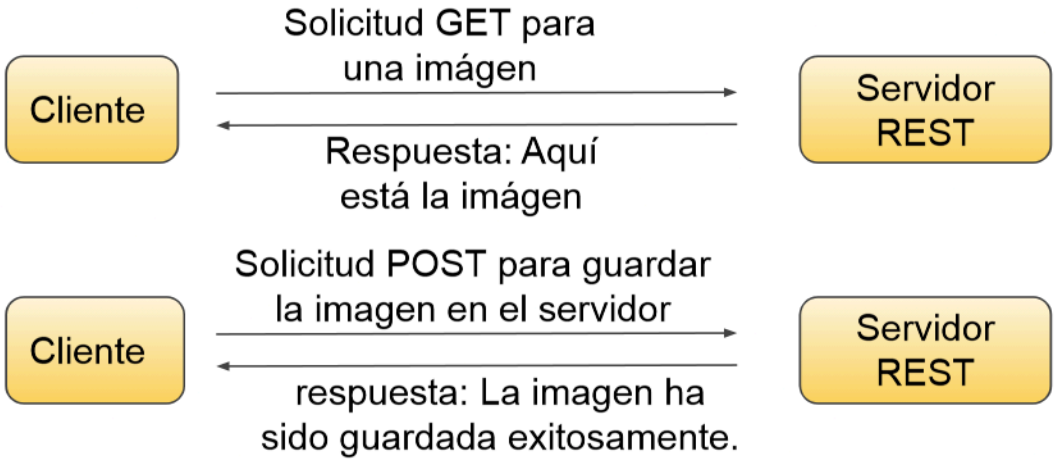
Solicita:
get, post,
etc…



¿Qué es Express?

Express es un módulo de node que nos permite construir un enrutamiento más simplemente.

Podemos construir un servidor que escuche solicitudes REST.



wawiwa

Primer app con express

Corre npm init en la terminal y luego npm install express.

En server.js:

```
const express = require("express");
const app = express();
const port = 3000;
app.get('/', (req, res) => {
      res.status(200).send('hello express');
});
app.listen(port);
```

Inicializa la variable para usar los métodos express

Escucha para obtener solicitudes con el url '/'

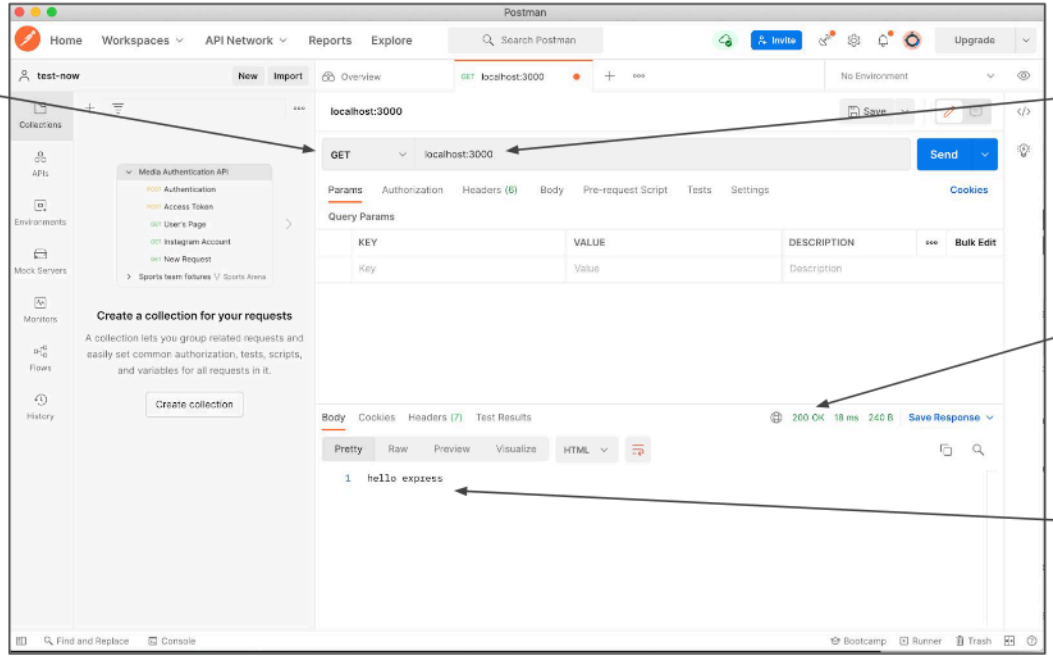
Códgio de Status

La respuesta puede también ser un json

wawiwa

Verificalo en postman

get



localhost:3000

Código de Status : 200

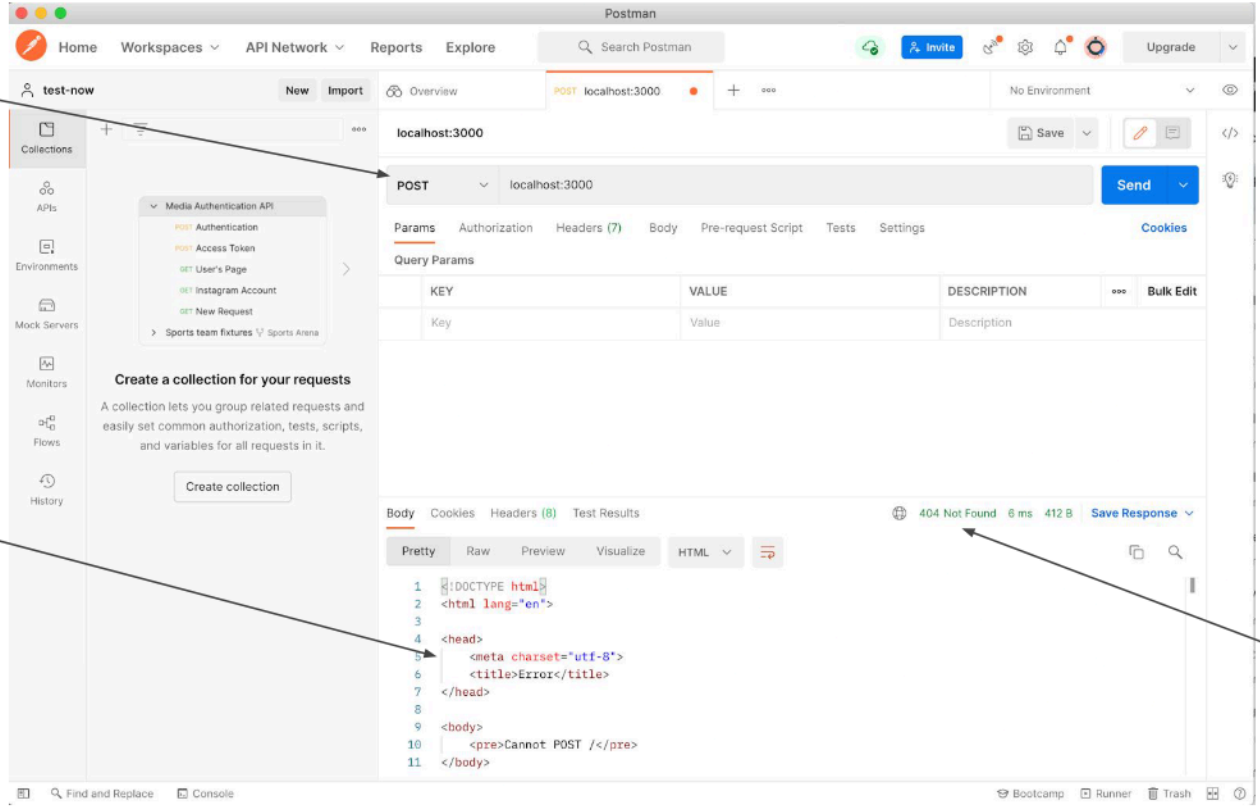
respuesta

wawiwa

Solicitud POST

post

Nuestra app no es compatible con la solicitud POST



404

wawiwa

Ahora nuestra app es compatible con la solicitud POST

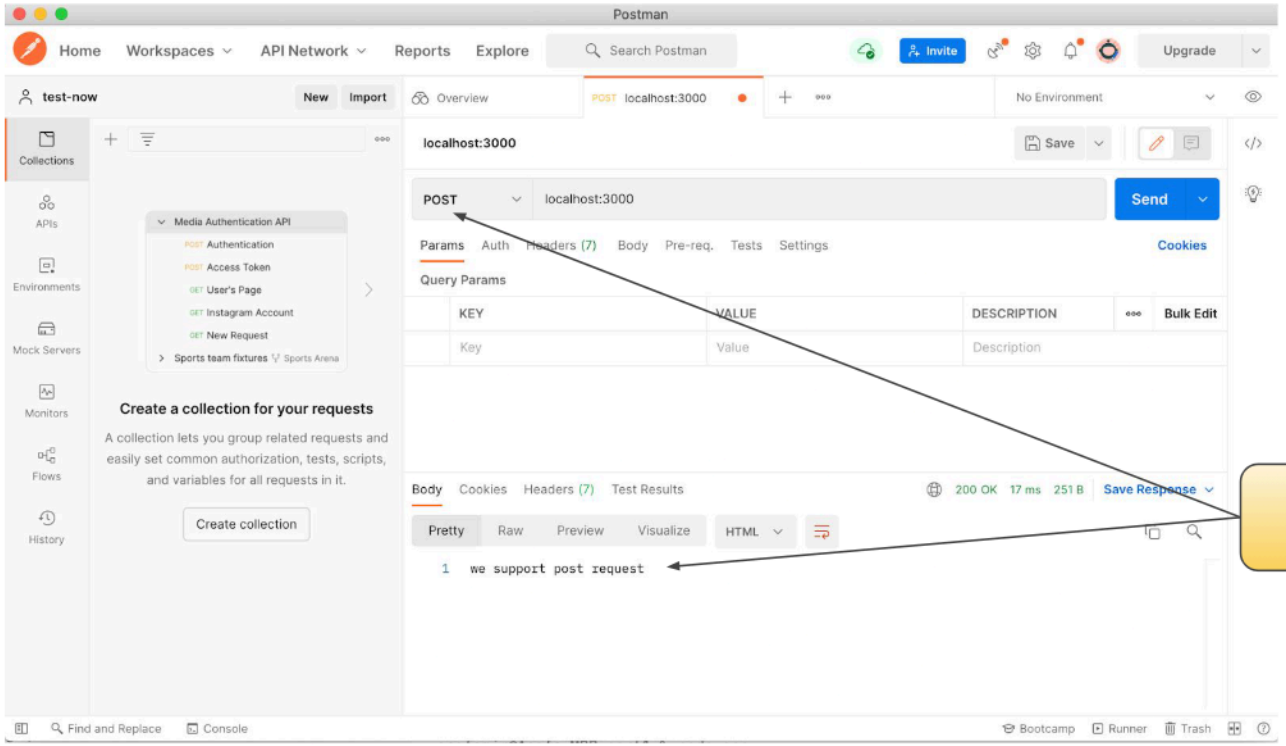
```
const express = require("express");
const app = express();
const port =  3000;

app.get('/', (req, res) => {
   res.status(200).send('hello express');
});

app.post('/', (req, res) => {
   res.status(200).send('we support post request');
});

app.listen(port);`
```

Escucha a solicitudes **get** con el url '/'

Escucha a solicitudes **post** con el url '/'

wawiwa

Now our app support post request



wawiwa

Ejemplo de Res json

```
const express = require("express");
const app = express();
const port = 3000;

app.get('/', (req, res) => {
    res.send('<html><head></head><body>Hello world</body></html>');
});

app.get('/api', (req, res) => {
    res.json({
        firstname: "asaf",
        lastname: "amir"
    });
});

app.listen(port);
```

Inicializa una variable para usar los métodos de express

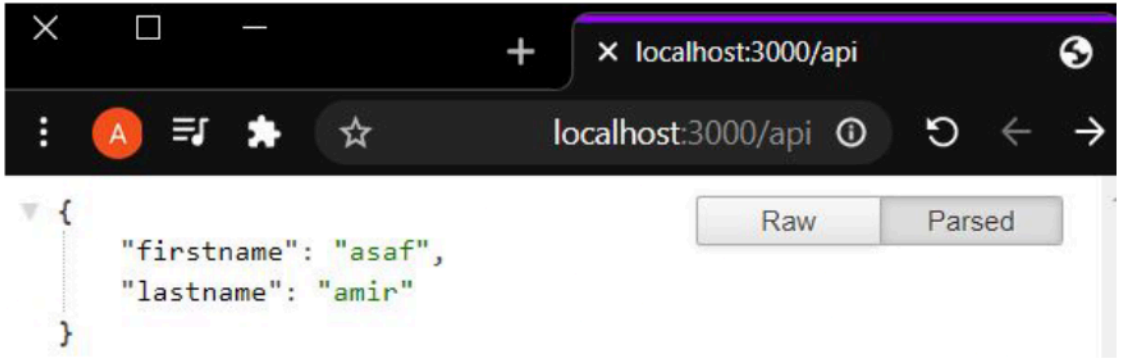
Escucha las solicitudes **get** con la url '/'

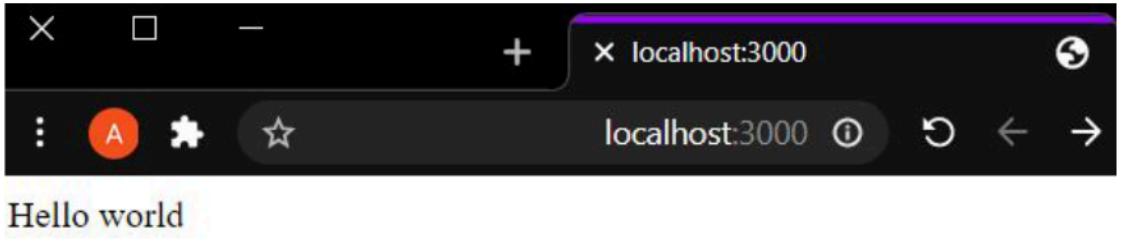
Escucha las solicitudes **get** con la url "/api'

La respuesta también puede ser un json

wawiwa

Construir un servidor usando Express





wawiwa

Hazlo tú mismo 1 - Construir un servidor usando Express

Cre un servidor con las siguientes solicitudes:

Request	Response	Method
localhost:3000/api/name	Tu nombre completo	get
localhost:3000/students/number	Número aleatorio entre 0 y 100	get
localhost:3000/courses/n1ton2	Número aleatorio entre 1000 y 2000	Post

wawiwa

Express - req.params

Podemos acceder a los parámetros que han sido enviados en la solicitud.

El ejemplo está en la siguiente diapositiva

wawiwa

Express - req.params

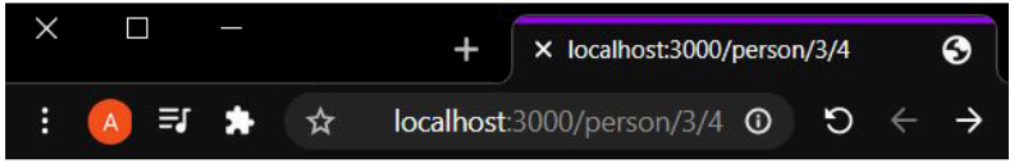
```
const express = require("express");
const app = express();
const port = 3000;

app.get('/', (req, res) => {
    res.send('<html><head></head><body>Hello world</body></html>');
});

app.get('/api', (req, res) => {
    res.json({ firstname: "asaf", lastname: "amir" });
});

app.get('/person/:id', (req, res) => {
    res.send('<html><head></head><body>hello person:' + req.params.id + '</body></html>');
});

app.get('/person/:id/:page', (req, res) => {
    res.send('<html><head></head><body>hello page:' + req.params.page + '</body></html>');
});

app.listen(port);
```

id es un parámetro que puede cambiar

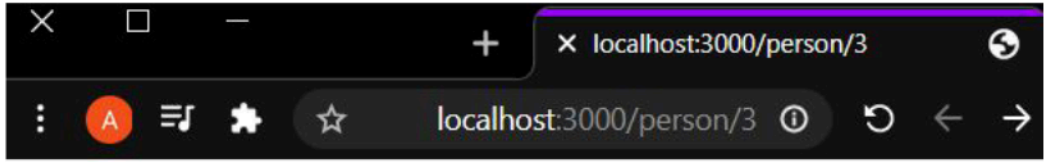
Hay dos parámetros en esta solicitud:
Uno después del segundo '/' y otra después del tercer '/'

wawiwa

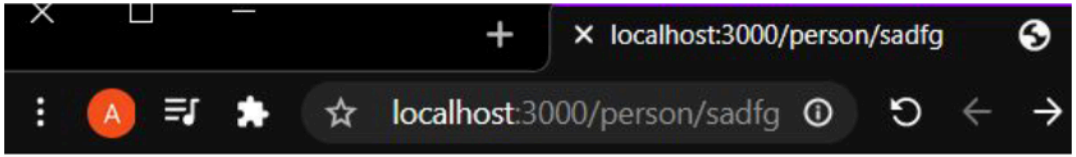
Express - req.params



hello page:4



hello person:3



hello person:sadfg

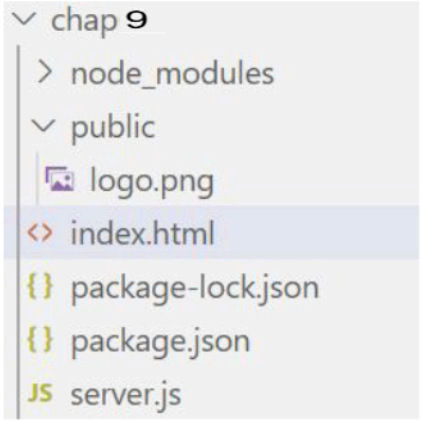
wawiwa

Renderizar archivos estáticos

Renderizar archivos estáticos del servidor usando express:

Crea un archivo llamado 'index.html'.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>hello</title>
</head>
<body>
    hello world
    <img src="/assets/logo.png">
</body>
</html>
```



Y crea una carpeta llamada 'public' y guarda una imagen dentro llamada called 'logo.png'

wawiwa

Renderizar archivos estáticos

Agrega a server.js:

```
const fs = require("fs");
app.use('/assets', express.static(__dirname + '/public'));


app.get('/firstpage', (req, res) => {
    fs.readFile("index.html", (err, buffer) => {
        const html = buffer.toString();
        res.setHeader('Content-Type', 'text/html');
        res.send(html);
    })
});
```

Dice a express que busque en los archivos estáticos bajo /public en el directorio actual, cuando reciba el path /assets

Guarda el archivo html como un string en una variable para que la podamos usar después

Muestra la el contenido de la respuesta como un texto HTML al cliente

wawiwa

ARQUITECTURA DE REST API

MÉTODOS HTTP

POST - CREATE (crear)

GET - READ (leer)

PUT - UPDATE (actualizar)

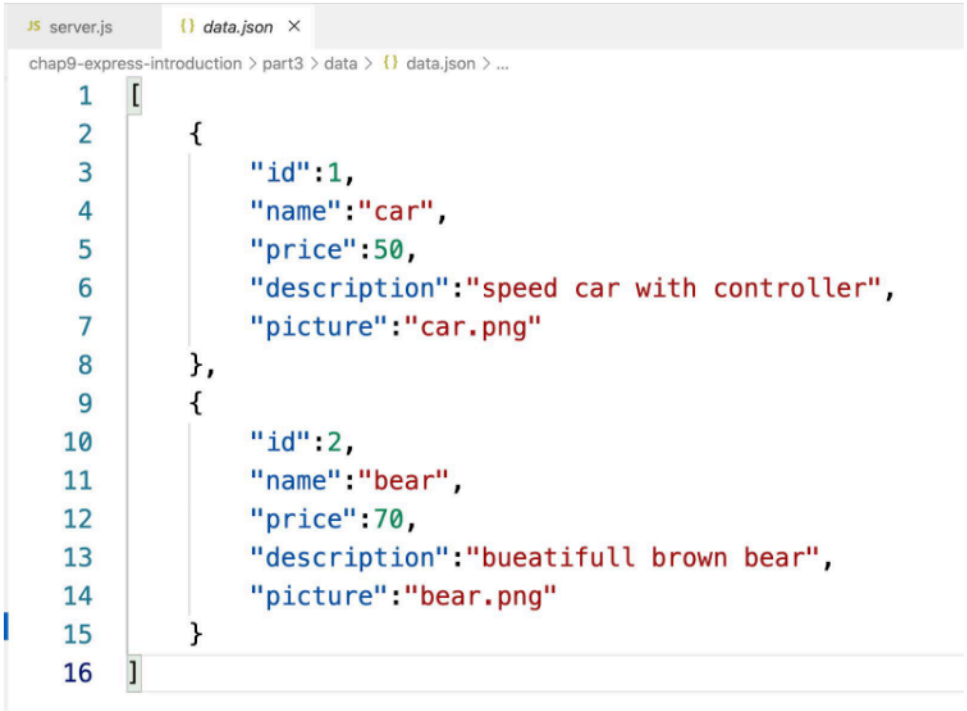
DELETE - DELETE (borrar)

CRUD - CREATE, READ, UPDATE, DELETE

Mandar la información como json

wawiwa

Ejemplo de CRUD



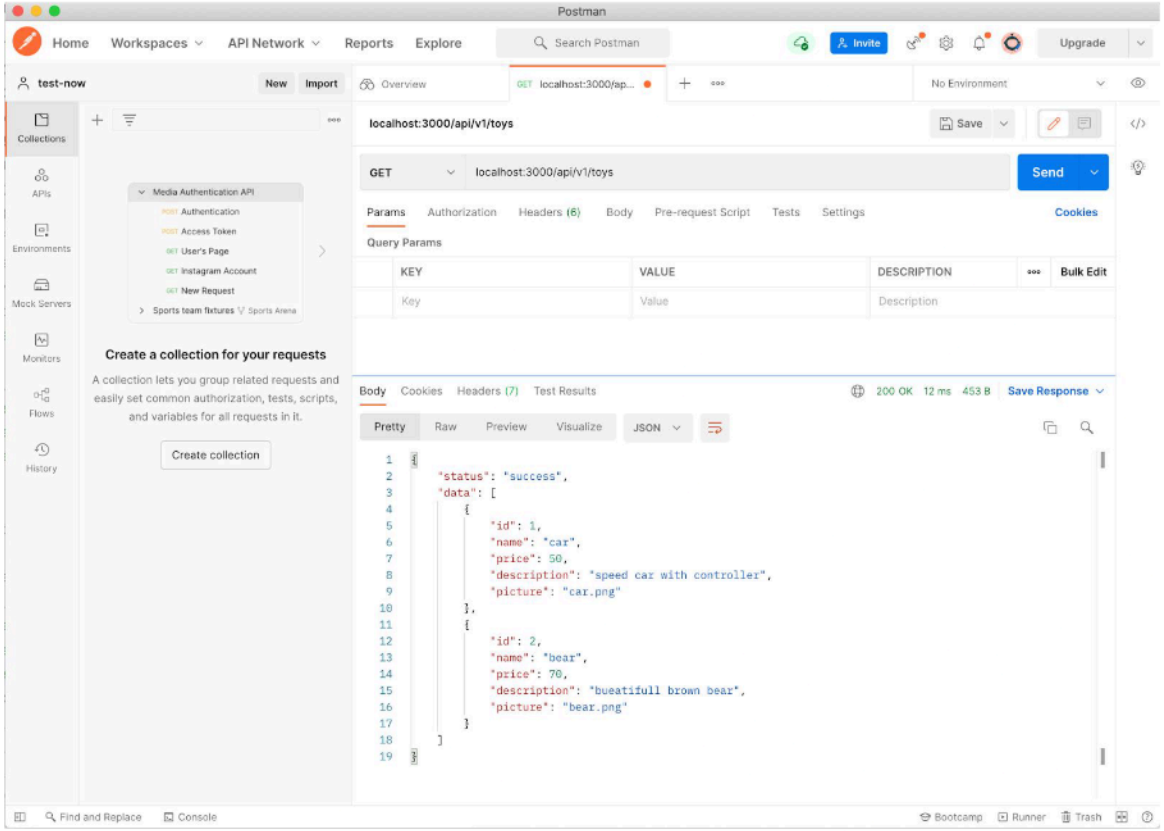
wawiwa

Ejemplo de CRUD - obtener todos los “toys”

```
const express = require("express");
const fs = require("fs");
const app = express();
const port = 3000;
app.use('/assets', express.static(__dirname + '/public'));
const toys =
    JSON.parse(fs.readFileSync("./data/data.json", 'utf-8'));

app.get('/api/v1/toys', (req, res) => {
    res.status(200).json({
        status: "success",
        data: toys
    });
});

app.listen(port);
```

wawiwa

localhost:3000/api/v1/toys



wawiwa

Ejemplo de CRUD - Agregar un nuevo “toy”

```
const express = require("express");
const fs = require("fs");
const app = express();
app.use(express.json());
const port = 3000;

app.use('/assets', express.static(__dirname + '/public'));

const toys = JSON.parse(fs.readFileSync("./data/data.json", 'utf-8'));

app.post('/api/v1/toys', (req, res) => {
      toys.push(Object
            .assign({ id: toys[toys.length - 1].id + 1 }, req.body));

      fs.writeFile("./data/data.json", JSON.stringify(toys), err =>
            res.status(201).json({ status: "success", data: toys }));
})

app.listen(port);
```

Agrega el body a la solicitud. Ahora req .body contiene la información de post

Crea un id para el próximo “toy”

Crea un nuevo objeto y le hace “push” al arreglo

Escribe el objeto en el archivo

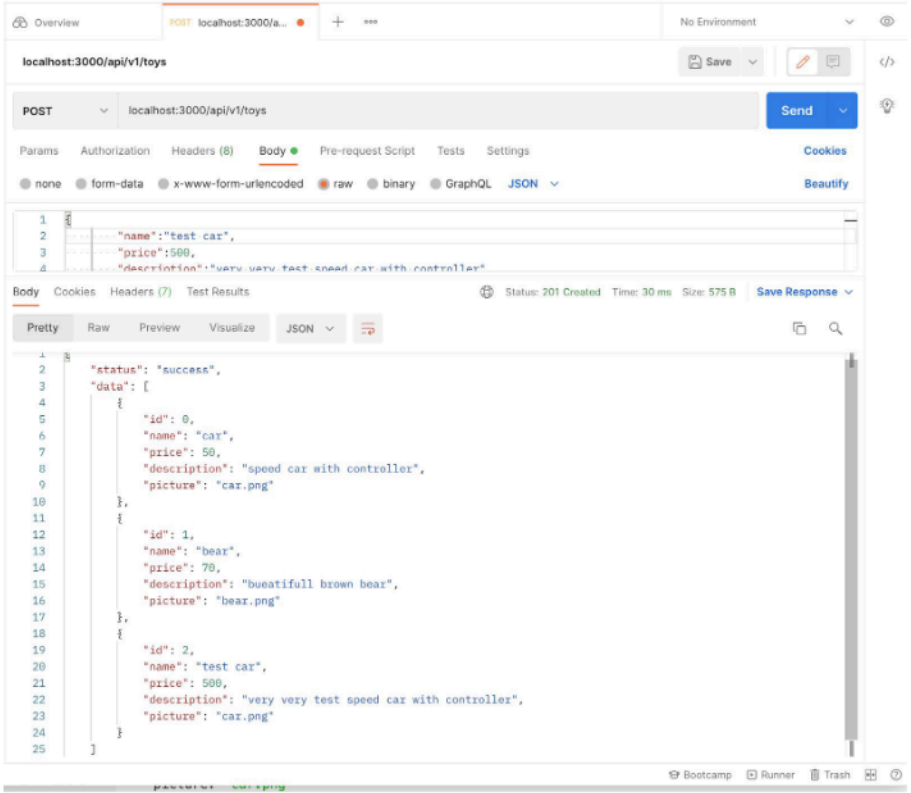
wawiwa

localhost:3000/api/v1/toys

1. El mismo url
2. Pero ahora es post req

raw - > body

res



wawiwa

Ejemplo de CRUD - obtener “toy” por su id

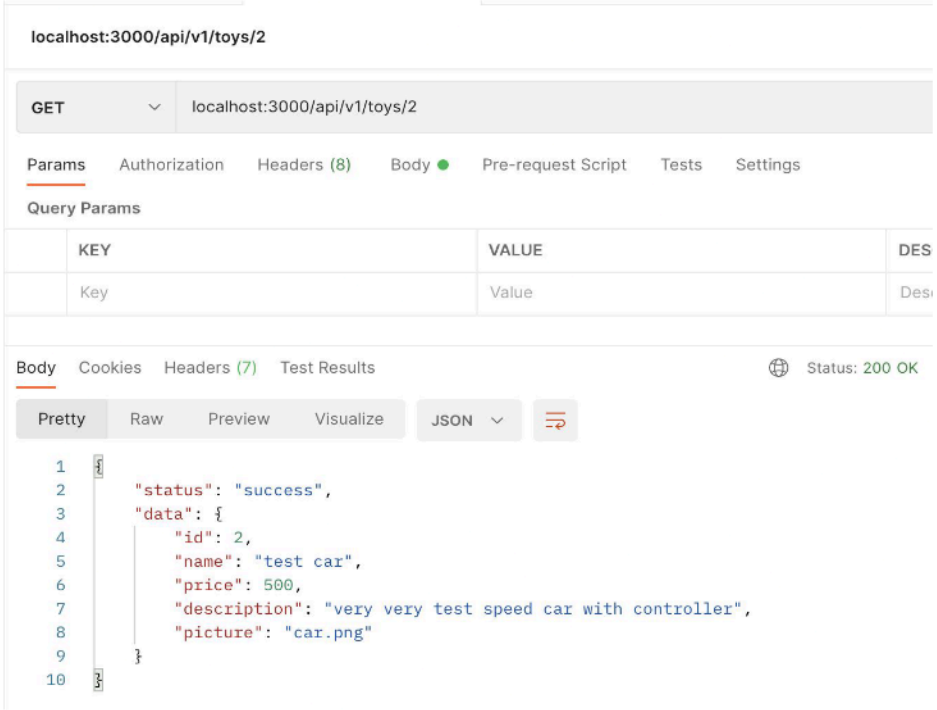
```
app.get('/api/v1/toys/:id', (req, res) => {
    const id = Number(req.params.id)
    const toy = toys.find(el => el.id === id);

    if (!toy) {
        return res.status(404).json({
            status: 'fail',
            message: 'id not exist'
        });
    }

    res.status(200).json({
        status: 'success',
        data: toy
    });
});
```

Si no existe el “toy”

Respuesta: el “toy” correcto

wawiwa

localhost:3000/api/v1/toys/2



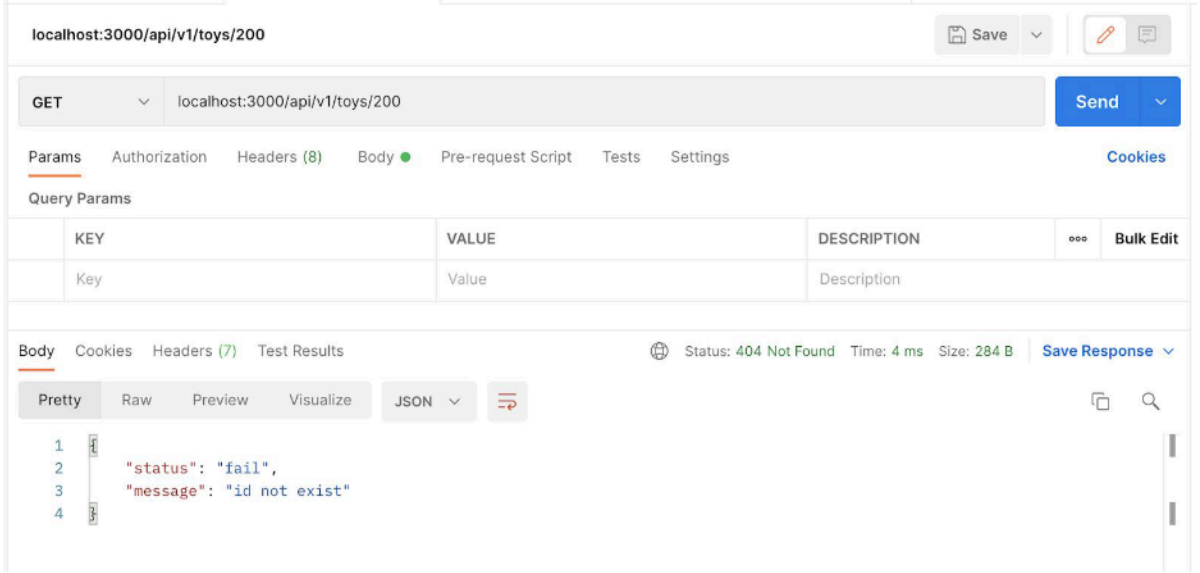
wawiwa

localhost:3000/api/v1/toys/200 - falla

Id 200 no existe

404

Respuesta



wawiwa

Ejemplo de CRUD - patch(actualizar) “toy” por id

```
app.patch('/api/v1/toys/:id', (req, res) => {
      const id = Number(req.params.id)
      const toy = toys.find(el => el.id === id);

      if (!toy) {
            return res.status(404).json({
                  status: 'fail',
                  message: 'id not exist'
            });
      }

      toys[id].name = req.body.name;
      toy.description = req.body.description;
      toy.picture = req.body.picture;
      toy.price = Number(req.body.price);
      toys[id] = toy;

      fs.writeFile("./data/data.json", JSON.stringify(toys), err => {
            res.status(200).json({
                  status: "success",
                  data: toys[id]
            });
      });
});
```

Si no existe el “toy”

Actualiza el “toy”

Escribir en el archivo

Respuesta: el “Toy” actualizado

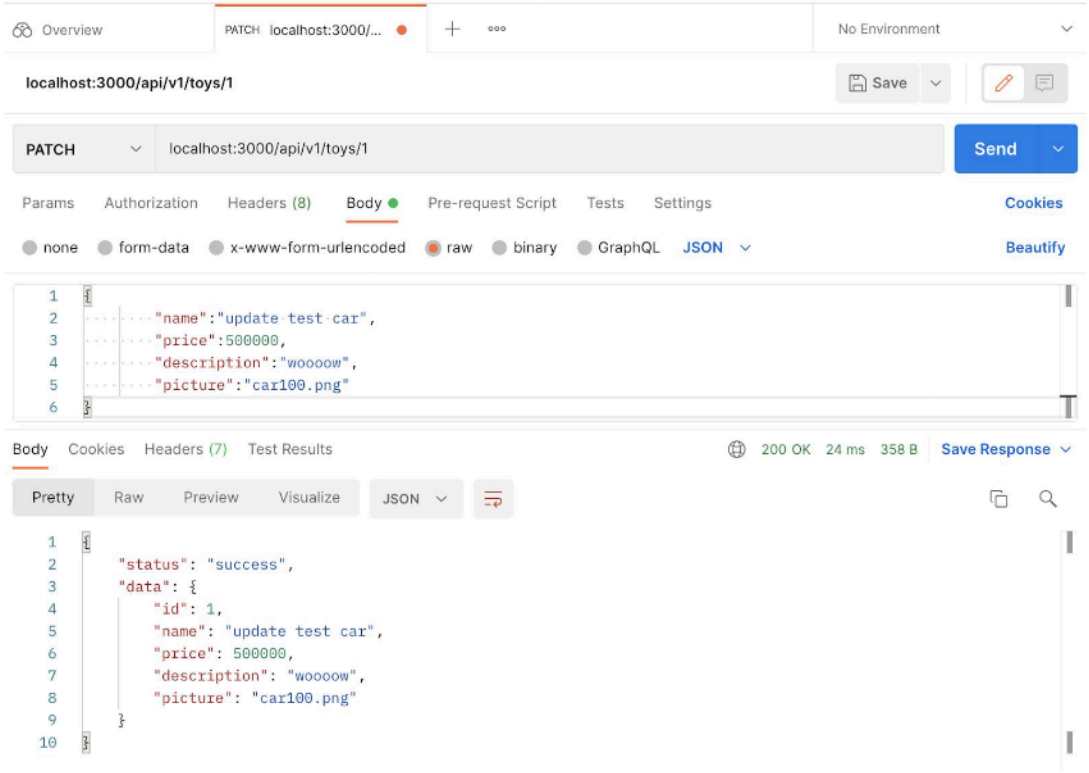
wawiwa

localhost:3000/api/v1/toys/1

1. El mismo url
2. Pero ahora es patch req

raw - > body

res



wawiwa

data.json

```
[
    { "id": 0, "name": "car", "price": 50,
        "description": "speed car with controller", "picture": "car.png" },
    {"id":1, "name": "update test car", "price": 500000,
        "description": "woooow", "picture": "car100.png" },
    {"id": 2, "name": "test car", "price": 500,
        "description": "very very test speed car with controller", "picture": "car.png" }
]
```

wawiwa

Ejemplo de CRUD - borrar un “toy” por su id

```
app.delete('/api/v1/toys/:id', (req, res) => {
      const id = Number(req.params.id)
      const toy = toys.find(el => el.id === id);

      if (!toy) {
            return res.status(404).json({
                  status: 'fail',
                  message: 'id not exist'
            });
      }

      toys.splice(toys.findIndex(obj => obj.id === id), 1);

      fs.writeFile("./data/data.json", JSON.stringify(toys),
      err => {
            res.status(200).json({
                  status: "success delete",
                  data: toy
            });
      });
});
```

Encuentra el índice del “toy”

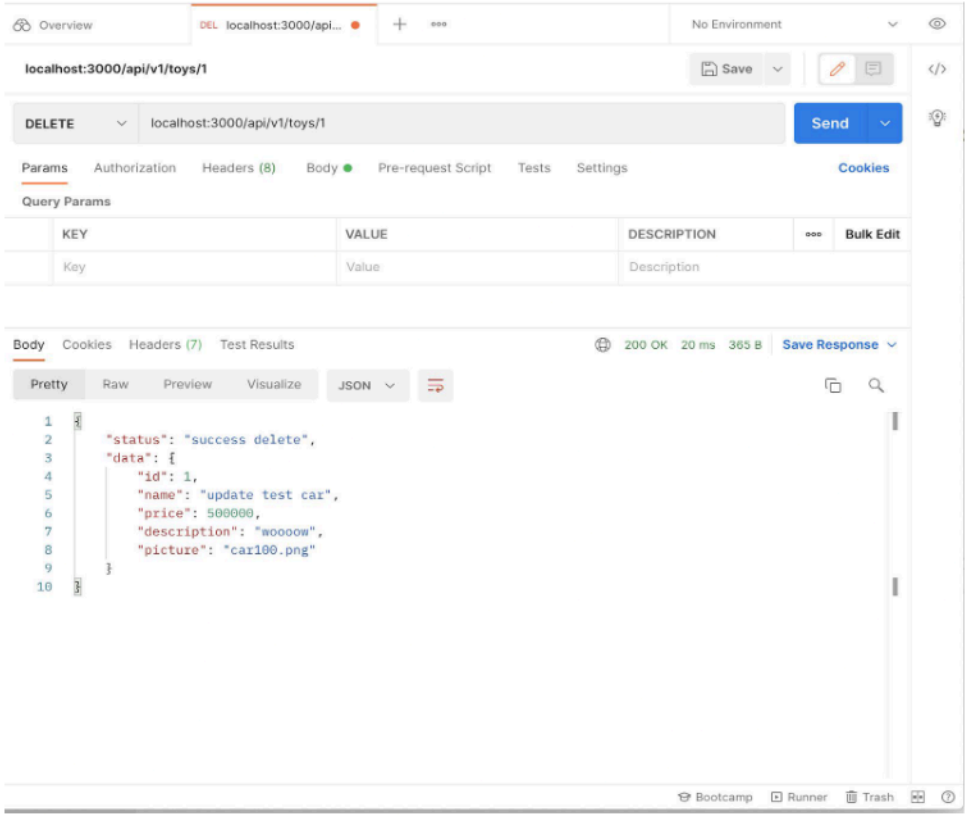
Borra el “toy” del arreglo

Respuesta: el “toy” borrado

wawiwa

localhost:3000/api/v1/toys/1

1. El mismo url
2. Pero ahora es delete req



"Toy" borrado

wawiwa

Hazlo tú mismo 2 - data/player.json

```
[
    {
        "id": 0,
        "firstname": "Andre",
        "lastname": "Iguodala",
        "age": 37,
        "Team": "Warriors"
    },
    {
         "id": 1,
         "firstname": "Carmelo",
         "lastname": "Anthony",
         "age": 37,
         "Team": "Lakers"
     }
]
```

wawiwa

Hazlo tú mismo 2 - Construir un servidor usando Express

Crea un servidor Express que responda a las siguientes solicitudes:

Solicitud	Respuesta	Método	Comentario
`/api/v1/players`	Todos los “players”	Get	Obtener todos los “players” de players.json
`/api/v1/players`	El “player creado”	Post	Crea un nuevo Player
`/api/v1/players/:id`	“Player” por su id	Get	Obtener el “player” por su id
`/api/v1/players/:id`	Actualizar “player”	Patch	Actualizar “player” por su id
`/api/v1/players/:id`	Borrar “player”	Delete	Borrar “player” por su id

wawiwa

REFACTORIZACIÓN DE CÓDIGO PARTE 1

wawiwa

Refactorización

Refactorización de código es definido como el proceso de reestructurar código de computadora sin cambiar o agregar a su comportamiento externo y funcionalidad. Existen muchas maneras de refactorizar, pero la mayoría de las veces consiste en aplicar una serie de métodos estandarizados.

wawiwa

Refactorizando el código de getAllToys

```
const express = require("express");
const fs = require("fs");
const app = express();
app.use(express.json());
const port = 3000;

const getAllToys = (req, res) => {
    res.status(200).json({
        status: "success",
        data: toys
    });
};

app.get('/api/v1/toys', getAllToys);
app.listen(port);
```

wawiwa

Refactorizando el código parte createToy

```
const createToy = (req, res) => {
     console.log(req.body)

     const newToy = Object.assign(
          { id: toys[toys.length - 1].id + 1 },
               req.body);

     toys.push(newToy);

     fs.writeFile("./data/data.json", JSON.stringify(toys),
          err => { res.status(201).json({
               status: "success",
               data: toys
          });
     });
}

app.post('/api/v1/toys', createToy);
```

wawiwa

Refactorizando el código parte getToyById

```
const getToyById = (req, res) => {
     const id = Number(req.params.id);
     const toy = toys.find(el => el.id === id);

     if (!toy) {
          return res.status(404).json({
               status: 'fail',
               message: 'id not exist'
          });
     };

     res.status(200).json({
          status: 'success',
          data: toy
     });
};

app.get('/api/v1/toys/:id', getToyById);
```

wawiwa

Refactorizando el código parte updateToyById

```
const updateToyById = (req, res) => {
      const id = Number(req.params.id);
      const toy = toys.find(el => el.id === id);

      if (!toy) {
            return res.status(404).json({
                  status: 'fail',
                  message: "id not exist"
            });
      }

      toys[id].name = req.body.name;
      toy.description = req.body.description;
      toy.picture = req.body.picture;
      toy.price = Number(req.body.price);
      toys[id] = toy;

      fs.writeFile("./data/data.json", JSON.stringify(toys), err => {
            res.status(200).json({
                  status: "success",
                  data: toys[id]
            });
      });
};

app.patch('/api/v1/toys/:id', updateToyById);
```

wawiwa

Refactorizando el código parte deleteToyById

```
const deleteToyById = (req, res) => {
      const id = Number(req.params.id);
      const toy = toys.find(el => el.id === id);

      if (!toy) {
            return res.status(404).json({
                  status: 'fail',
                  message: 'id not exist'
            });
      }

      const index = toys.findIndex(obj => obj.id === id);
      toys.splice(index, 1);

      fs.writeFile("./data/data.json", JSON.stringify(toys), err => {
            res.status(200).json({
                  status: "success delete",
                  data: toy
            })
      });
};

app.delete('/api/v1/toys/:id',deleteToyById);
```

wawiwa

El servidor final después de refactorizar - parte 1

```
1  var express = require("express");
2  var fs = require("fs");
3  var app = express();
4  app.use(express.json())
5  var port = 3000;
6  let getAllToys=(req, res)=>{…
12 }
13 let createToy=(req, res)=>{…
25 }
26 let getToyById=(req, res)=>{…
39 }
40
41 let updateToyById=(req, res)=>{…
63 }
64 let deleteToyById=(req, res)=>{…
82 }
83 app.get('/api/v1/toys', getAllToys);
84 app.post('/api/v1/toys',createToy);
85 app.get('/api/v1/toys/:id',getToyById);
86 app.patch('/api/v1/toys/:id',updateToyById);
87 app.delete('/api/v1/toys/:id',deleteToyById);
88 app.listen(port);
```

wawiwa

Hazlo tú mismo 3- refactoriza ejercicio 2

Crea un servidor Express que responda a las siguientes solicitudes:

Solicitud	Respuesta	Método	Comentario
`/api/v1/players`	Todos los “players”	Get	Obtener todos los “players” de players.json
`/api/v1/players`	El “player creado”	Post	Crea un nuevo Player
`/api/v1/players/:id`	“Player” por su id	Get	Obtener el “player” por su id
`/api/v1/players/:id`	Actualizar “player”	Patch	Actualizar “player” por su id
`/api/v1/players/:id`	Borrar “player”	Delete	Borrar “player” por su id


wawiwa
¿Preguntas?