



wawiwa

CONSTRUIR UN SERVIDOR USANDO EXPRESS

NODE.JS

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/>

Antes de empezar, descarga Postman - Registrarse

The screenshot shows the Postman application window. At the top, there's a navigation bar with icons for Home, Workspaces, API Network, Reports, Explore, and Search Postman. To the right of the search bar are buttons for Upgrade, a gear icon, and three colored dots (red, yellow, green). Below the navigation bar, there's a "Good morning," message with a placeholder "Pick up where you left off." A "test1" workspace is listed under "Recently visited workspaces".

Postman works best with teams
Collaborate in real-time and establish a single source of truth for all API workflows.

Get started with Postman

- Start with something new**
Create a new request, collection, or API in a workspace.
[Create New](#)
- Import an existing file**
Import any API schema file from your local drive or GitHub.
[Import file](#)
- Explore our public network**
Browse featured APIs, collectors, and workspaces published by the Postman community.
[Explore](#)

Help

- [Learning Center](#)
- [Support Center](#)
- [Bootcamp](#)
- [Community](#)

Announcements [Show](#)

Antes de empezar, descarga Postman - Registrarse

The screenshot shows the Postman application window. At the top, there's a navigation bar with icons for Home, Workspaces, API Network, Reports, Explore, and Create Workspace. A search bar is also present. Below the navigation bar, there's a sidebar with sections for Recently visited workspaces (including 'test1') and a note about no workspaces found. The main area has a large green button labeled 'Create workspace'. To the right, there are several cards: one for 'Explore our public network' (Postman community), one for 'Import an existing file' (local drive or GitHub), one for 'Add something new' (new request, collection, or workspace), and one for 'Start with Postman' (Learn, Support Center, Bootcamp, Community). On the far right, there's an 'Announcements' section and a 'Show' button.

Create workspace

Postman

Home Workspaces API Network Reports Explore Create Workspace

Search Postman

norming, asddqwe10!

re you left off.

visited workspaces

No workspaces found

Postman

Collab

single sc

Search Workspaces

Recent workspaces

test1

More workspaces

Add with Postman

Import an existing file

Import any API schema file from your local drive or GitHub

Import file →

Add something new

New request, collection, or workspace

New →

Start with Postman

View all workspaces →

Support Center

Bootcamp

Community

Learn →

Explore →

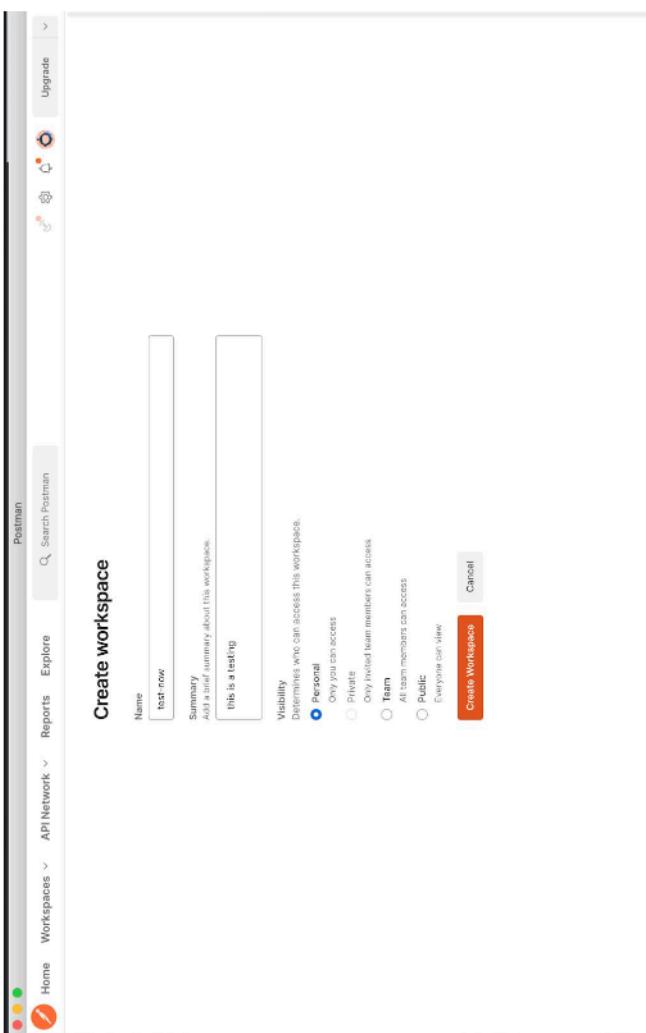
Explore our public network

Browse featured APIs, collections, and workspaces published by the Postman community.

Show

Announcements

Antes de empezar, descarga Postman - Registrarse



Create
workspace

Antes de empezar, descarga Postman - Registrarse

The screenshot shows the Postman application interface. At the top, there is a navigation bar with icons for Home, Workspaces, API Network, Reports, Explore, New, Import, Overview, and a search bar labeled "Search Postman". To the right of the search bar are buttons for "Invite", "Import", "Upgrade", and "Logout". Below the navigation bar, it says "No Environment" and "In this workspace". A large green callout box with the text "Haz Click en +" points to the "New" button in the top navigation bar.

In this workspace

- Requests (0)
- Collections (0)
- APIs (0)
- Environments (0)
- Mock Servers (0)
- Monitors (0)

Activity

today asdpose10 created this personal workspace just now

Create a collection for your requests

A collection lets you group related requests and easily see common authorization, tests, scripts, and variables for all requests in it.

Create collection

Postman

Home Workspaces API Network Reports Explore New Import Overview Search Postman No Environment

test-now

this is a testing Add a description to explain all about this workspace

Media Authentication API

- new Authentication
- new Access Token
- new User's Page
- new Instagram Account
- new Request
- Sports team Duties V. Sports Areas

Activity

today asdpose10 created this personal workspace just now

Create a collection for your requests

A collection lets you group related requests and easily see common authorization, tests, scripts, and variables for all requests in it.

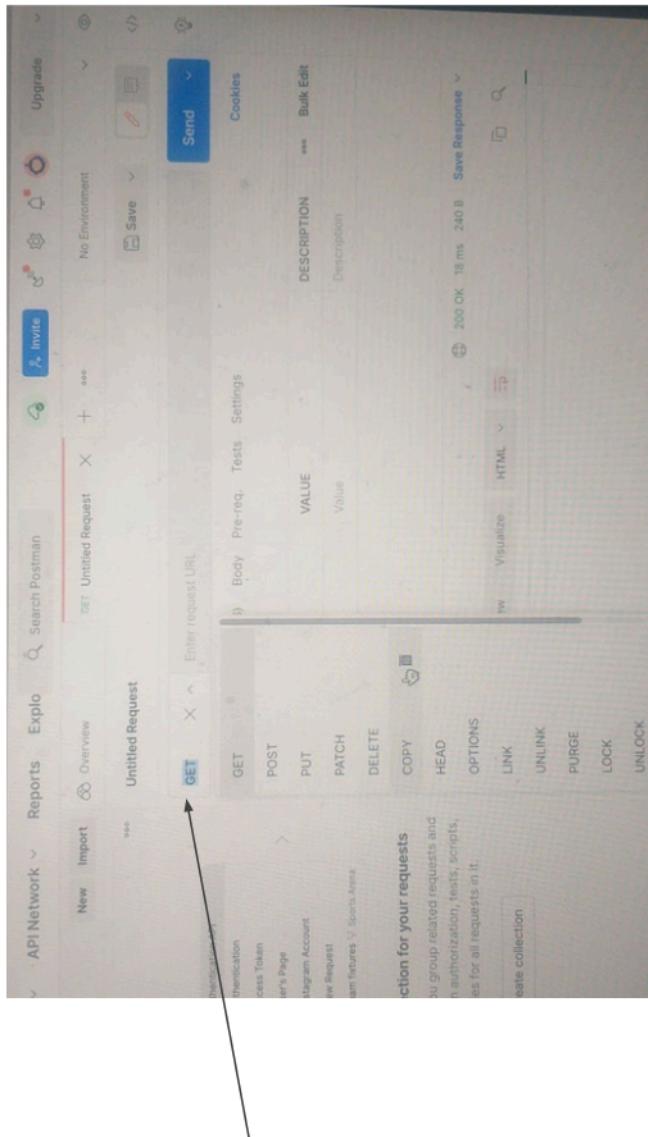
Create collection

Monitors

1+2 Flows

History

Antes de empezar, descarga Postman - Registrarse

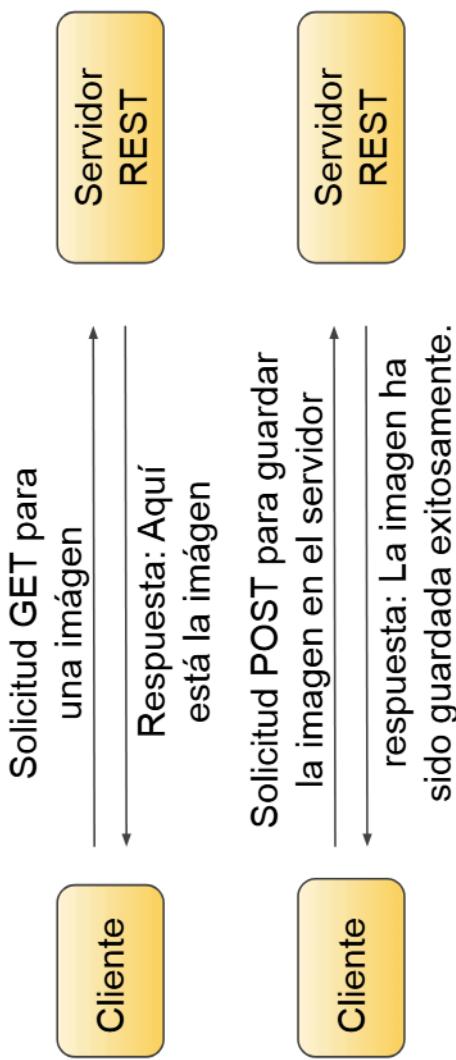


Solicita:
get, post,
etc...

¿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.



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 '/'

La respuesta puede también ser un json

Código de Status

Verificalo en postman

get

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Postman' and several icons. Below it, a search bar says 'Search Postman'. The main area is titled 'test-new'.

Collections:

- APIS
- Environments
- Mock Services
- Metrics

Requests:

- GET localhost:3000**: Headers tab (Authorization, Pre-request Script, Tests, Settings), Body tab (Value, Value), Description tab (Description).
- POST localhost:3000**: Headers tab (Content-Type, Authorization, Pre-request Script, Tests, Settings), Body tab (JSON, Value), Description tab (Description).

Environments:

- API
- Authentication
- Access Token
- User Role
- Markware Account
- New Request
- Sports Game Fixture
- Sports Arena

Tools:

- Send
- Cookies
- Save
- Bulk Edit

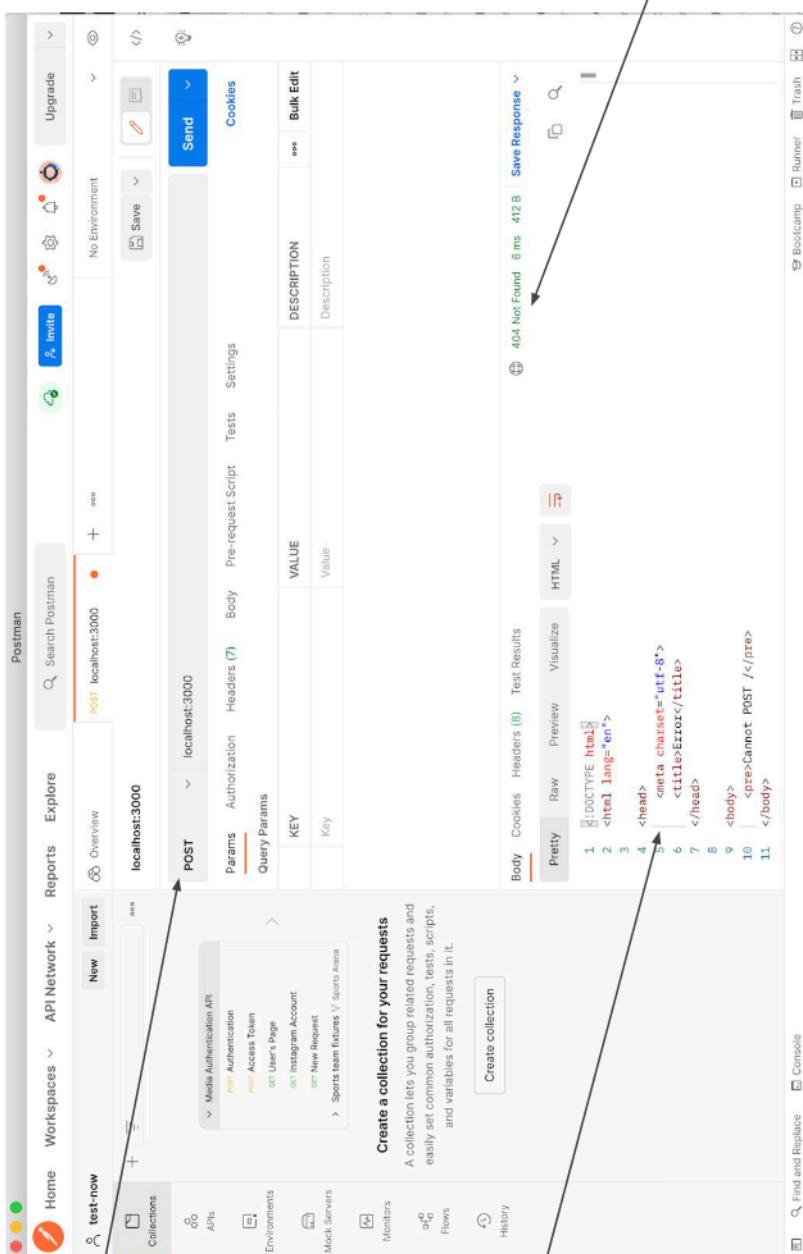
Bottom right:

- 200 OK 18 ms 246 B Save Response
- HTML
- 1 hello express
- Find and Replace
- Console

localhost:3000

Código de Status : 200

respuesta



Nuestra app no es compatible con la solicitud POST

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 '/'

Now our app support post request

The screenshot shows the Postman application interface. In the top navigation bar, there are icons for Home, Workspaces, API Network, Reports, Explore, and Search postman. Below the navigation, there are buttons for Import, New, and Collections. A 'test-now' button is highlighted in orange.

The main workspace shows a POST request to 'localhost:3000'. The request details are as follows:

- Method: POST
- URL: localhost:3000
- Headers:
 - Params
 - Auth
 - Headers (7)
 - Body
 - Pre-req.
 - Tests
 - Settings

The 'Headers' section contains the following key-value pairs:

KEY	VALUE
Key	Value

The 'Body' section shows a JSON object:

```
{ "key": "value" }
```

The response details are as follows:

- Status: 200 OK
- Time: 17 ms
- Size: 251 B
- Save Response

A large yellow callout box with the text 'Support post req' is positioned over the 'Send' button and the response area.

At the bottom of the interface, there are buttons for Find and Replace, Console, and other application controls.

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

Construir un servidor usando Express

The screenshot shows two network requests made to the endpoint `localhost:3000/api`. The first request is a POST method, and the second is a GET method. Both requests have a status of 200 OK.

Raw Request (POST):

```
POST /api HTTP/1.1
Content-Type: application/json
Accept: application/json

{
  "firstname": "asaf",
  "lastname": "amir"
}
```

Parsed Response (POST):

```
{
  "firstname": "asaf",
  "lastname": "amir"
}
```

Raw Request (GET):

```
GET /api HTTP/1.1
Accept: application/json
```

Parsed Response (GET):

```
Hello world
```

Hazlo tú mismo 1 - Construir un servidor usando Express

Cre un servidor con las siguientes solicitudes:

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

Express - req.params

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

El ejemplo está en la siguiente diapositiva

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} + ${req.params.page}</body></html>`);
});

app.get('/person/:id/:page', (req, res) => {
  res.send(`<html><head></head><body>Hello page: ${req.params.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 tercero '/'

Express - req.params

X	□	-	+ x localhost:3000/person/3/4	⟳
:	A	≡	★	localhost:3000/person/3/4 ① ↵ ← →
hello page:4				
X	□	-	+ x localhost:3000/person/3	⟳
:	A	≡	★	localhost:3000/person/3 ① ↵ ← →
hello person:3				
X	□	-	+ x localhost:3000/person/sadfg	⟳
:	A	≡	★	localhost:3000/person/sadfg ① ↵ ← →
hello person:sadfg				

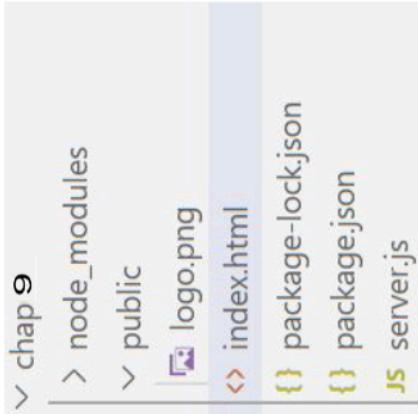
Renderizar archivos estáticos

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
  
</body>
</html>
```

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



Renderizar archivos estáticos

Agrega a server.js:

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

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

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

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

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

Ejemplo de CRUD

```
js server.js { data.json x
chap9-express-introduction > part3 > data > { data.json > ...
1 [ 2 {
3   "id":1,
4     "name":"car",
5     "price":50,
6     "description":"speed car with controller",
7     "picture":"car.png"
8   },
9   {
10    "id":2,
11    "name":"bear",
12    "price":70,
13    "description":"beautiful brown bear",
14    "picture":"bear.png"
15  }
16 ]
```

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);
```

localhost:3000/api/v1/toys

The screenshot shows the Wawiwa API testing interface. At the top, there's a navigation bar with Home, Workspaces, API Network, Reports, Explore, and a search bar for 'Search Postman'. Below the navigation is a toolbar with 'Import', 'New', 'Overwrite', 'Delete', 'Create', 'Import', 'Run', 'Save', 'Upgrade', and a dropdown for 'Environment' set to 'No Environment'.

The main area displays a collection named 'test-now' with a single request:

- Method:** GET
- URL:** localhost:3000/api/v1/toys
- Headers:** (empty)
- Authorization:** (empty)
- Params:** (empty)
- Query Params:** (empty)
- Body:** (empty)
- Pre-request Script:** (empty)
- Tests:** (empty)
- Settings:** (empty)

Below the request, there's a table for 'Value' and 'Key' pairs. The table has two rows:

KEY	Value
Key	Value

On the right side, there's a 'Send' button and a 'Save' button. Below the table, there are buttons for 'DESCRIPTION' and 'Description'.

At the bottom, there's a 'Bulk Edit' section with '200' and '300' buttons. To the right of the bulk edit, there are 'Send Response' and 'Save Response' buttons, along with a status indicator '200 OK 12 ms 453 B'.

A sidebar on the left lists collections: 'Media Authentication API', 'User Authentication', 'User Access Token', 'User's Page', 'Instagram Account', 'New Request', 'Sports team features', and 'Sports API'. A note says 'Create a collection for your requests'.

The bottom right corner shows a footer with 'Find and Replace', 'Console', 'Bootstrap', 'Runner', 'Trash', and a help icon.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

```
[{"id": 1, "name": "car", "price": 90, "description": "speed car with controller", "picture": "car.png"}, {"id": 2, "name": "bear", "price": 70, "description": "beautiful brown bear", "picture": "bear.png"}]
```

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

localhost:3000/api/v1/toys

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

raw -> body

localhost:3000/api/v1/toys

POST /api/v1/toys

Params: none

Headers: (8)

Body (raw) `{ "name": "test car", "price": 500, "description": "very very fast speed car with controller"}`

Pre-request Script

Tests

Settings

Send

Cookies

Beauty

Status: 201 Created Time: 30 ms Size: 575 B

Save Response

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
  
```

res

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

localhost:3000/api/v1/toys/2

localhost:3000/api/v1/toys/2

GET	Params	Headers (8)	Body	Pre-request Script	Tests	Settings
✓	Query Params	Headers (8)	Body	Pre-request Script	Tests	Settings
KEY	Value	DES	DES	DES	DES	DES
Key	Value	Value	Value	Value	Value	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ↻

Status: 200 OK

```
1 "status": "success",
2 "data": {
3   "id": 2,
4   "name": "test car",
5   "price": 500,
6   "description": "very very test speed car with controller",
7   "picture": "car.png"
8 }
9 }
10 }
```

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

localhost:3000/api/v1/toys/200

GET	localhost:3000/api/v1/toys/200	Save	Send
Params	Authorization	Headers (8)	Body ●
Query Params		Pre-request Script	Tests
		Settings	

404

Id 200 no existe

Respuesta

KEY	Value	DESCRIPTION	... Description
Key	Value	Description	...

Body

404

Status: 404 Not Found Time: 4 ms Size: 284 B Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2   "status": "fail",
3   "message": "id not exist"
4 }
```

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(e1 => e1.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

localhost:3000/api/v1/toys/1

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

raw -> body

The screenshot shows a POSTMAN interface with the following details:

- URL:** `localhost:3000/api/v1/toys/1`
- Method:** PATCH
- Body (JSON):**

```
1 {  
2   "name": "update test car",  
3   "price": 500000,  
4   "description": "woooooo",  
5   "picture": "car100.png"  
6 }
```
- Headers:** Body (8), Headers (8), Pre-request Script, Tests, Settings, JSON (selected), GraphQL, Binary, Cookies, Beautify.
- Response:**

```
1 {  
2   "status": "success",  
3   "data": {  
4     "id": 1,  
5     "name": "update test car",  
6     "price": 500000,  
7     "description": "woooooo",  
8     "picture": "car100.png"  
9   }  
10 }
```
- Metrics:** 200 OK, 24 ms, 358 B, Save Response.

res

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": "every very test speed car with controller", "picture": "car.png" }  
]
```

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.findIndex(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.writeFileSync("./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

localhost:3000/api/v1/toys/1

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

The screenshot shows the Postman interface with the following details:

- URL:** localhost:3000/api/v1/toys/1
- Method:** DELETE
- Headers:** Authorization (Bearer token)
- Body:** (Empty)
- Test Results:** (Empty)
- Preview:** Shows a JSON response with status "success delete".
- Code:** (Copied to clipboard)

```
1
2   "status": "success delete",
3
4   "data": [
5     {
6       "id": 1,
7       "name": "update test car",
8       "price": 500000,
9       "description": "wopowow",
10      "picture": "car100.png"
11    }
12  ]
```

“Toy” borrado

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"  
  }]
```

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

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.

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);
```

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);
}
```

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);
```

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);
```

Refactorizando el código parte deleteToyById

```
const deleteToyById = (req, res) => {
  const id = Number(req.params.id);
  const toy = toys.find(e1 => e1.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);
```

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);
```

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?