



Express



Mohamed Mukhtar Ahmed

1



Express



Outline

Issues with HTTP

Why Web Frameworks?

Express

Opinionated v/s Unopinionated

Data-Driven Web Apps

First Express App

Route Handlers

Serving Static Files

Processing JSON Data

Using Middleware

Postman

Why do we need it?

Testing HTTP Methods

Mohamed Mukhtar Ahmed

2

Express Web Framework



Mohamed Mukthar Ahmed

3

Issues with http



- Not a **GOOD IDEA** to have everything (data and functionality) in one single Node.js file
- We don't provide any info
 - We did not provide the data about what we are sending
 - We did not provide meta-data what we are sending
 - MIME type or Media type
- All the above have to be EXPLICITLY specified
- What will happen when we append the URL?

Mohamed Mukthar Ahmed

4

Why Web Frameworks?



- Common web-development tasks are not directly supported by Node.js itself.
- If you want to
 - Add specific handling for different HTTP methods (e.g. GET, POST, DELETE, etc.).
 - Separately handle requests at different URL paths ("routes")
 - Serve static files or
 - Use templates to dynamically create the response etc.
- Node.js won't be of much use on its own.
- You will either need to write the code yourself, or you can **avoid reinventing the wheel** and use a web framework!

Mohamed Mukthar Ahmed

5

Express



- Express is the **most popular** Node Web Framework.
- It is the **underlying library** for a number of other popular Node Web Frameworks.
- It provides mechanisms to:
 - Write handlers for requests with different HTTP verbs at different URL paths (routes).
 - Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
 - Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
 - Add additional request processing "middleware" at any point within the request handling pipeline.

Mohamed Mukthar Ahmed

6

Express



- While Express itself is fairly **minimalist**.
- Developers have created compatible **middleware packages** to address almost any web development problem.
- There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and many more.
- You can find a list of middleware packages maintained by the Express team at Express Middleware (along with a list of some popular 3rd party packages).

<https://expressjs.com/en/resources/middleware.html>

■

Mohamed Mukthar Ahmed

7

Opinionated vs Unopinionated



- Web frameworks often refer to themselves as **opinionated** or **unopinionated**
- **Opinionated** Web Framework
 - Those with opinions about the "**right way**" to handle any particular task.
 - They often support **RAD** in a particular domain (solving problems of a particular type) because the right way to do anything is usually well-understood and well-documented.
 - However they can be **less flexible** at solving problems outside their main domain, and tend to offer fewer choices for what components and approaches they can use.
 - Django, Spring Boot etc.

Mohamed Mukthar Ahmed

8

Opinionated vs Unopinionated

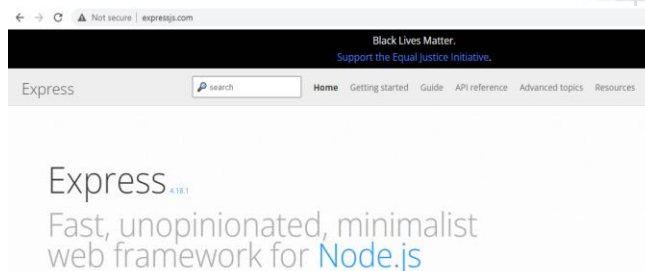
- Web frameworks often refer to themselves as **opinionated** or **unopinionated**
- **Unopinionated** Web Framework
 - Have far **fewer restrictions on the best way** to glue components together to achieve a goal, or even what components should be used.
 - They **make it easier for developers to use the most suitable tools** to complete a particular task, albeit at the cost that you need to find those components yourself.
 - Express.js

Mohamed Mukthar Ahmed

9

Express is Opinionated

- You can insert almost any compatible middleware you like into the request handling chain, in almost any order you like.
- You can structure the app in one file or multiple files, and using any directory structure.
- You may sometimes feel that you have too many choices!



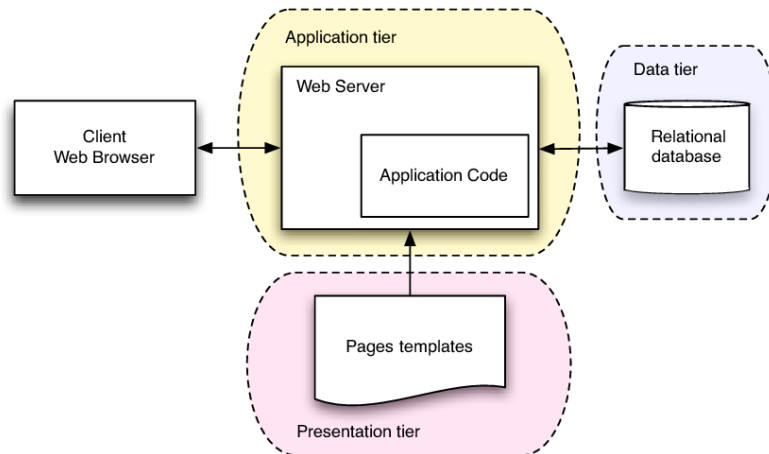
Mohamed Mukthar Ahmed

10

Data Driven Web App



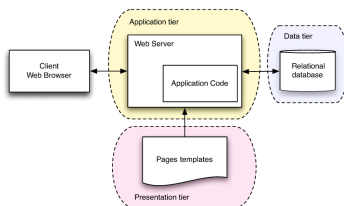
- A traditional Data Driven Web App architecture will look as shown below:



Mohamed Mukthar Ahmed

11

Data Driven Web App



- Web app waits for an HTTP request from it's client
- App works out what action is required based on URL pattern and HTTP method
 - May read or write info from Database
 - Perform other tasks
- Returns response to web browser, often dynamically creating HTML page

Mohamed Mukthar Ahmed

12

Data Driven Web App



- Express provides methods to specify
 - What function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route")
- Methods to specify
 - what template ("view") engine is used,
 - where template files are located, and
 - what template to use to render a response
- Can use Express middleware to add support for getting POST/GET parameters, sessions, and users, cookies, etc.
- Can use any database mechanism supported by Node.js (Express does not define any database-related behavior).

Mohamed Mukthar Ahmed

13

First Express App



- Our first Express app

```

1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.get('/', function(req, res) {
6    res.send('Hello World!')
7  });
8
9  app.listen(port, function() {
10   console.log(`Example app listening on port ${port}!`)
11 });
  
```

Mohamed Mukthar Ahmed

14

Route Handlers



- In the example, we defined a (callback) route handler function for **HTTP** GET requests to the site root ('/').
- The Express application object also provides methods to define route handlers for all the other HTTP verbs, which are mostly used in exactly the same way.
- Moreover, we can also send '**Status Code**' by using the **status()** method and chaining it with **send()** .
- There are a number of other response methods for ending the request/response cycle, for example,
 - You could call **res.sendFile()** to send a file or
 - **res.json()** to send a JSON response.

Mohamed Mukthar Ahmed

15

Route Handlers



- In the example, we defined a (callback) route handler function for HTTP GET requests to the site root ('/').
- There is a special routing method, **app.all()**, which will be called in response to any HTTP method.
 - This is used for loading **middleware** functions at a particular path for all request methods.

Mohamed Mukthar Ahmed

16

Code Example



```

File Edit Selection View Go Run ... exp_ex2.js - Express-Demos - Visual St...
JS exp_ex2.js x
First > JS exp_ex2.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.get('/', function(req, res) {
6    res.send('<h1>Home Page!</h1>');
7  });
8
9  app.get('/about', (req, res) => {
10    res.send('<h1>About!</h1>');
11  });
12
13 app.all('*', (req, res) => {
14   res.status(404).send(`
15     <h1>Oops!</h1>
16     <p>Page NOT Found...</p>
17     <a href="/">Home</a>`)
18   });
19 app.listen(port, function() {
20   console.log(`Express app listening on port ${port}!`)
21 });

```

Mohamed Mukthar Ahmed

17

Serving Static Files



- You can use the **express.static()** **middleware** to serve static files.
 - Including your images, CSS and JavaScript etc.
- **NOTE:** **static()** is the only **middleware function** that is actually part of Express.
- Example:
 - You would use the line below to serve images, CSS files, and JavaScript files from a directory named **'public'**

```
app.use(express.static('./public'));
```
- Any files in the **public** directory are served by adding their filename (relative to the base "public" directory) to the base URL.

Mohamed Mukthar Ahmed

18

Code Example



```

app.js
1  const express = require('express');
2  const path = require('path');
3
4  const app = express();
5  const port = 3000;
6
7  app.get('/', function(req, res) {
8    res.sendFile(path.resolve(__dirname, './future_value/index.html'))
9  });
10
11
12 app.listen(port, function() {
13   console.log(`Express app listening on port ${port}!`)
14 });
15

```

```

[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Express app listening on port 3000!

```

Mohamed Mukthar Ahmed

19

Serving Static Files



- Now create a sub-directory by name 'public' and move or copy all the files of future_value in the public directory
- Add the below line


```
app.use(express.static('./public'));
```
- Observer the UI the functionality.
 - Also check in the network if any links are missing.

Mohamed Mukthar Ahmed

20

Processing JSON Data



- Express provides with the **res.json()** to work with JSON data.
- If we pass the JSON file it's contents will be displayed

```

1  const express = require('express');
2  const app = express();
3  const port = 3000;
4
5  app.get('/', function(req, res) {
6    res.json([
7      {name: 'Amar', age:45},
8      {name: 'Akbar', age: 32},
9      {name: 'Antony', age:42}
10   ])
11 });
12
13 app.listen(port, function() {
14   console.log(`Express app listening on port ${port}!`)
15 });

```

Mohamed Mukthar Ahmed

21

Using Middleware



- Middleware is used extensively in Express apps, for tasks
 - from serving static files,
 - to error handling,
 - to building business logic,
 - to compressing HTTP responses etc.
- Middleware functions typically perform some operation on the request or response and then call the **next** function in the "stack", which might be more middleware or a route handler.
- The order in which middleware is called is up to the app developer.

Mohamed Mukthar Ahmed

22

Using Middleware

- The middleware can perform any operation, execute any code, make changes to the request and response object, and it can also end the request-response cycle.
- If it does not end the cycle, then it must call **next()** to pass control to the next middleware function (or the request will be left hanging).
- The only difference between a middleware function and a route handler callback is that middleware functions have a **third argument next**, which middleware functions are expected to call if they are not that which completes the request cycle (when the middleware function is called, this contains the next function that must be called).
- You can add a middleware function to the processing chain for all responses with **app.use()**.

Mohamed Mukthar Ahmed

23

Code Example

```

app.js
First > JS app.js > app.all(*) callback
5 // req => middleware => res
6
7 const sysDate = function (req, res, next) {
8   const method = req.method;
9   const url = req.url;
10  const today = new Date();
11  console.log(method, url, today);
12  next();
13 }
14
15 app.get('/', sysDate, function(req, res) {
16   res.send('<h1>Home Page!</h1>');
17 });
18
19 app.get('/about', sysDate, (req, res) => {
20   res.send('<h1>About!</h1>');
21 })
22
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
GET / 2022-09-09T06:22:01.501Z
GET /about 2022-09-09T06:23:20.592Z
GET / 2022-09-09T06:23:43.396Z
GET /about 2022-09-09T06:23:56.014Z
GET / 2022-09-09T06:24:17.260Z
  
```

Home Page!

What if we have many middleware functions?

What if we want to use middleware in more routes?

Mohamed Mukthar Ahmed

24

Code Example



The screenshot shows a VS Code editor with a file named `app.js`. The code defines an Express application that listens on port 3000 and has a GET endpoint `/api/people` that returns a JSON array of people. A terminal window at the bottom shows the command `node app.js` being executed, and the output indicates the server is listening on port 3000. To the right, a web browser at `localhost:3000` displays the JSON response from the `/api/people` endpoint.

```

app.js
1 const express = require('express');
2 const app = express();
3 const port = 3000;
4 let {people} = require('./data');
5
6 app.get('/api/people', (req, res) => {
7   res.status(200).json({success:true, data:people});
8 })
9
10 app.listen(port, function() {
11   console.log(`Express app listening on port ${port}!`)
12 });
13

[Terminal]
[nodemon] starting 'node app.js'
Express app listening on port 3000!
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Express app listening on port 3000!

[Browser]
{
  "success": true,
  "data": [
    {
      "id": 1,
      "name": "Amar"
    },
    {
      "id": 2,
      "name": "Akbar"
    },
    {
      "id": 3,
      "name": "Antony"
    },
    {
      "id": 4,
      "name": "Tom"
    },
    {
      "id": 5,
      "name": "Cherry"
    }
  ]
}

```

Mohamed Mukthar Ahmed

25

Code Example



The screenshot shows a VS Code editor with a file named `app.js`. The code defines an Express application that listens on port 3000. It has a GET endpoint `/api/people` and a POST endpoint `/login`. The `/login` endpoint checks if a name is provided and returns a welcome message or a 401 status. A terminal window at the bottom shows the command `node app.js` being executed, and the output indicates the server is listening on port 3000. To the right, a web browser at `localhost:3000` displays a 'Traditional Form' with a text input field labeled 'Enter Name' containing the value 'Mukthar' and a 'Submit' button.

```

app.js
6 app.use(express.static('methods_public'));
7 // parse form data
8 app.use(express.urlencoded({extended:false}));
9
10 app.get('/api/people', (req, res) => {
11   res.status(200).json({success:true, data:people});
12 })
13
14 app.post('/login', (req, res) => {
15   const {name} = req.body;
16   if (name) {
17     return res.status(200).send(`<h1>Welcome ${name}</h1>`);
18   }
19   res.status(401).send('Please provide credentials...');
20 })
21
22 app.listen(port, function() {
23   console.log(`Express app listening on port ${port}!`)
24 });
25

[Terminal]
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Express app listening on port 3000!
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Express app listening on port 3000!

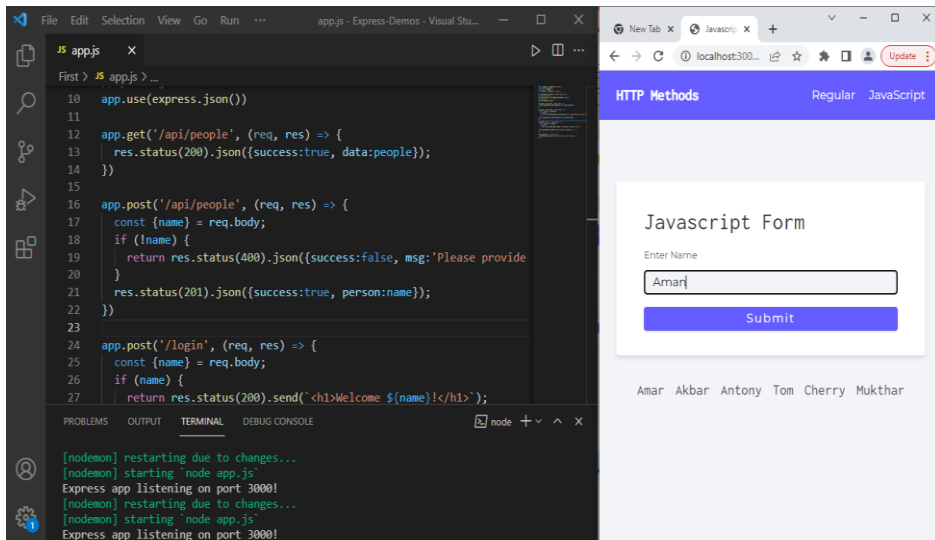
[Browser]
Traditional Form
Enter Name
Mukthar
Submit

```

Mohamed Mukthar Ahmed

26

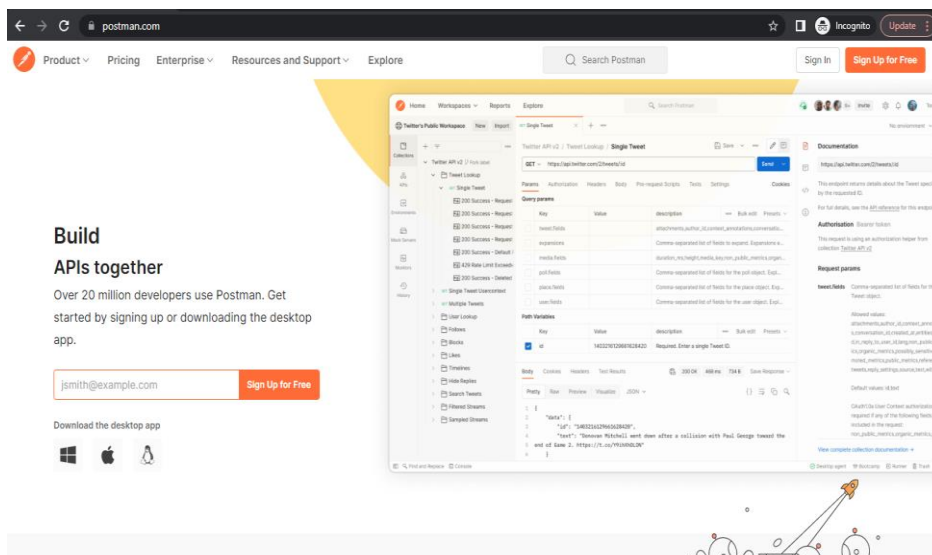
Code Example



Mohamed Mukthar Ahmed

27

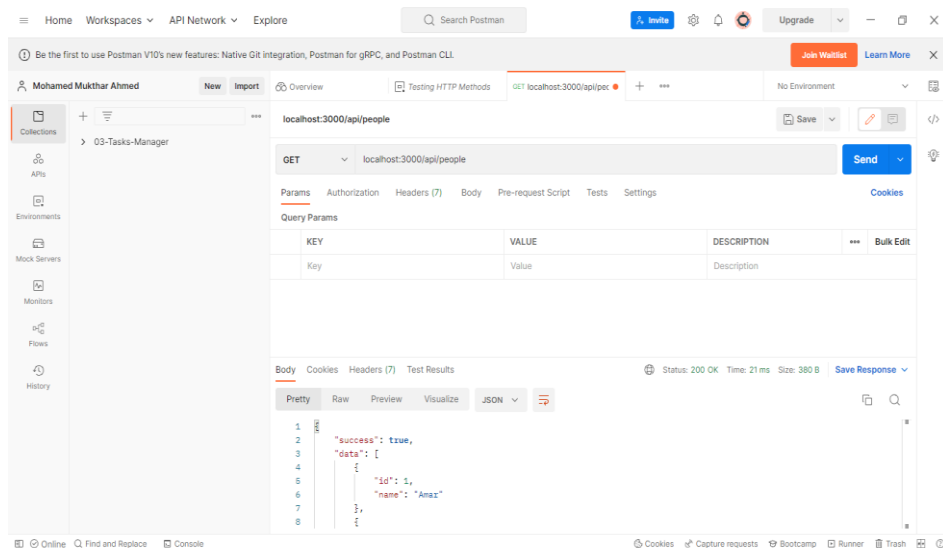
Postman



Mohamed Mukthar Ahmed

28

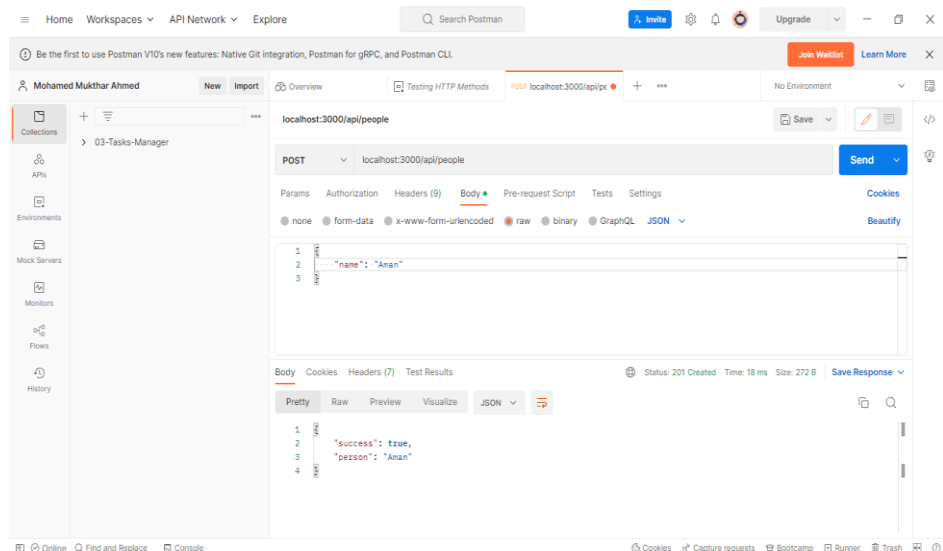
Postman – GET Method



Mohamed Mukhtar Ahmed

29

Postman – POST Method



Mohamed Mukhtar Ahmed

30



Code Example



```

app.js
First > JS app.js > app.post('/api/postman/people') callback
16 app.post('/api/people', (req, res) => {
17   const {name} = req.body;
18   if (!name) {
19     return res.status(400).json({success:false, msg:'Please provide
20   }
21   res.status(201).json({success:true, person:name});
22 })
23 // Another route for testing with Postman
24 app.post('/api/postman/people', (req, res) => {
25   const {name} = req.body;
26   if (!name) {
27     return res
28       .status(400)
29       .json({success:false, msg:'Please provide the name...'});
30   }
31   res.status(201).json({success:true, data: [...people,name]});
32 })
33
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[nodemon] starting 'node app.js'
Express app listening on port 3000!
[nodemon] restarting due to changes...
[nodemon] starting 'node app.js'
Express app listening on port 3000!
[nodemon] restarting due to changes...
Ln 31, Col 64 Spaces: 2 UTF-8 CRLF JavaScript Go Live

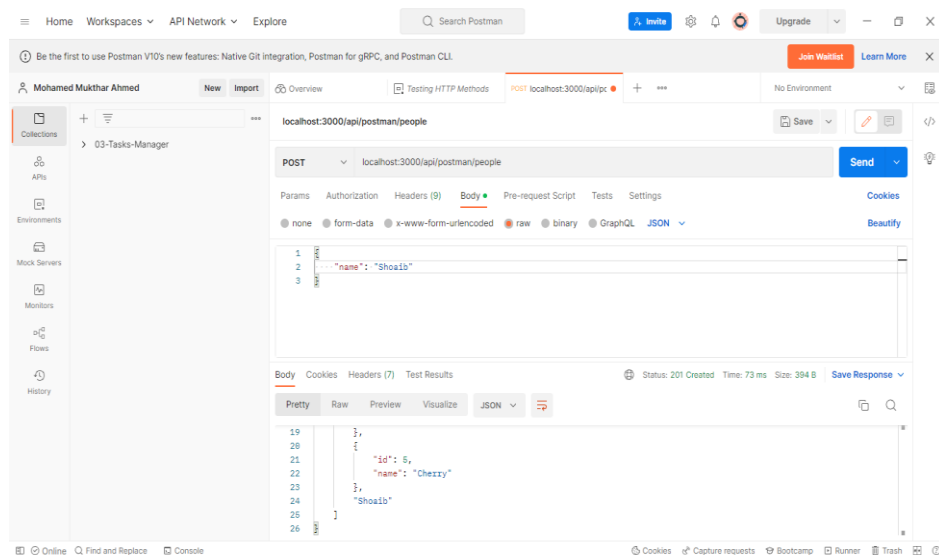
```

Mohamed Mukhtar Ahmed

31



Postman – POST Method



Mohamed Mukhtar Ahmed

32



Code Example



```

app.js
First > JS app.js > app.put('/api/people/id') callback
43 app.put('/api/people/:id', (req, res) => {
44   const {id} = req.params;
45   const {name} = req.body;
46   const person = people.find((person) => person.id === Number(id));
47
48   if (!person) {
49     return res.status(404)
50       .json({success:false, msg:`No person with ID ${id}...`});
51   }
52
53   const newPeople = people.map( (person) => {
54     if (person.id === Number(id)) {
55       person.name = name
56     }
57     return person
58   })
59
60   res.status(200).send({success:true, data: newPeople});
61

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```

[nodemon] starting `node app.js`
Express app listening on port 3000!
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Express app listening on port 3000!

```

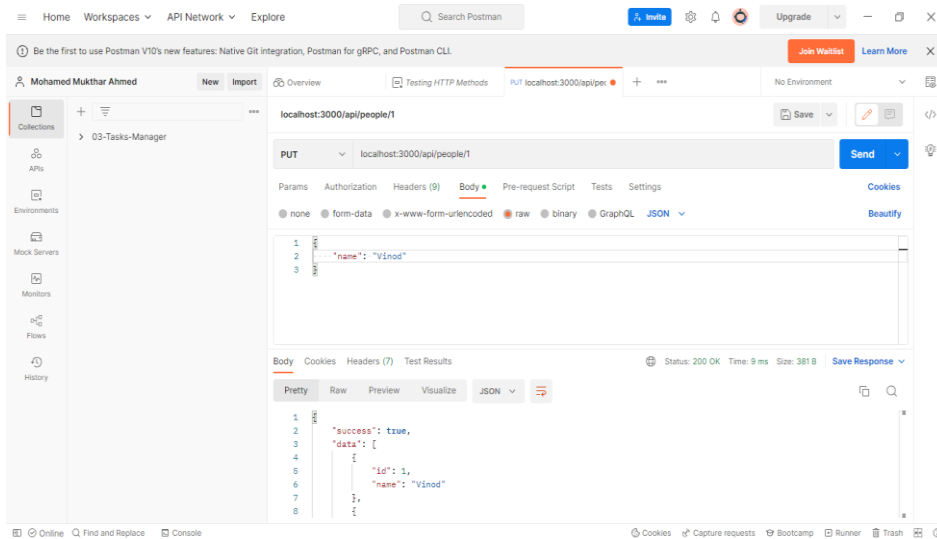
Ln 49, Col 15 Spaces: 2 UTF-8 CRLF JavaScript Go Live

Mohamed Mukhtar Ahmed

33



Postman – PUT Method



Mohamed Mukhtar Ahmed

34



Code Example



```

1  First > JS app.js > app.delete('/api/people/:id') callback
2  62 // Checking DELETE method
3  63 // Checking PUT method
4  64 app.delete('/api/people/:id', (req, res) => {
5  65   const {id} = req.params;
6  66   const person = people.find((person) => person.id === Number(id));
7  67
8  68   if (!person) {
9  69     return res.status(404)
10  70     .json({success:false, msg:`No person with ID ${id}...`});
11  71   }
12  72
13  73   const newPeople = people.filter(
14  74     (person) => person.id !== Number(id)
15  75   )
16  76
17  77   return res.status(200).send({success:true, data: newPeople});
18  78 }
19  79
20  80

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```

[nodemon] starting `node app.js`
Express app listening on port 3000!
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Express app listening on port 3000!

```

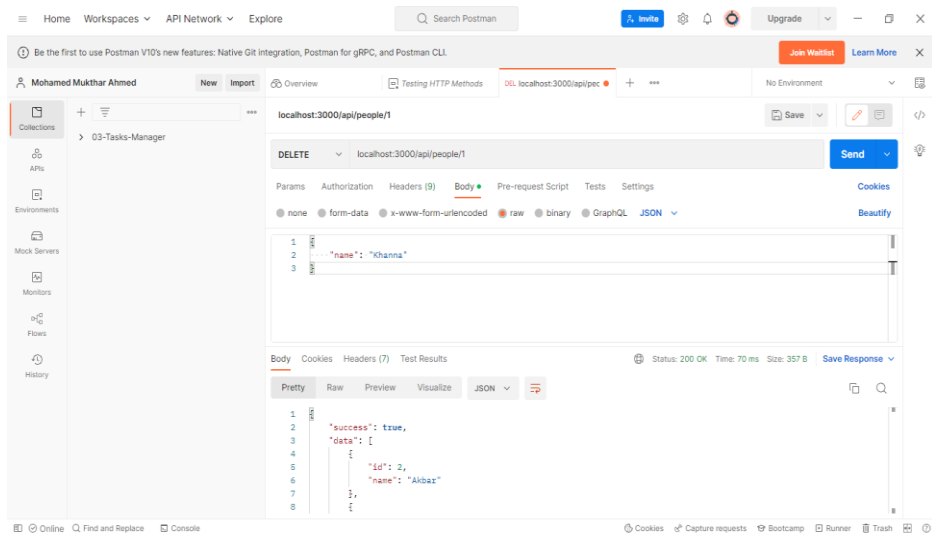
Ln 78, Col 1 Spaces: 2 UTF-8 CRLF JavaScript Go Live

Mohamed Mukhtar Ahmed

35



Postman – DELETE Method



Mohamed Mukhtar Ahmed

36