Express

Mohamed Mukthar 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 Mukthar Ahmed

2

Express Web Framework

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 EXPLICTILY specified
- What will happen when we append the URL?

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
-

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.

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

9

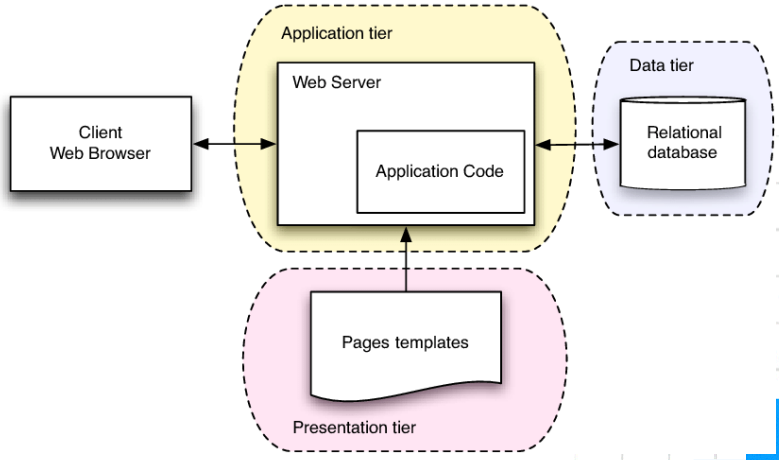
Express is Opinionated

- 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!

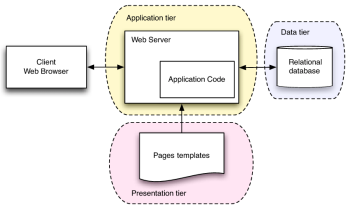
10

Data Driven Web App

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



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

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).

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 });
```


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.

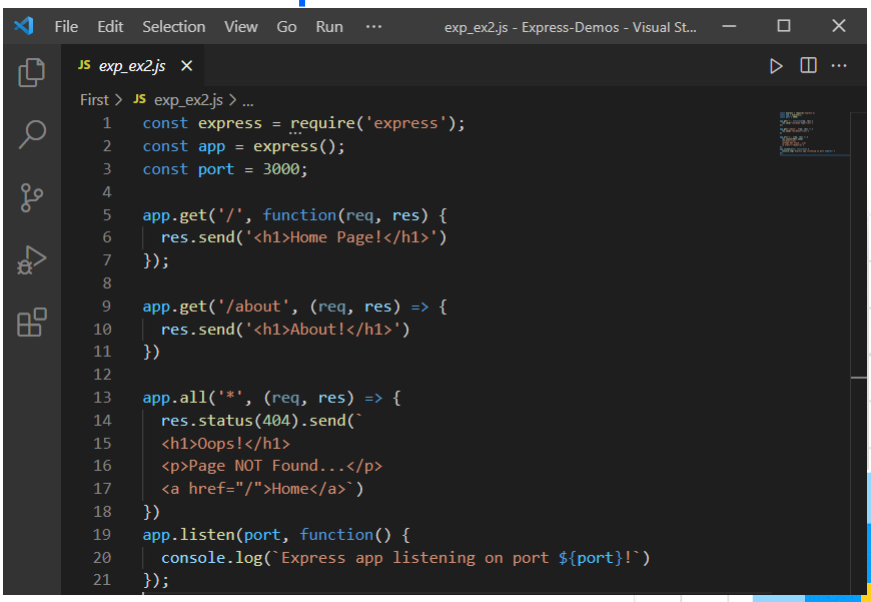
15

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.

16

Code Example



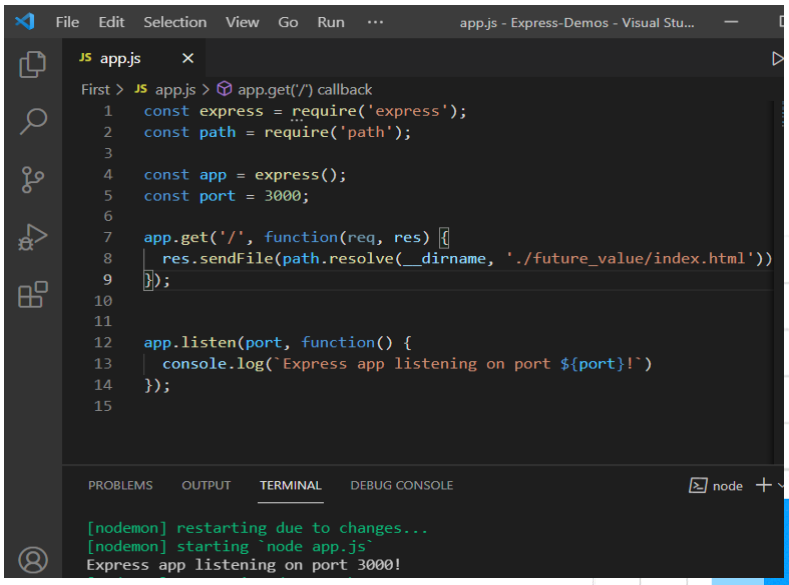
Serving Static Files

- 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.

Code Example



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.

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

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

app.get('/', function(req, res) {
  res.json([
    {name: 'Amar', age:45},
    {name: 'Akbar', age: 32},
    {name: 'Antony', age:42}
  ])
});

app.listen(port, function() {
  console.log(`Express app listening on port ${port}!`)
});
```


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.

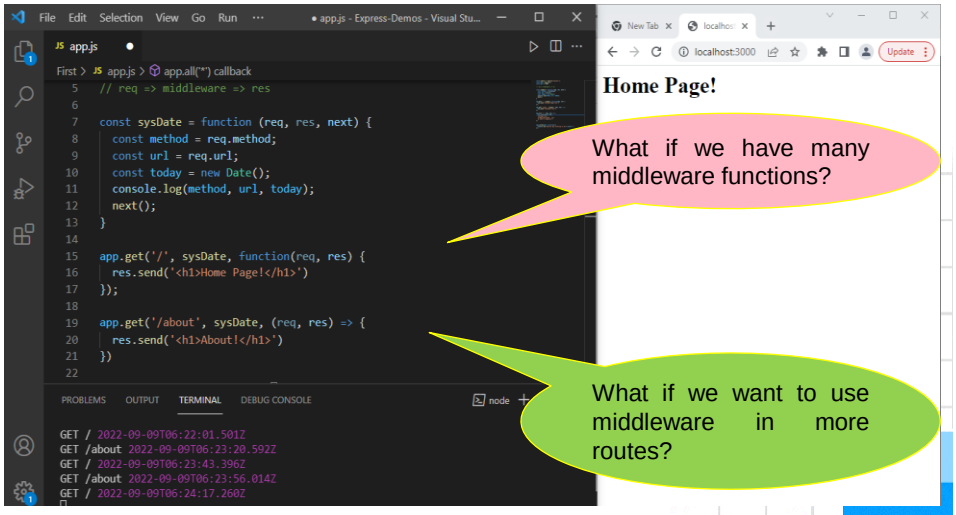
22

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()**.

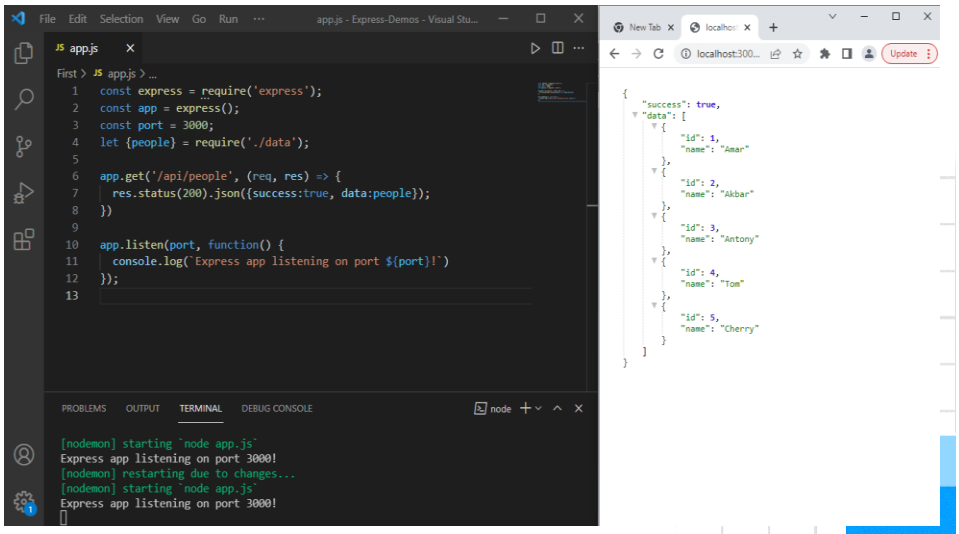
23

Code Example



24

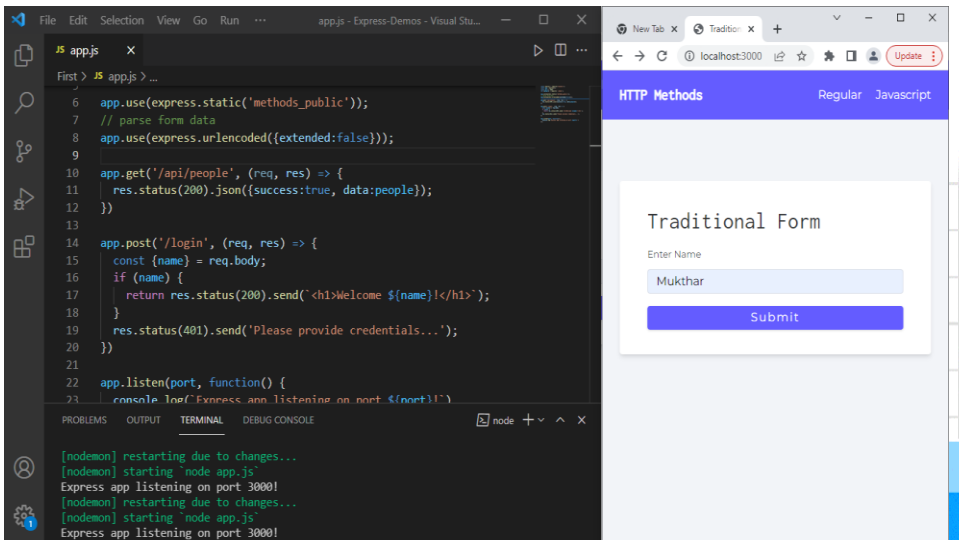
Code Example



Mohamed Mukthar Ahmed

25

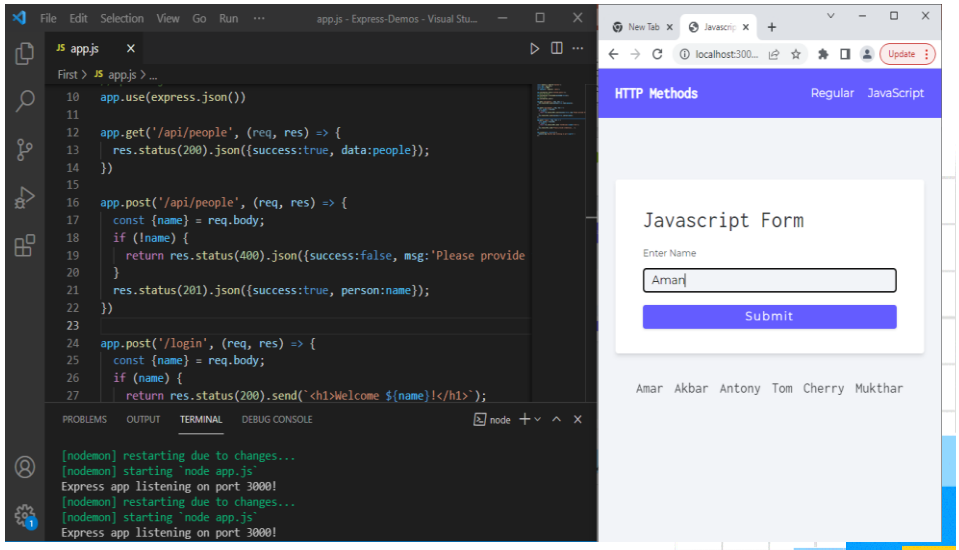
Code Example



Mohamed Mukthar Ahmed

26

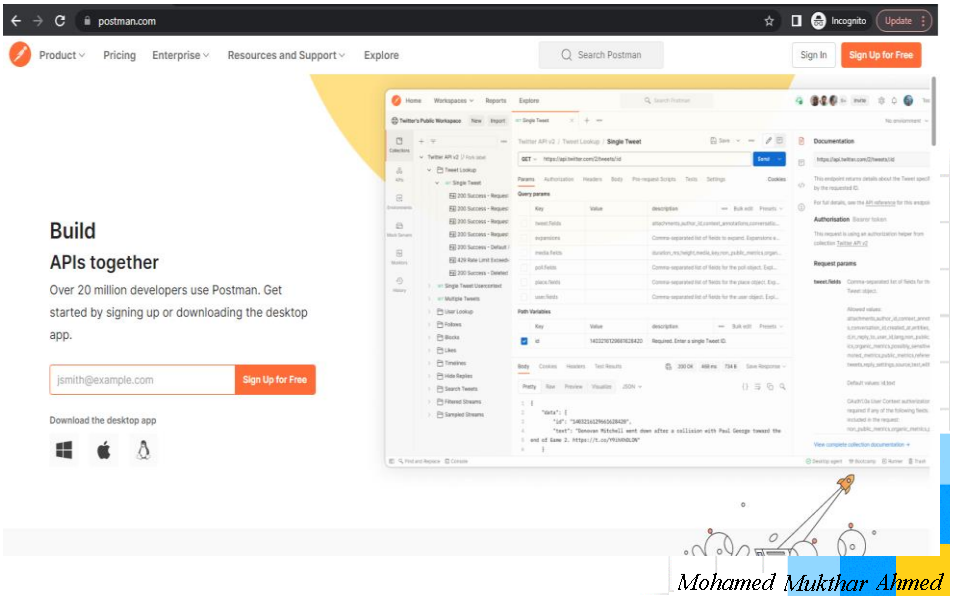
Code Example



Mohamed Mukthar Ahmed

27

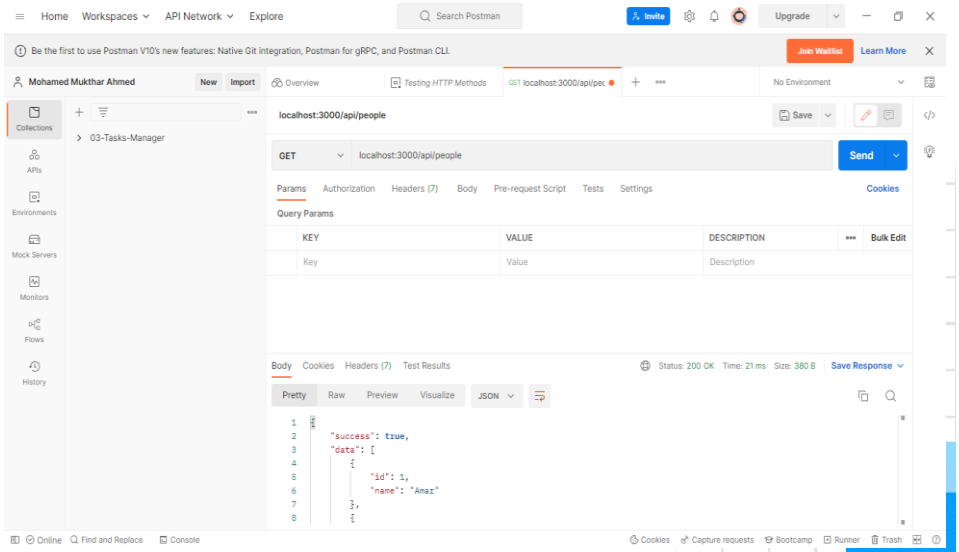
Postman



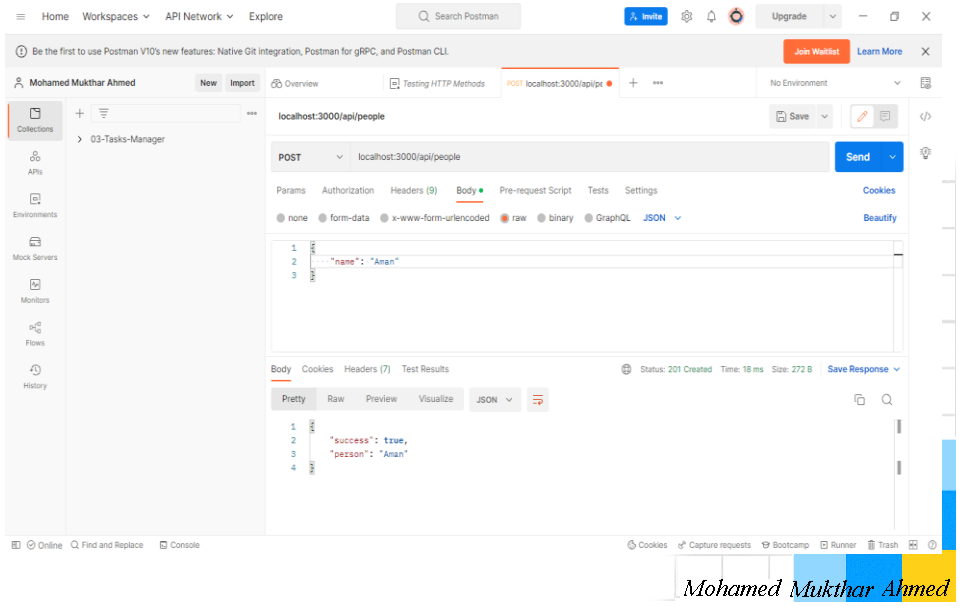
Mohamed Mukthar Ahmed

28

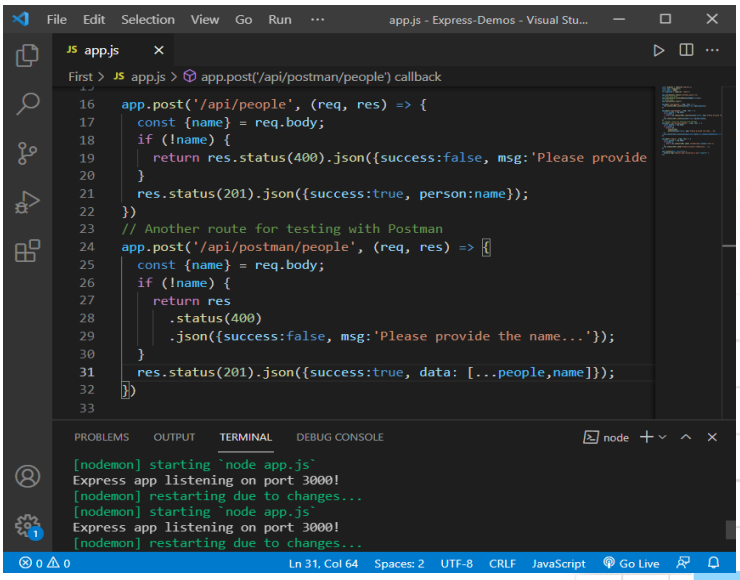
Postman – GET Method



Postman – POST Method



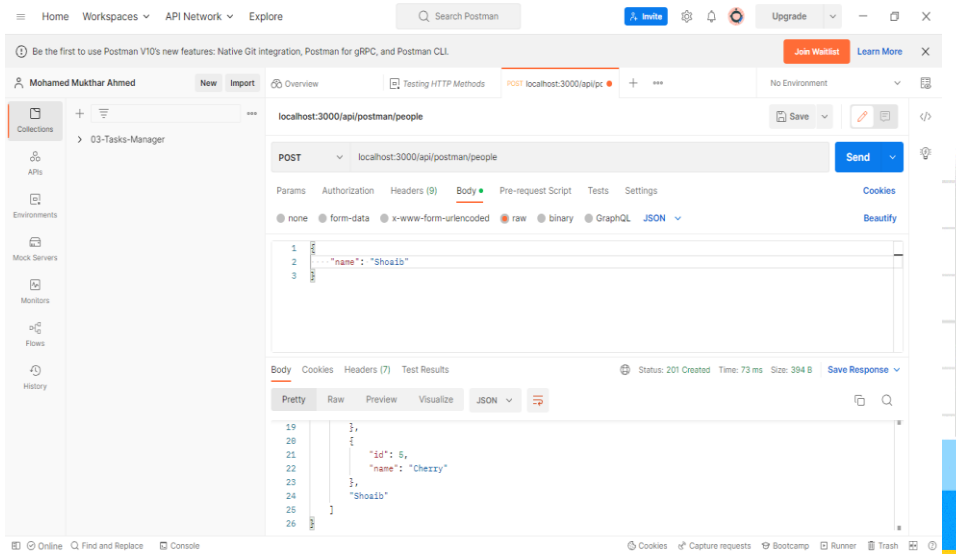
Code Example



Mohamed Mukthar Ahmed

31

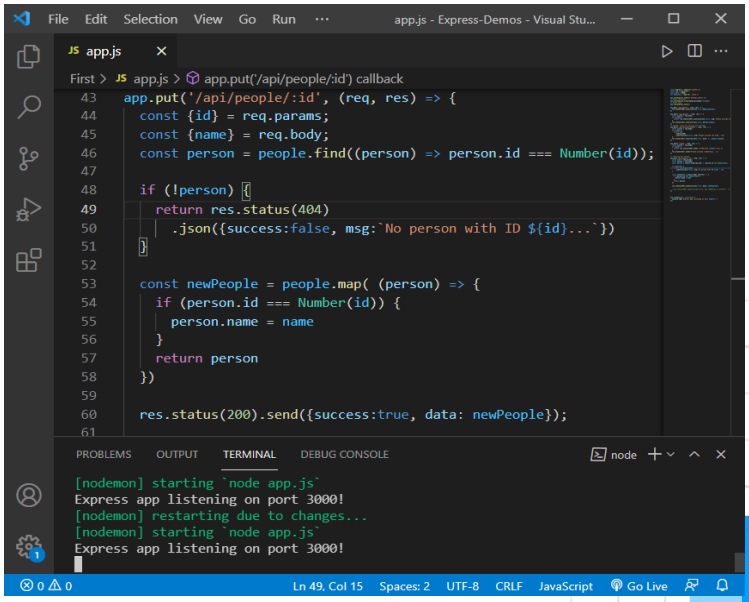
Postman – POST Method



Mohamed Mukthar Ahmed

32

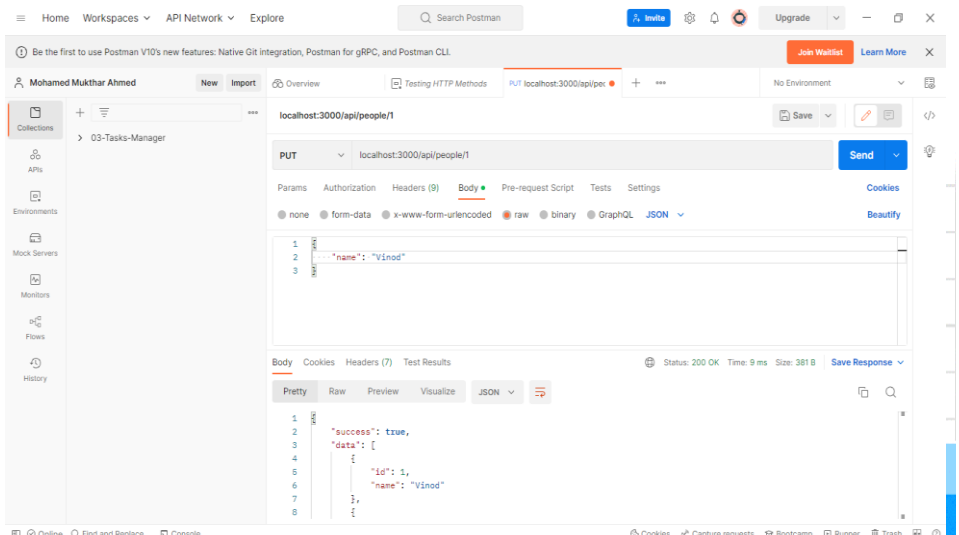
Code Example



Mohamed Mukthar Ahmed

33

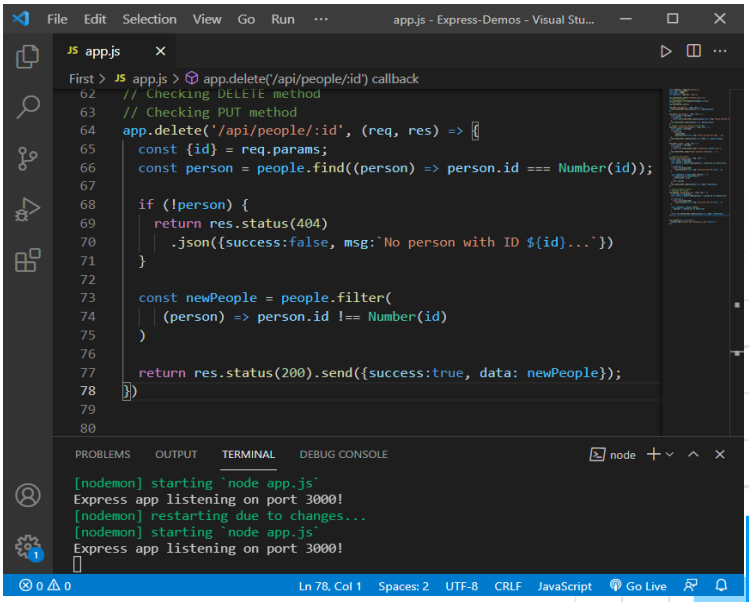
Postman – PUT Method



Mohamed Mukthar Ahmed

34

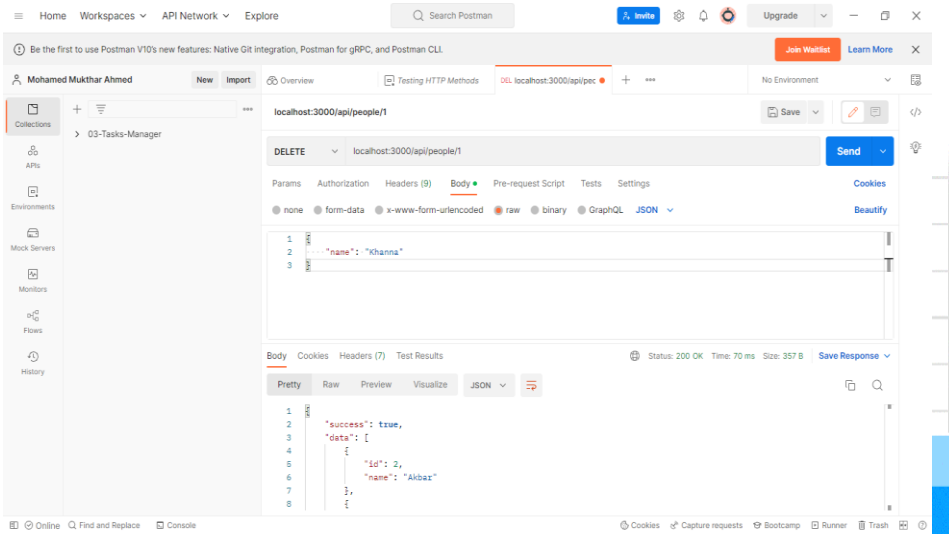
Code Example



Mohamed Mukthar Ahmed

35

Postman – DELETE Method



Mohamed Mukthar Ahmed

36