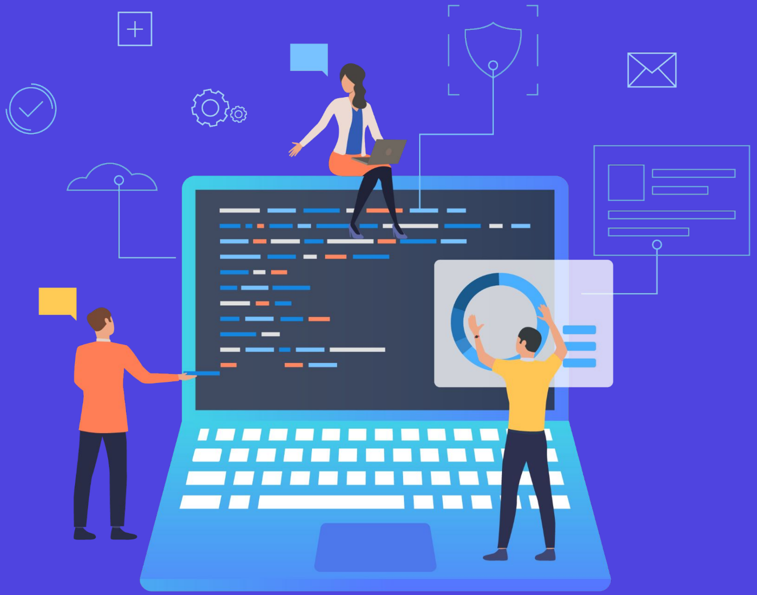
Introduction to Express and REST



Relevel
by Unacademy

Recap

1. How we created a new web server in Node.js
2. HTTP response codes
3. HTTP methods

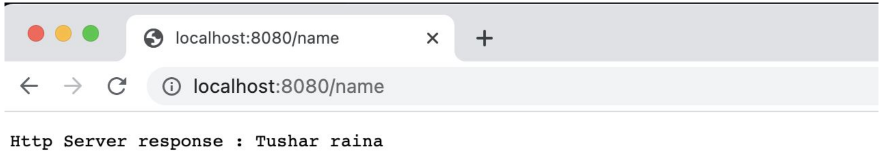
Routing Logic

1. Return Different response data based on different request URL’s

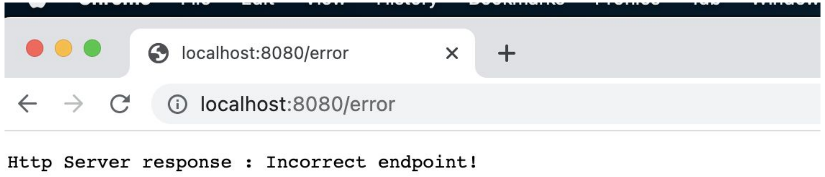
```
   {

      case "/name":
        res.end('Http Server response : Tushar Raina);
        break;
      case "/company":
        res.end('Http Server response : A reputed MNC');
        break;
      default:
        res.end('Http Server response : Incorrect
endpoint!');
  }
```

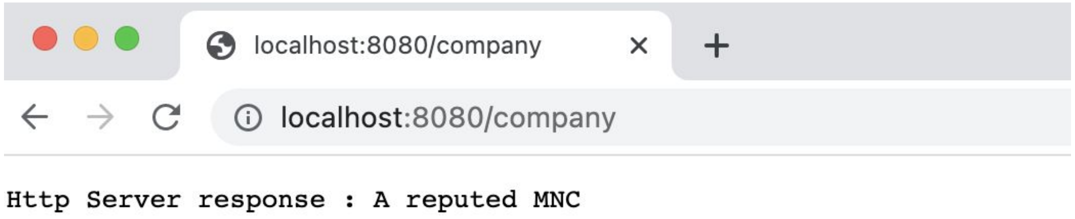
Routing in action



```
Http Server response : Tushar raina
```

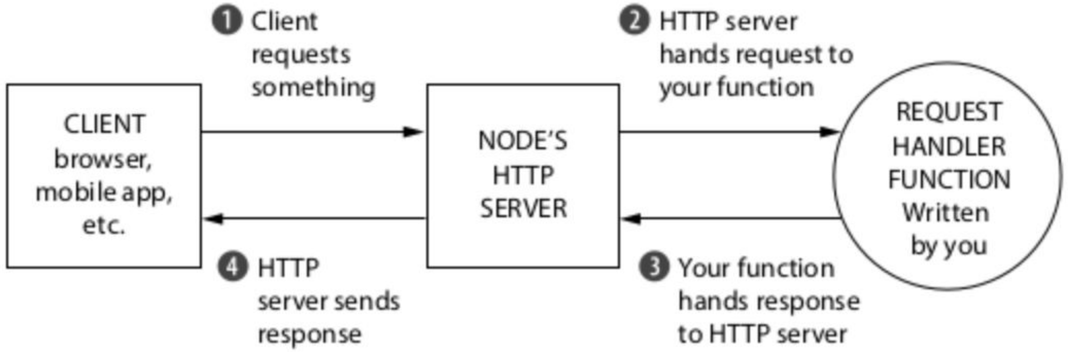


```
Http Server response : Incorrect endpoint!
```



```
Http Server response : A reputed MNC
```

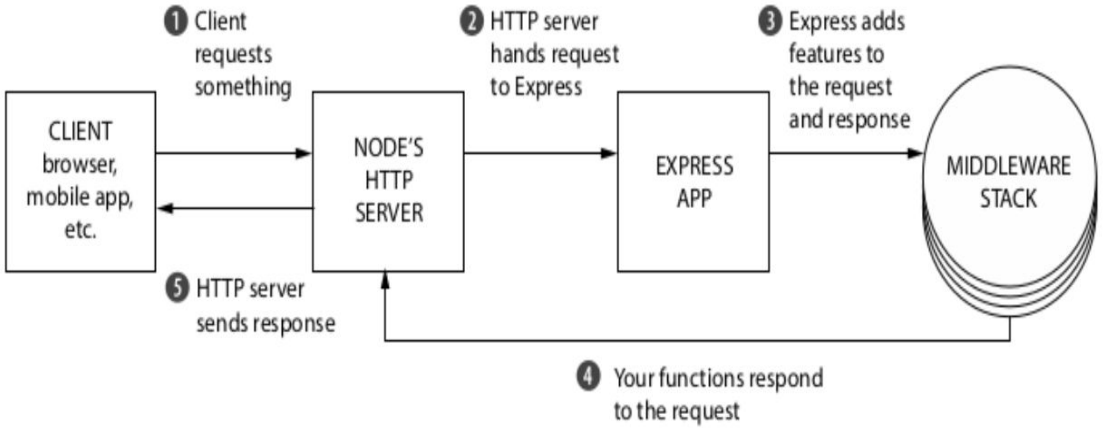
Noticed anything !!



1 Client requests something
2 HTTP server hands request to your function
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
REQUEST HANDLER FUNCTION Written by you
4 HTTP server sends response
3 Your function hands response to HTTP server

How we need to write redundant code which is not extensible to add new route everytime

Express.js to rescue



1 Client requests something
2 HTTP server hands request to Express
3 Express adds features to the request and response
CLIENT browser, mobile app, etc.
NODE'S HTTP SERVER
EXPRESS APP
MIDDLEWARE STACK
5 HTTP server sends response
4 Your functions respond to the request

Express.js Features

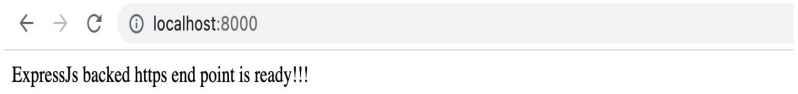
- Fast and simple server side development
- Access to middleware
- Routing
- Templates

Let us Write our first Express code!!!

```
var expressJs = require("express");
var expressApp = expressJs();

// Add routing logic
expressApp.get("/", function(request, response){
 response.send("ExpressJs backed https endpoint is ready!!!");
 });

//Ask app to listen on given a port
expressApp.listen(8000, function(){
 console.log("Application is listening to port 8000");
 });
```


localhost:8000

ExpressJs backed https end point is ready!!!

Understanding express

1. Import modules and create express app
2. expressApp.get() helps us respond to get requests.
3. expressApp.listen() helps us listen to a particular code

Let us now add multiple routes using Express and test it

```
expressApp.get("/", function(request, response){
   response.send("ExpressJs backed http endpoint is ready!!!");
 });

// Add routing logic
expressApp.get("/name", function(request, response){
   response.send("ExpressJs backed http endpoint is ready! Name : Tushar Raina");
   });

// Add routing logic
expressApp.get("/company", function(request, response){
   response.send("ExpressJs backed http endpoint is ready! Company : A reputed MNC");
   });
```

Routing Tricks : Getting Route Parameters

Let us build a system

- which has multiple students
- We need to get details of students based on their userId.
- Each student has its own userId

Approach 1

We build route path corresponding to each customer , something like this :

```
expressApp.get("/users/1", function(request, response){
   response.send("ExpressJs backed http endpoint is ready! Name : Tushar
Raina");
   });

expressApp.get("/users/2", function(request, response){
   response.send("ExpressJs backed http endpoint is ready! Name : Sachin
Tendulkar");
   });
```

Approach 2 (Preferred)

- We define a single route of the form /users/:userId .
- This ensured that for each request we have a different Id.
- Even if we have 1000’s of users still we have only one route function.

Let us implement both the approaches !!!

Routing Tricks: Using Regular Expressions

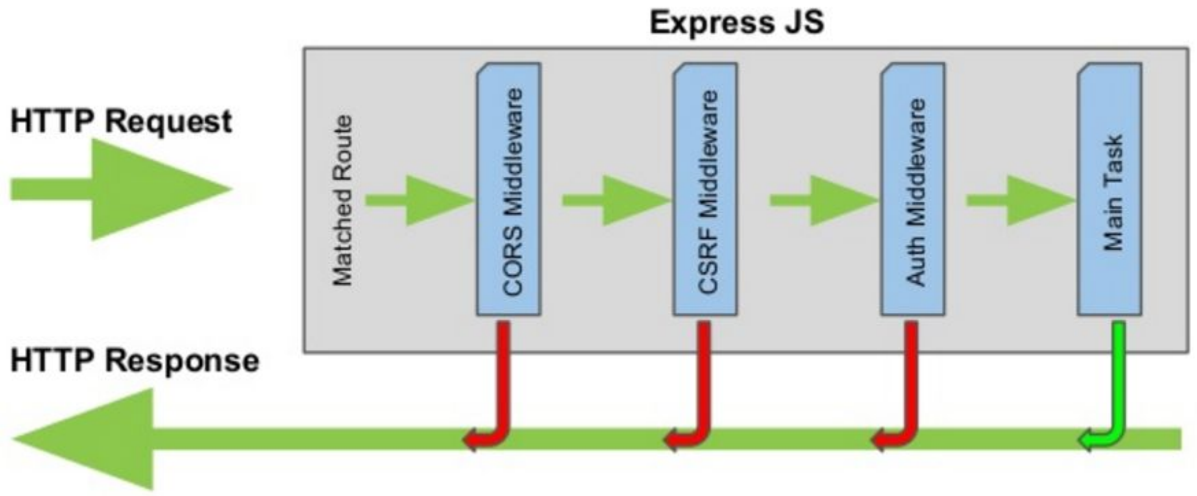
what happens if someone tries end point like

- /users/ads
- /users/12dfds
- /users/fgfgd%$%$%

We know user Id can only be integer , so we need a way to filter them out.

If we create an expression that filters out incorrect values, and filter it while routing , it solves out use case!

Express Middleware

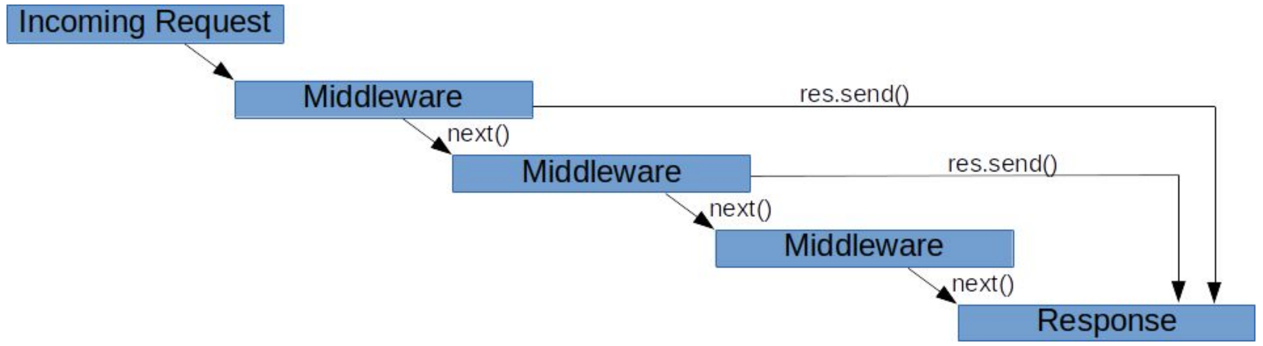


What can Middlewares can handle

1. Logging
2. Sending Static Files
3. Authentication and Authorisation
4. Session Management
5. Parsing
6. Rate Limiting

How do middlewares work?

1. Each middleware has access to request , response and next() method
2. Each of them can terminate the flow
3. Ultimately we get a graph like structure which keeps our code modular and simpler

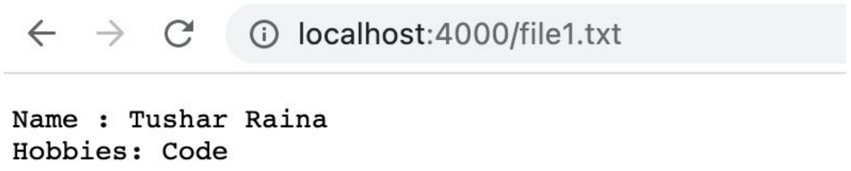


Common Express middlewares

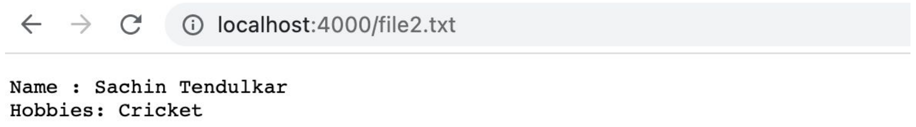
1. Morgan
2. Helmet
3. Body-Parser
4. Cors
5. Express rate Limit

Serving Static Files

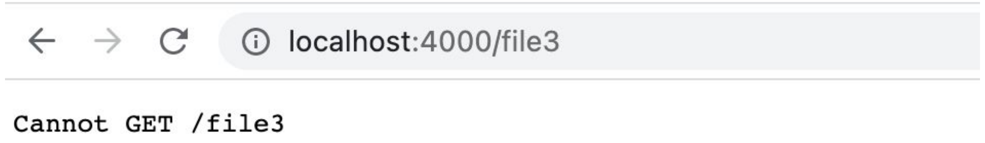
Let us try to build a service which returns the local static file if filename is passed in the url.


localhost:4000/file1.txt

```
Name : Tushar Raina
Hobbies: Code
```


localhost:4000/file2.txt

```
Name : Sachin Tendulkar
Hobbies: Cricket
```


localhost:4000/file3

```
Cannot GET /file3
```

Let us add simple middlewares !

We will add 2 logging components.

a. First one will log all incoming requests.
b. Second one will log only when there is an incorrect url passed

Let us look into the results and see how different middlewares actually run together!

REST API

Let us build a service which manages student information

1. We can add new student details
2. We can delete student details
3. We can view all student details

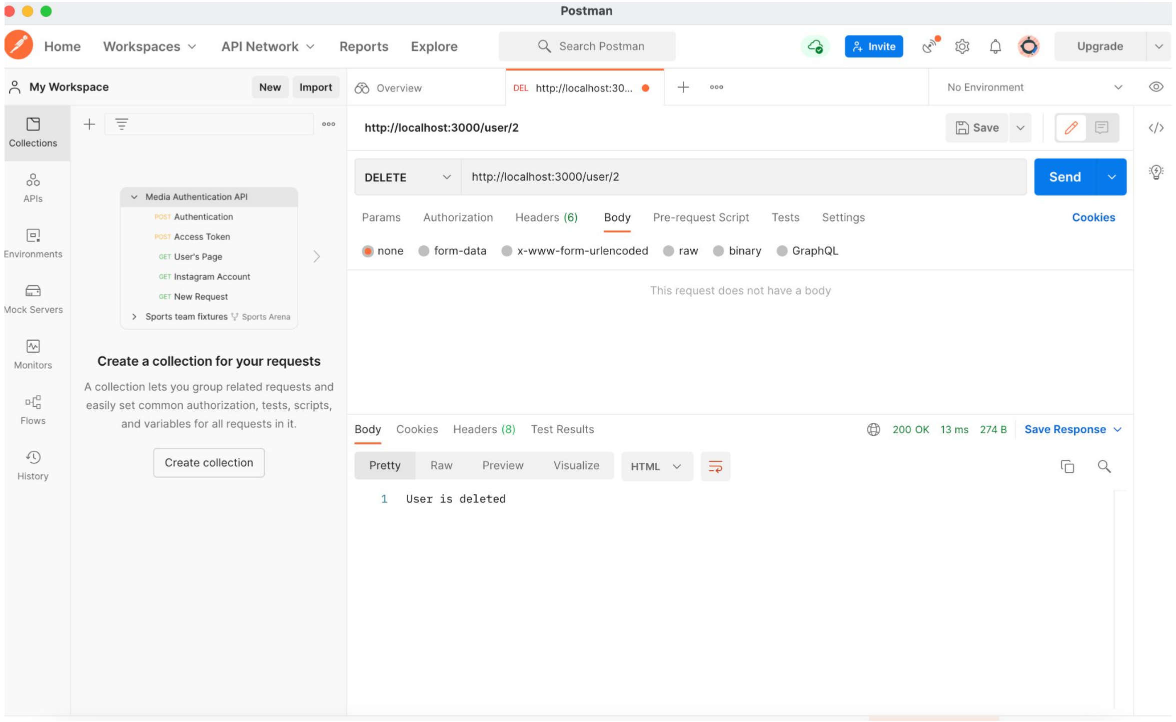
We can map each request to HTTP method type

GET-> users/ (Returns list of all users)

POST-> user/ (Adds the user based upon information in request body in our data store)

DELETE -> user/:id (Deleted the user based on id provided)

PostMan Setup and Testing : Testing DELETE



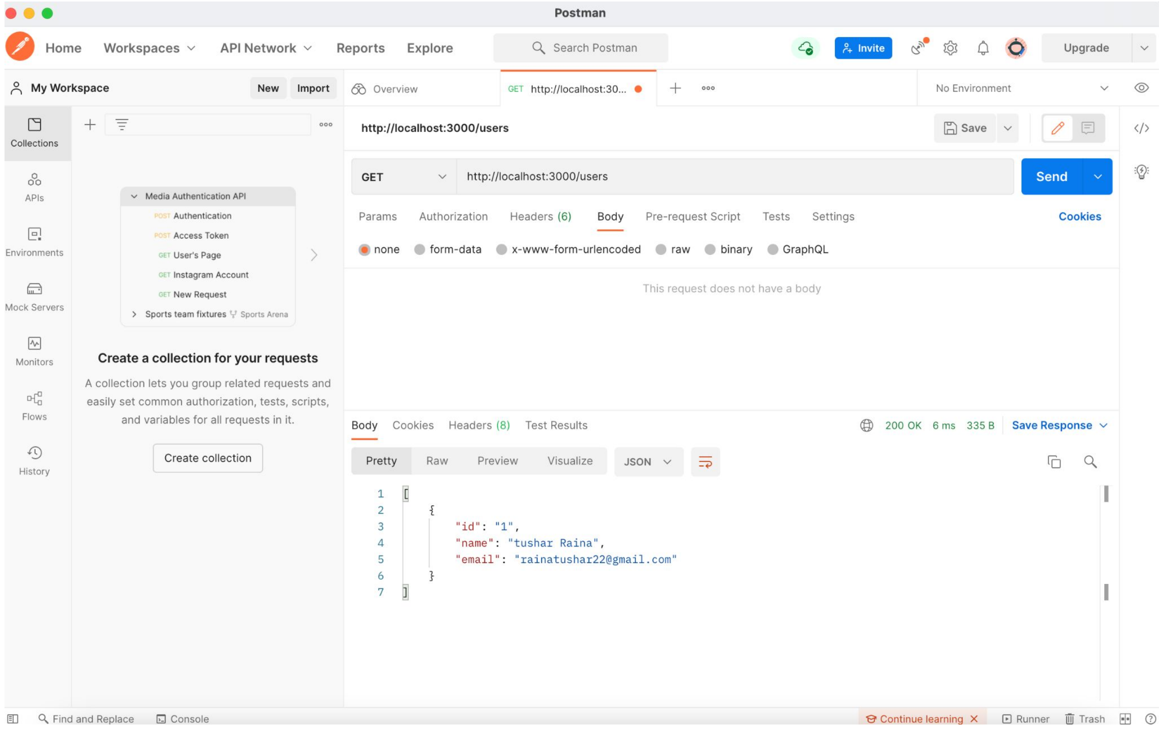
http://localhost:3000/user/2

DELETE http://localhost:3000/user/2

200 OK 13 ms 274 B

```
User is deleted
```

PostMan Setup and Testing : Testing GET



http://localhost:3000/users

GET http://localhost:3000/users

This request does not have a body

Create a collection for your requests

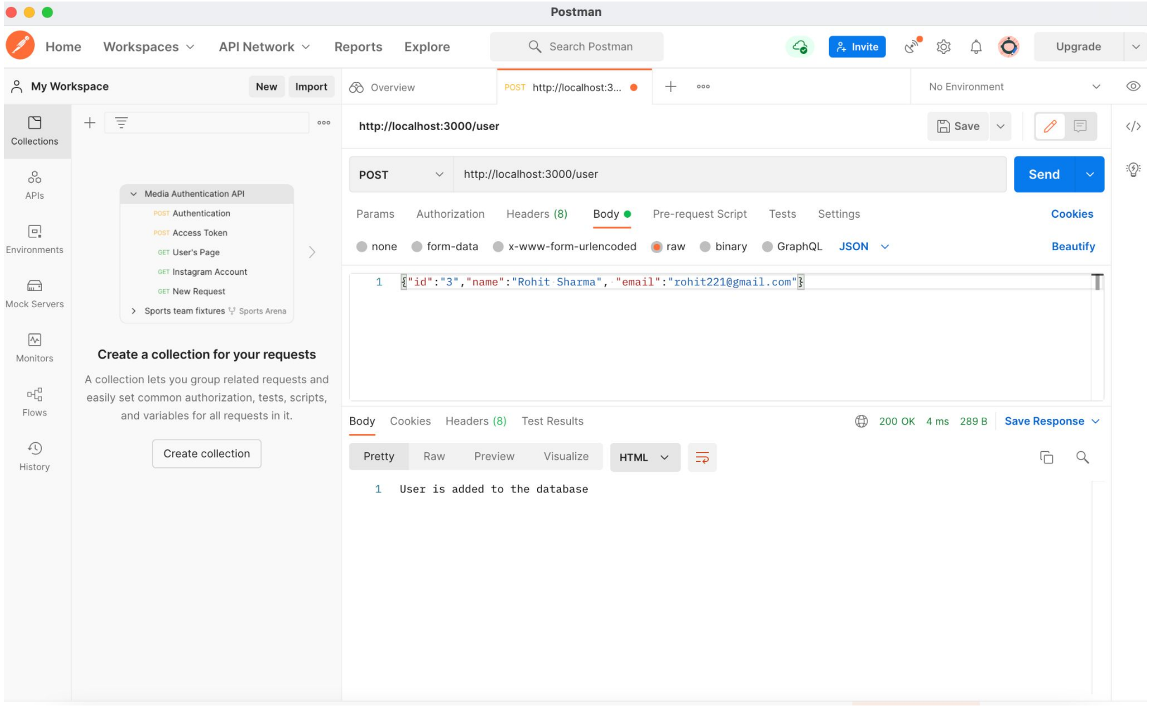
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create collection

Body — 200 OK 6 ms 335 B

```
[
    {
        "id": "1",
        "name": "tushar Raina",
        "email": "rainatushar22@gmail.com"
    }
]
```

PostMan Setup and Testing : Testing POST



http://localhost:3000/user

POST http://localhost:3000/user

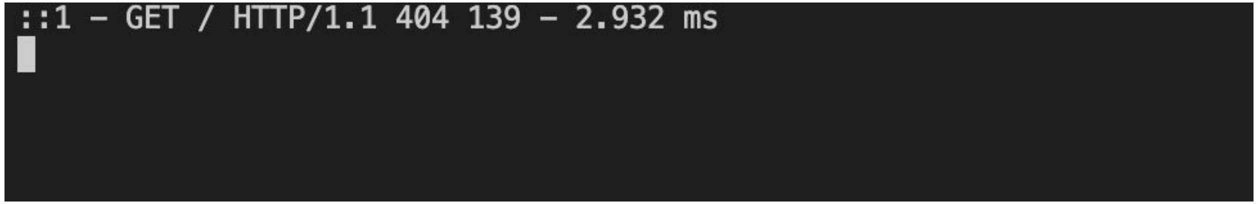
```
{"id":"3","name":"Rohit Sharma", "email":"rohit221@gmail.com"}
```

```
User is added to the database
```

MCQ’S

Homework

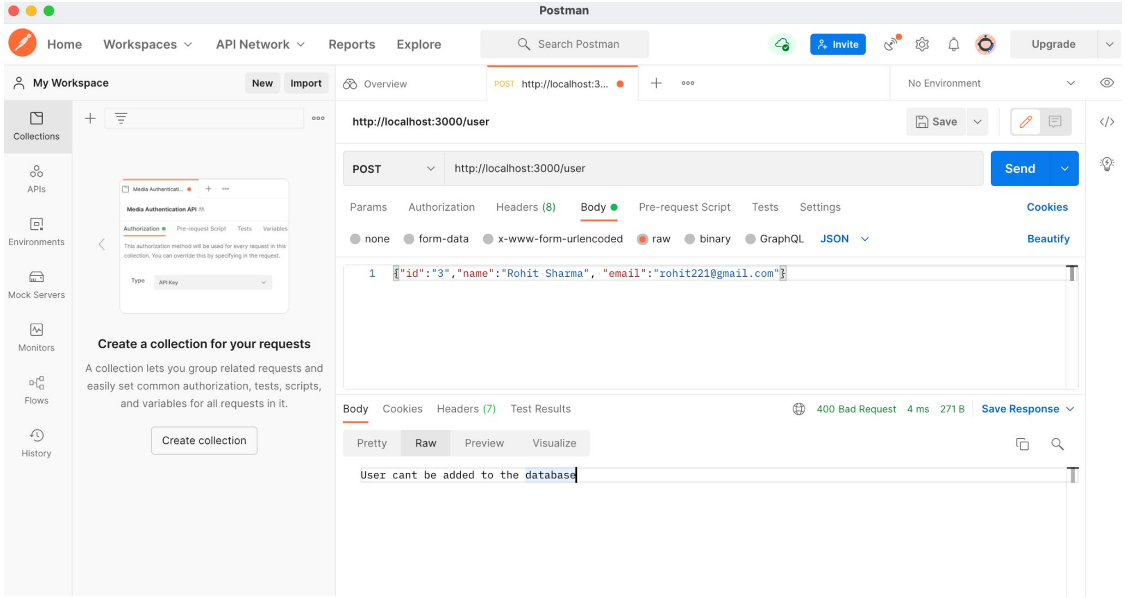
Let us build a simple web server using morgan as middleware for logging



```
::1 - GET / HTTP/1.1 404 139 - 2.932 ms
```

Homework

Add Error Logic in POST API that we just built. In case we try to add same user twice . Return error message with 400 status code



http://localhost:3000/user

POST http://localhost:3000/user

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{"id":"3","name":"Rohit Sharma", "email":"rohit221@gmail.com"}
```

Body Cookies Headers (7) Test Results — 400 Bad Request 4 ms 271 B

```
User cant be added to the database
```

Thankyou