



Express.js

CRUD



Create



Read



Update



Delete

req.params

- Lets see a very common behavior we wish our API to have
- Lets say we want to select one landmark by id
 - a parameter is needed to define the id
- **Route parameters -> *named* URL segments, used to capture values specified in position in the URL.**
- **req.params object-> used to access the aforesaid values**

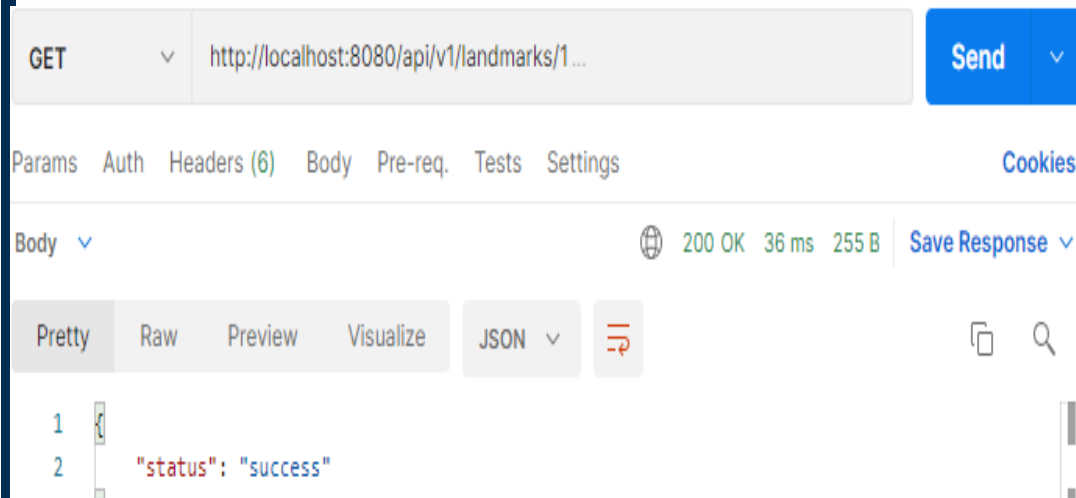
req.params

- Route path: `/users/:userId/programmingL/:id`
- Request URL: `http://localhost:8080/users/2/ programmingL /1`
- req.params: `{ "userId": "2", " id ": "1" }`

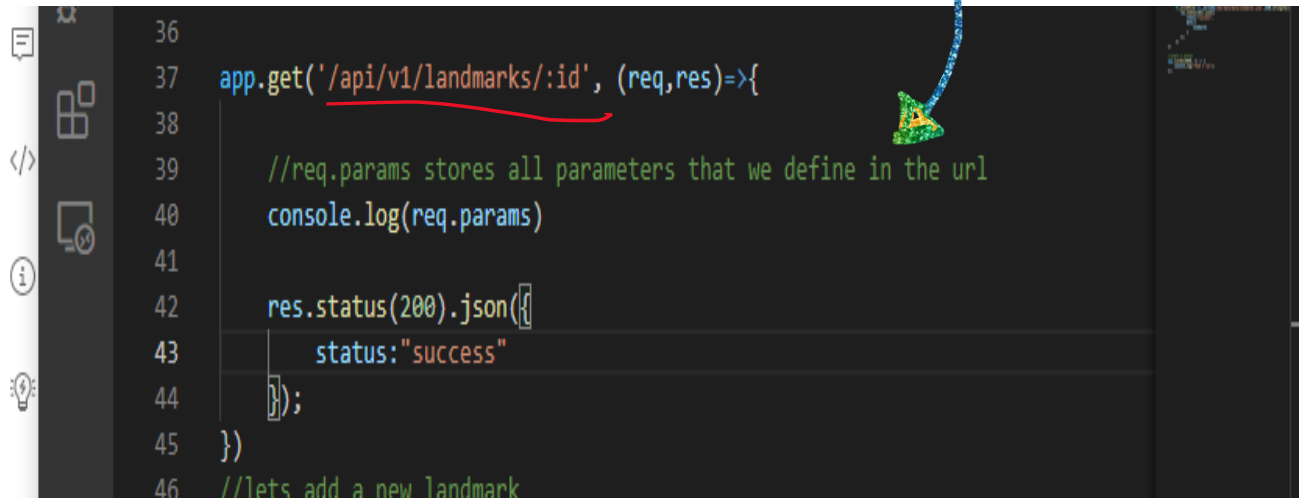
req.params

- Lets see a very common behavior we wish our API to have.
- We want to select one landmark (in this use case scenario) by id

Parameter (:id): The :id in the route path indicates a path parameter. It means that part of the URL is a variable, and its value will be captured and made **available in the req.params object**. In this case, it's capturing an identifier for a landmark. For example, if the user visits /api/v1/landmarks/123, the value 123 will be available in req.params.id.

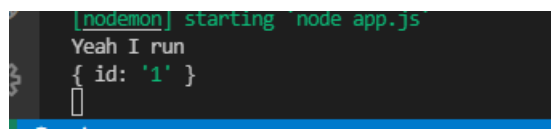


```
GET http://localhost:8080/api/v1/landmarks/1...  
Send  
Params Auth Headers (6) Body Pre-req. Tests Settings Cookies  
Body 200 OK 36 ms 255 B Save Response  
Pretty Raw Preview Visualize JSON  
1 {  
2   "status": "success"
```



```
36  
37 app.get('/api/v1/landmarks/:id', (req,res)=>  
38  
39   //req.params stores all parameters that we define in the url  
40   console.log(req.params)  
41  
42   res.status(200).json({  
43     status:"success"  
44   });  
45 })  
46 //lets add a new landmark
```

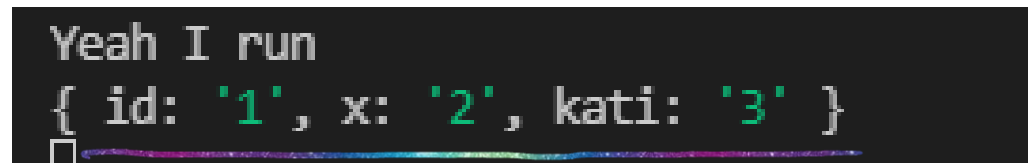
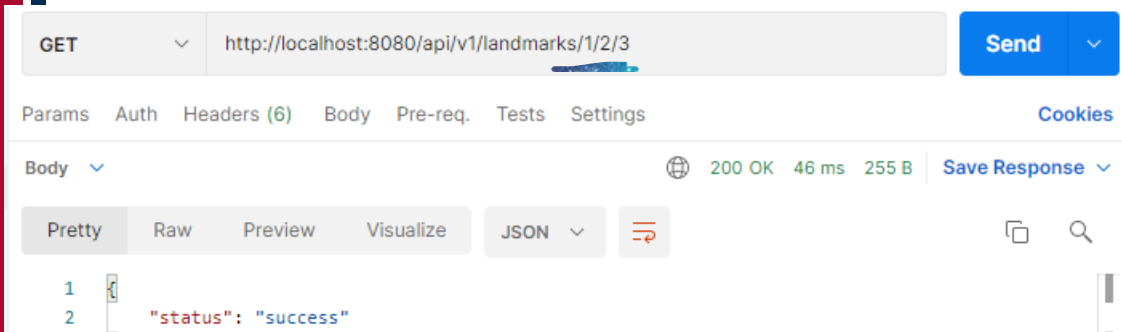
here is our param →



```
[nodemon] starting node app.js  
Yeah I run  
{ id: '1' }
```

req.params

- Could we define more than one parameters in the url?
- Yeap we can!



req.params

- We can also define optional parameters
- That we may or may not specify at the endpoint

```
app.get('/api/v1/landmarks/:id/:x/:kati?', (req, res) => {
```

GET

http://localhost:8080/api/v1/landmarks/1/2/

Send

Yeah I run

```
{ id: '1', x: '2', kati: undefined }
```

Select by id

- We want to find a landmark by its id
- So how could we do that?

Select by id

- We want to find a landmark by its id
- So how could we do that?
- We have a landmarks object... so we can use the *find* method
- find() method -> returns the value of the **first element** in the provided array that satisfies the provided testing function
- no values -> undefined is returned.

Select by id

find() method -> returns the value of the **first** element in the provided array that satisfies the provided testing function

i.E

```
var array1 = [2, 12, 1, 3, 13, 4, 66];
```

```
var found = array1.find(element => element > 10);
```

```
console.log(found); // expected output: 12
```

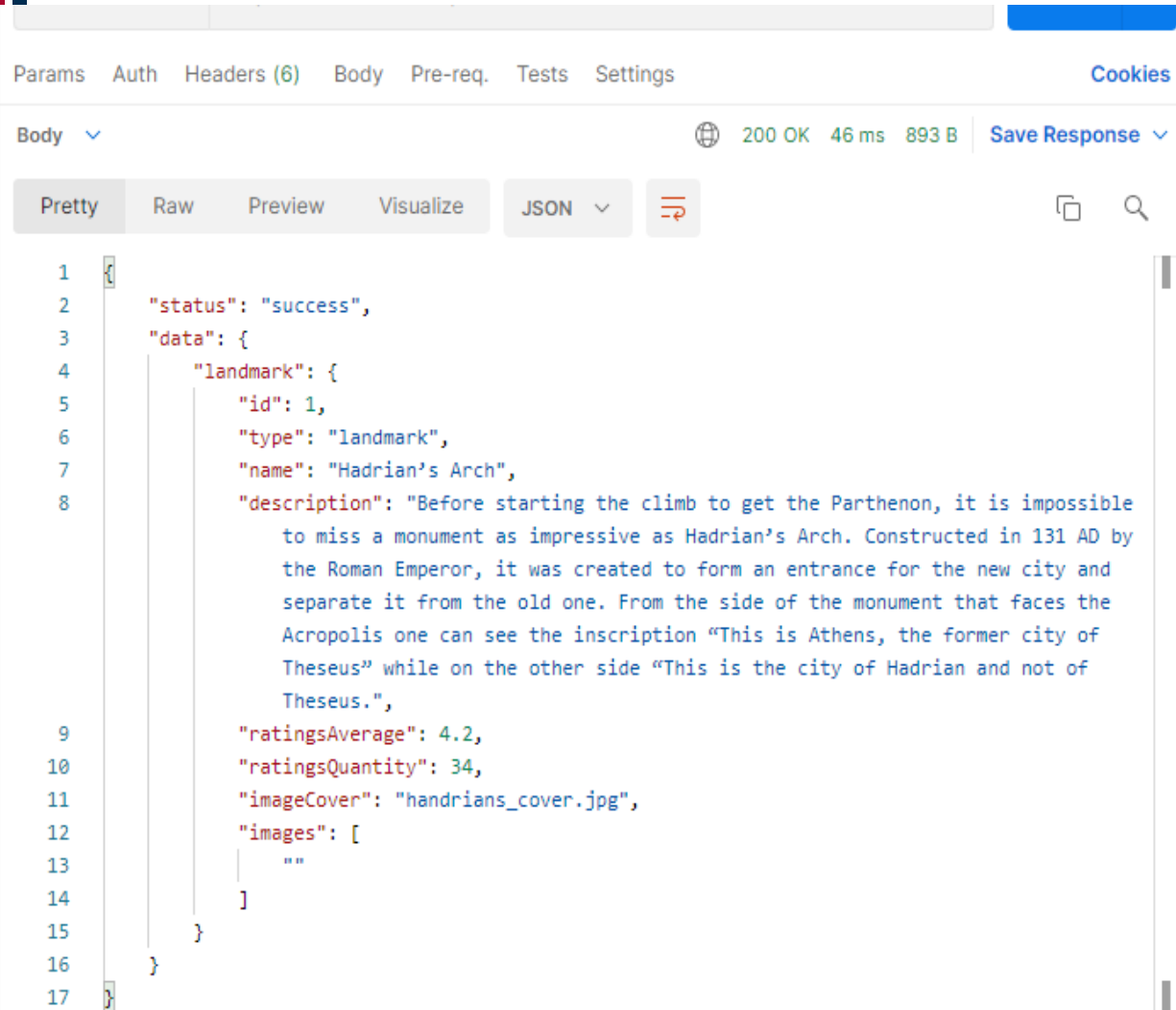
Select by id

Check here: id is string in the request so we need to convert it to number!



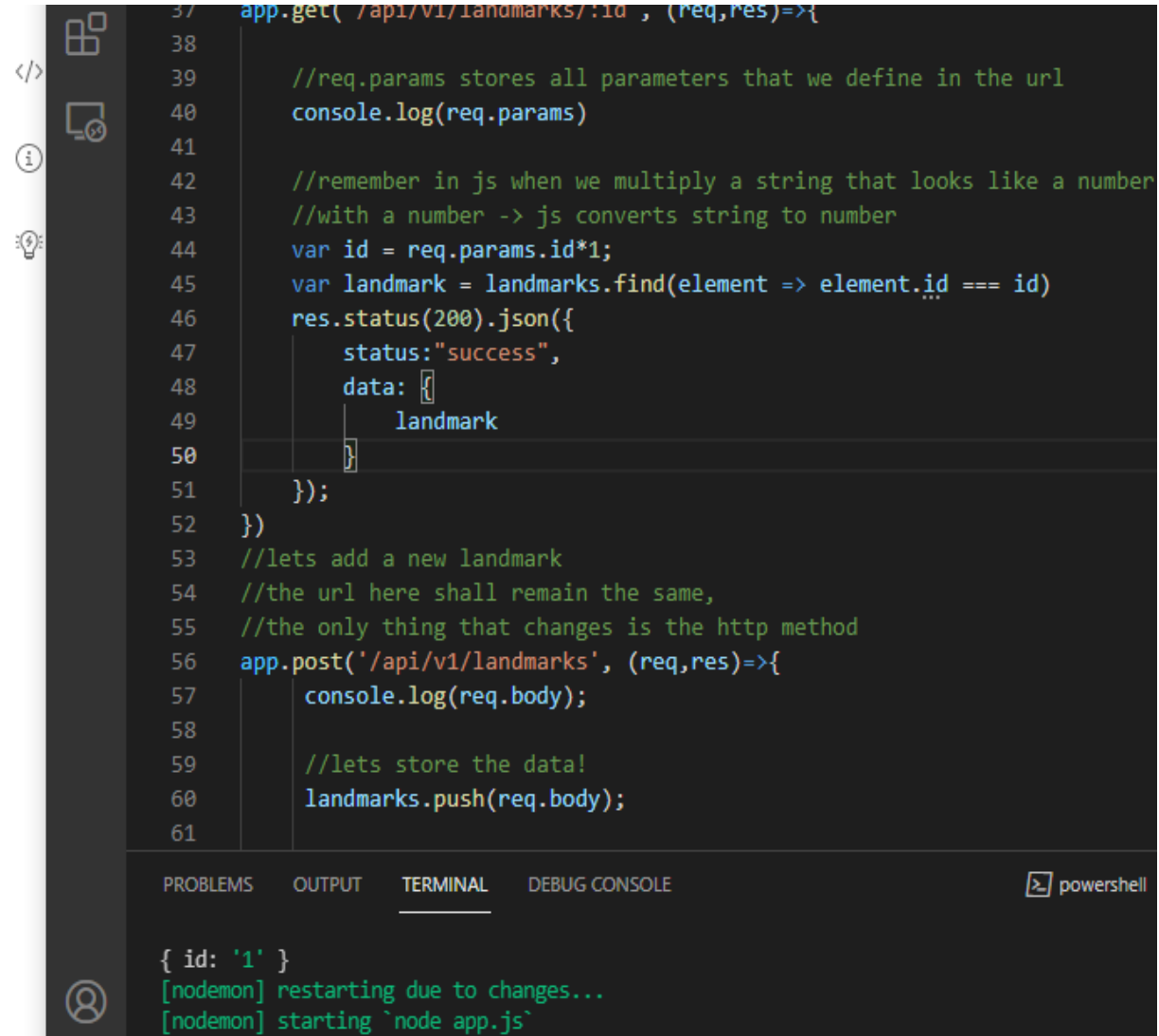
```
//remember in js when we multiply a string that looks like a number  
//with a number -> js converts string to number  
var id = req.params.id*1;  
var landmark = landmarks.find(element => element.id === id)  
res.status(200).json({
```

Select by id



The screenshot shows the 'Body' tab of a web browser's developer tools. The response is a JSON object with the following structure:

```
1 {
2   "status": "success",
3   "data": {
4     "landmark": {
5       "id": 1,
6       "type": "landmark",
7       "name": "Hadrian's Arch",
8       "description": "Before starting the climb to get the Parthenon, it is impossible
9         to miss a monument as impressive as Hadrian's Arch. Constructed in 131 AD by
10        the Roman Emperor, it was created to form an entrance for the new city and
11        separate it from the old one. From the side of the monument that faces the
12        Acropolis one can see the inscription 'This is Athens, the former city of
13        Theseus' while on the other side 'This is the city of Hadrian and not of
14        Theseus.'",
15       "ratingsAverage": 4.2,
16       "ratingsQuantity": 34,
17       "imageCover": "handrians_cover.jpg",
18       "images": [
19         ""
20       ]
21     }
22   }
23 }
```



The screenshot shows a code editor with the following JavaScript code:

```
37 app.get('/api/v1/landmarks/:id', (req, res) => {
38
39   //req.params stores all parameters that we define in the url
40   console.log(req.params)
41
42   //remember in js when we multiply a string that looks like a number
43   //with a number -> js converts string to number
44   var id = req.params.id*1;
45   var landmark = landmarks.find(element => element.id === id)
46   res.status(200).json({
47     status: "success",
48     data: {
49       landmark
50     }
51   });
52 })
53 //lets add a new landmark
54 //the url here shall remain the same,
55 //the only thing that changes is the http method
56 app.post('/api/v1/landmarks', (req, res) => {
57   console.log(req.body);
58
59   //lets store the data!
60   landmarks.push(req.body);
61 }
```

The terminal window at the bottom shows the following output:

```
{ id: '1' }
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
```

Select by id

Handle the situation of no such id in Json



```
//remember in js when we multiply a string that looks like a number
//with a number -> js converts string to number
var id = req.params.id*1;
var landmark = landmarks.find(element => element.id === id)
if(!landmark){
  res.status(404).json({
    status: "fail",
    message: "Not Found"
  });
}
res.status(200).json({
  status: "success",
  data: {
    landmark
  }
});
})
```

Multiple callback functions

```
app.get('/api/v1/landmarks',(req,res,next)=>{
  console.log("More than one callback can handle a route, dont' forget next!");
  next();
},(req,res)=>{
  res.status(200).json({
    status:"success",
    results: landmarks.length,
    data:{
      landmarks
    }
  });
})
```

More than one callback function can handle a route-> don't forget to specify next

Array of route handlers

```
var a=(req,res,next)=>{  
  console.log("hi there");  
  next();  
}  
var b=(req,res,next)=>{  
  console.log("My friend");  
  next();  
};  
var c=(req,res,)=>{  
  res.send('Hello!')  
};  
  
app.get('/api/v1/array',[a,b,c]);
```

Response methods

- *methods* on the response object (**res**) in the following table can **send** a response to the client, and **terminate** the request-response cycle.
- Note that If none of these methods are called from a route handler-> the client request will be left **hanging**

Method	Description
<code>res.download()</code>	Prompt a file to be downloaded.
<code>res.end()</code>	End the response process.
<code>res.json()</code>	Send a JSON response.
<code>res.jsonp()</code>	Send a JSON response with JSONP support.
<code>res.redirect()</code>	Redirect a request.
<code>res.render()</code>	Render a view template.
<code>res.send()</code>	Send a response of various types.
<code>res.sendFile()</code>	Send a file as an octet stream.
<code>res.sendStatus()</code>	Set the response status code and send its string representation as the response body.

Source <https://expressjs.com/>

Update Data

- For updating data we have two methods:
- PUT -> is used to modify an existing entity
 - it replaces an entity.-> If we don't include a property that an entity contains, it should be removed
- PATCH -> is used to apply a **partial** modification to a resource.
 - Used to update only the properties we wish
- We shall see such an example when we **get to databases**

Delete

```
//we shall see more in databases about that
app.delete('/api/v1/landmarks/:id', (req,res)=>{
  //204 status usually stands for no content
  //this is what we usually use with delete
  res.status(204).json({
    status:"success",
    data: null
  });
})
//start a server
```


Structure our code a little bit better

- Now, in order to be a little bit more organized, we are going to separate the **http methods** from the **route handler functions**

Structure our code a little bit better

```
const getAllLandmarks = (req,res)=>{  
  
  res.status(200).json({  
    status:"success",  
    results: landmarks.length,  
    data:{  
      landmarks  
    }  
  });  
}  
  
const getLandmarkById = (req,res)=>{  
  
  //req.params stores all parameters that we define in the route  
  console.log(req.params)  
  
  //remember in js when we multiply a string that looks like  
  //with a number -> js converts string to number  
  var id = req.params.id*1;  
  var landmark = landmarks.find(element => element.id === id);  
  if(!landmark){  
    res.status(404).json({  
      status:"fail",  
      message: "Not Found"  
    });  
  }  
}
```

```
app.get('/api/v1/landmarks',getAllLandmarks)  
  
app.get('/api/v1/landmarks/:id',getLandmarkById )  
  
//lets add a new landmark  
app.post('/api/v1/landmarks',addLandmark)  
  
//update one property with patch  
app.patch('/api/v1/landmarks/:id', updateLandmarkById)  
  
//we shall see more in databases about that  
app.delete('/api/v1/landmarks/:id', deleteLandmarkById )  
  
//start a server  
app.listen(8080, () => {  
  console.log('Yeah I run');  
});
```



assign callback functions(route handlers) to variables
(req,res)=>{} this callback function is called the route handler

Can we make it better?

- **app.route()** method -> **returns instance of a single route**, which we can then use to handle HTTP methods
- This allows us to group all same urls... avoid duplicate route names !
- Easier to read and maintain code

- Lets see how

Can we make it better?

```
app.route('/api/v1/landmarks')  
  .get(getAllLandmarks)  
  .post(addLandmark)  
  
app.route('/api/v1/landmarks/:id')  
  .get(getLandmarkById)  
  .patch(updateLandmarkById)  
  .delete(deleteLandmarkById)
```

To be continued...

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

<https://www.geeksforgeeks.org/express-js-app-route-function/>