
Task

# Express II: Create a custom API with Express

Visit our website

# Introduction

## WELCOME TO THE CREATE A CUSTOM API WITH EXPRESS TASK!

As a full-stack web developer, it is important that you are able to create custom Restful web APIs that can be used to store and manipulate data on the backend of your full-stack application. By the end of this task, you should understand what a Restful Web API is, how it is used and how to create a custom Restful Web API using Express.

Get in touch

**Connect for support**

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

A note from the

**HyperionDev Team**

There's a reason that true full-stack web developers are sometimes referred to as unicorns. It's the rarity factor. A genuine full stack web developer has such a diverse range of skills that he or she is hard to find. Let's look in more detail at what these developers do and why it's so hard to find full-stack web developers **here**.

## MORE ON RESTFUL APIS

You learned a bit about RESTful APIs in the task on fetching data with React. Let's look at the other end of that API: the backend.

As a refresher, according to **Oracle**, RESTful web services are based on the following principles:

- RESTful web services expose resources using URIs.
- Resources are manipulated using PUT, GET, POST, and DELETE operations.
    - PUT creates a new resource
    - DELETE deletes a resource.
    - GET retrieves the current state of a resource.
    - POST transfers a new state onto a resource.
- Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON and others.
- Interaction with resources is stateless. State information is exchanged using techniques like URI rewriting, cookies, hidden form fields and embedding state information in response messages.

## CREATE A CUSTOM RESTFUL API USING EXPRESS

A Restful web API is code that is written to respond to HTML PUT, GET, POST and DELETE requests. To create a Restful API, we are going to write JavaScript functions using Express and Node to handle each of these requests.

Express has built-in middleware routing functions to handle each of these HTTP request methods. In the previous task, we already used the `app.get()` function to respond to HTTP GET requests with the specified URL path ('`/`'). The app object contains methods to handle each of the HTTP verbs: `app.post()`, `app.get()`, `app.put()` and `app.delete()`.

Like the `app.get()` method, each of these methods takes two arguments:

- The route. These methods are used to perform routing. Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- A callback function. The callback function that is passed as the second argument to the app.get/post/put/delete() method acts as a route handler.

Each route handler that we write will be used to either **create** data (e.g. create a JSON file), **read** data, **update** data or **delete** data. These operations are often referred to as CRUD (Create, Read, Update and Delete) operations.

Each CRUD operation can be accessed using a corresponding HTTP request as shown in the table below:

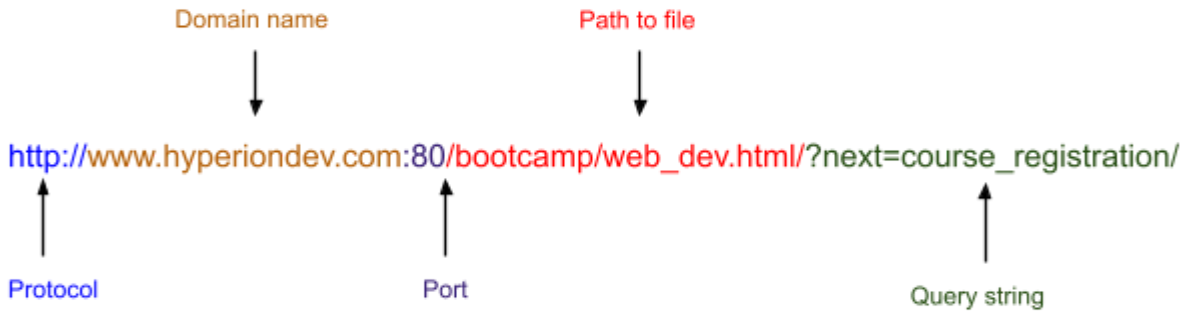
| HTTP verb | CRUD operation | Express method | Description |
|---|---|---|---|
| Post | Create | `app.post()` | Used to submit some data about a specific entity to the server. |
| Get | Read | `app.get()` | Used to get a specific resource from the server. |
| Put | Update | `app.put()` | Used to update a piece of data about a specific object on the server. |
| Delete | Delete | `app.delete()` | Used to delete a specific object. |

You create an API that receives a URI with an HTTP request and uses the appropriate Express routing middleware to call the corresponding functions that handle the CRUD operations.

If we are going to be able to add and update data on our servers though, we need a way to be able to pass our data from the browser to the server.

## PASSING DATA THROUGH TO THE SERVER USING THE REQUEST OBJECT

An important role of the server is to receive necessary data that is passed through from the browser. This data can be passed from the client to the server using the URL. Remember that, a URL contains a lot of information:




1. It identifies the protocol being used to send information. In the example above, the protocol being used is HTTP.
2. It identifies the domain name of the web server on which the resource can be found, e.g. www.hyperiondev.com.
3. It identifies the port on the server. In this example, the port number is given as port 80. In reality, if the default HTTP ports are used (port 80 is the default for HTTP, port 443 for HTTPS) they don't have to be given in the URL.
4. It gives the path to the resource on the web server, e.g. /bootcamp/web_dev.html
5. Data can be passed using **parameters** (as shown in the image below) or using a **query string** (as shown in the image above).

With query strings, data are passed as key-value pairs (?key=value&key2=value2), e.g. ?next=course_registration.


The image above illustrates what a URL may look like if parameters are added to the URL. In the example above, the parameter '2315' may represent the id of a student at HyperionDev.

To access the data passed through using the URL, we use the `req` object that is passed through as an argument to the `app.post` or `app.put` route handler.

`req.params` is used to get parameters from a URL and `req.query` is used to get data from a query string.

See in the example below how the code `req.query.name` is used to get the value of the key-value pair where the key is 'name'.

```
app.post('/', (req, res) => {
    fileHandler.writeFile('person.json', `{name: ${req.query.name}}`, (err) => {
        if (err) throw err;
        res.send('File created!');
    });
})
```

URL: localhost:3000?name=Gareth

See in the example below how the code `req.params.name` is used to get the value of the parameter 'name' that is defined in the route argument of the `app.put()` method.

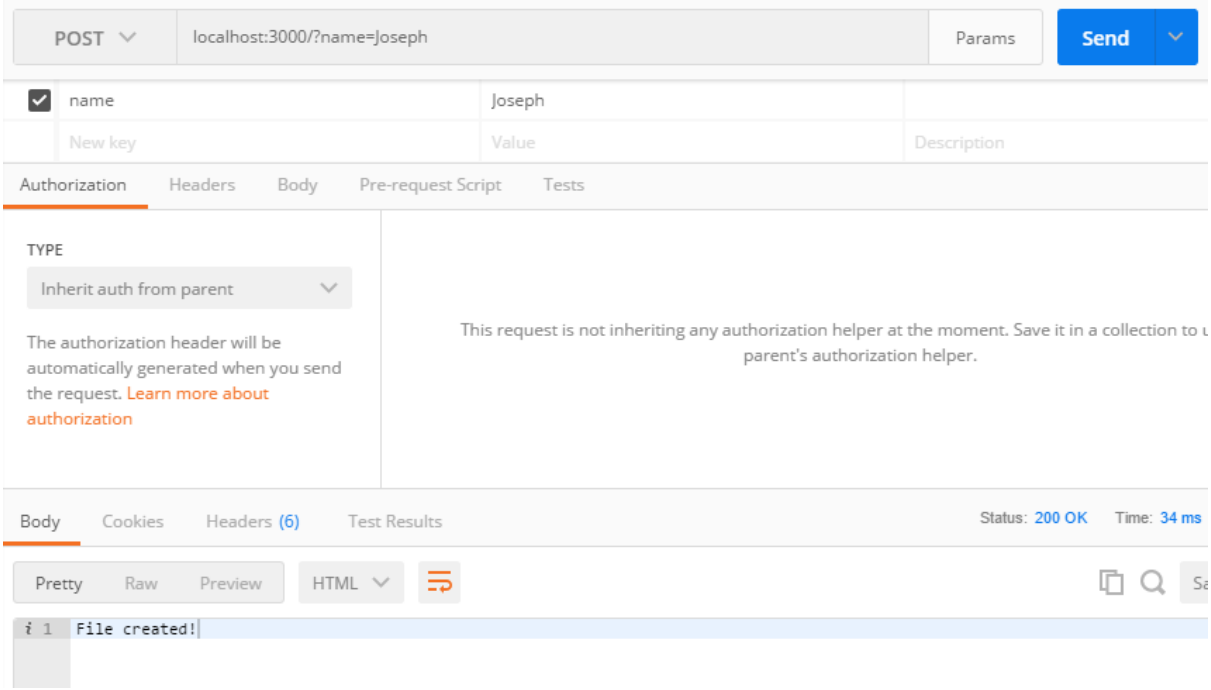
```
app.put('/:name', (req, res) => {
    fileHandler.writeFile('person.json', `{name: ${req.params.name}}`, (err) => {
        if (err) throw err;
        res.send('File updated!');
    });
})
```

URL: localhost:3000/Sue

See the code example that accompanies this task to see this code in action. Remember that you should be using JSON data (**.json**) when creating an API. JSON data can be defined by double quotes around both the key and the value.

## TEST YOUR API USING POSTMAN

You will later learn how to use your front-end to pass data through to the server but, for now, we can test that our server is sending and receiving data using Postman. Postman is a free API development environment. You can see how postman works and download it **here**. The image below shows the results you could expect from using postman to send a post request with the parameters shown in the image to the server that we configured with the code above.




Some IDEs, like **IntelliJ** for instance, also have built-in functionality for testing APIs.

**SPOT CHECK 1**

Let's see what you can remember from this section.

1. Match up the following:

| | |
|---|---|
| 1. Post | a. Used to update a piece of data about a specific object on the server. |
| 2. Get | b. Used to delete a specific object. |
| 3. Put | c. Used to submit some data about a specific entity to the server. |
| 4. Delete | d. Used to get a specific resource from the server. |

# Instructions

- The Express related tasks would see you creating apps that need some modules to run. These modules are located in a folder called 'node_modules', which is created when you run the following command from your command line: `npm install` or similar. Please note that this folder typically contains thousands of files which, if you're working directly from Dropbox, has the potential to ***slow down Dropbox sync and possibly your computer***. As a result, please follow this process when creating/running such apps:
    - Create the app on your local machine (outside of Dropbox) by following the instructions in the compulsory task.
    - When you're ready to have your mentor review the app, please **delete the node_modules** folder.
    - Compress the folder and upload it to Dropbox.
    - Your mentor will, in turn, decompress the folder, install the necessary modules and run the app from their local machine.

- Read through the example files that accompany this task before attempting the compulsory task. To execute the code in the example files:
    - Copy the example folder to your local computer.
    - Navigate to the folder that contains the example files from your command line interface.
    - Use the command line interface to type `npm install` to install all the needed dependencies (including Express). This command will use the package.json file in the example folder to see which dependencies to install.
    - Use the command line interface to type `npm start` to start this application.
- Remember that the primary goal of this task is to get to grips with writing code for the backend of your web application. Therefore, even though your app will display output in the browser, your main concern should be the server-side functionality and not the appearance of the front-end.

# Compulsory Task 1

Follow these steps:

- Download Postman **here**.
- Copy the example folder that accompanies this task to your local PC. In your command line interface, navigate to this folder and type `npm install`.
- Run the **app.js** file by typing `npm start`.
- Test the Restful API created in app.js with Postman. Create a folder called 'Screenshots' in the folder for this task in Dropbox and insert screenshots (make sure each screenshot includes the response) of how you have used Postman to test this API to:
    - Make an HTTP post request to the API with the query string `?name=Jack` (e.g. `http://localhost:3000/?name=Jack`) Make sure you enter the correct port number in the URL you use for testing.
    - Make an HTTP put request to the API with the URL containing the parameter value 'Samantha' (**http://localhost:3000/Samantha**)
    - Make an HTTP get request to the API
    - Make an HTTP delete request to the API

# Compulsory Task 2

Follow these steps:

- Create a Restful API using Express that will allow the user to store a list of 'Web project' items.
    - When the user navigates to **http://localhost:8080/api** an array of 'Web Project' items should be returned. E.g. of array:`[{"id": 1, "title": "React Game!", "description": "Tic tac toe game created using Create React app.", "URL": "http://heroku/myapp/game/" }, {"id": 2, "title":"Online store", "description": "Online store created with HTML, CSS and JavaScript.", "URL": "https://git.com/myrepos/shop/index"}]`
    - The user should be able to use Postman to make an HTTP Post request that adds an additional item to the list of Web Project items.

    - The user should be able to use Postman to make an HTTP Delete request that deletes an item with a specific id from the list of Web Project items.
    - The user should be able to make an HTTP Put request to update the title or description of an item on the list of Web Project items using Postman.
- Create a file called **readme.md** in the project folder of this application that provides the user instructions for testing this API with Postman.

## Completed the task(s)?

Ask your mentor to review your work!

**Review work**

## Things to look out for:

1. Make sure that you delete 'Node_modules' before submitting the code.

Rate us

**Share your thoughts**

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

**Click here** to share your thoughts anonymously.

**SPOT CHECK 1 ANSWERS**

1.

| | |
|---|---|
| 1. Post | c. Used to submit some data about a specific entity to the server. |
| 2. Get | d. Used to get a specific resource from the server. |
| 3. Put | a. Used to update a piece of data about a specific object on the server. |
| 4. Delete | b. Used to delete a specific object. |