the Master Course

{CODENATION}

Backend Development

Request, Response & CRUD

{CODENATION}

Learning Objectives

To be able to use HTTP Routes in Express.

To be able to use the Thuder Client API client extension.

To be able to define CRUD operations

To explain how CRUD operations link to HTTP verbs POST, GET, PUT and DELETE.

To program CRUD functionality into a basic Express API.

The Plan?

This session, we will be building a new basic web server and putting some data (an object array) into that server.

We will also focus on the **Create**, **Read**, **Update** and **Delete** functions.

This will help us when we look at databases tomorrow.

Let’s install Express.js!

Create a new project folder.

In Terminal:

```
npm init -y
```

{} package.json ?

CRUD

- Create a .gitignore file.
- Add node_modules to it.

.gitignore

```
1    node_modules
```

CODE ALONG

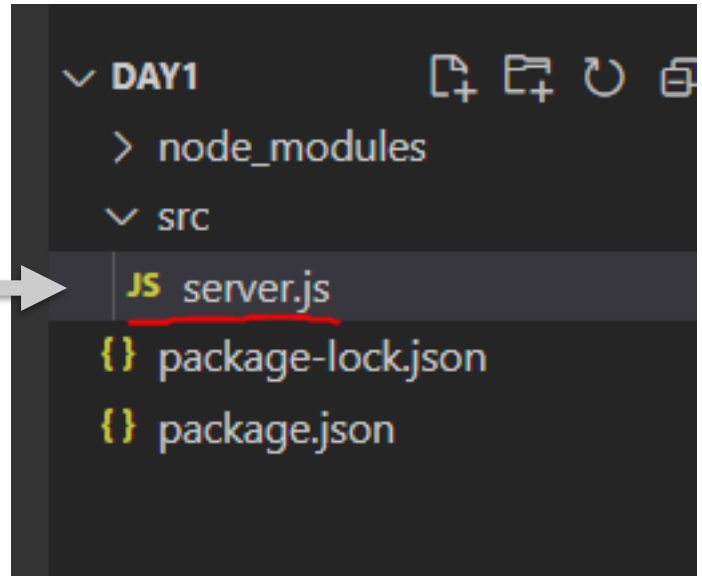
In Terminal:

```
npm install express
```

New Express Server - recap

In the **src** folder…

add a new file: server.js



New Express Server - recap

We need to import the express library into our project

JS server.js

Day1 > src > JS server.js > ...

```
1  const express = require('express');
2  const app = express();
```

Let's Get On With It…

We will use **app.get()** for our server routes.
https://expressjs.com/en/4x/api.html#app.get.method

This **get** method identifies the path, and the function for getting there.

It **GETs** information from the server – a kind of Read.

Let's Get On With It...

In the server.js file...

add the **use** method

```
src > JS server.js > ...
1   const express = require('express');
2   const app = express();
3
4   app.use('/book', (request, response) => {

6   });
```

NOTE: Can also be seen as (req, res).

We're Getting On With It…

```
4   app.use('/book', (request, response) => {
5       const book = {
6           title: 'LOTR',
7           author: 'J.R.R Tolkein',
8           genre: 'fantasy'
9       };
10
11      const successResponse = {
12          message: 'Book successfully retrieved',
13          book: book
14      };
15
16      response.send(successResponse);
17  });
```

Add some **data**

Create an **object** called **book** and give it some details…

Add a **response message**

Create another **object** and give it a **text** message and the **book** object.

Send the 'success' response

The Route has been created!

We must now use **app.listen** to tell it to listen on the port.

```
15
16        response.send(successResponse);
17    });
18
19    app.listen(5001, () => console.log('Server is listening on port 5001.'));
```

Add the port number and a helpful message.

Start the Server

Open a fresh web browser and we will access the book webpage.

Enter this into the URL:
http://localhost:5001/book

localhost:5001/book

```
// 20230911133828
// http://localhost:5001/book

{
  "message": "Book successfully retrieved",
  "book": {
    "title": "LOTR",
    "author": "J.R.R Tolkein",
    "genre": "fantasy"
  }
}
```

Web browsers are

… not always the best way of talking to API servers.

Whilst we can perform **GET requests**, we can't really do **POST requests** in a web browser.

We need a piece of software which assist us to have two-way communications.

Thunder Client!

By Ranga Vadhineni

In your **VS Code extensions**, search for, and install **'thunder client'**

Thunder Client

Types of request

Address Bar

Create a new HTTP request.

Request list

To open Thunder Client.

Send the request

Display window

New Request — Day1

THUNDER CLIENT

New Request

Activity | Collections | Env

filter activity

GET localhost:5001/book
just now

JS servers | New Request

GET | http://localhost:5001/book | Send

Query | Headers 2 | Auth | Body | Tests | Pre Run

Query Parameters

parameter | value

Status: 200 OK | Size: 108 Bytes | Time: 4 ms | Response

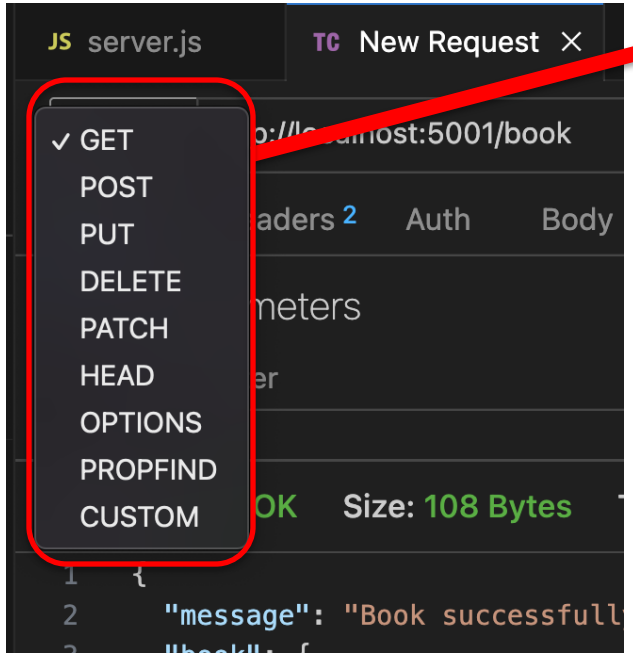
Copy

```
1 {
2   "message": "Book successfully retrieved",
3   "book": {
4     "title": "LOTR",
5     "author": "J.R.R Tolkein",
6     "genre": "fantasy"
7   }
8 }
```

CODENATION

Thunder Client

It allows us to test our Web Servers by generating requests to retrieve responses quickly and easily.



We can choose from this list of standard HTTP Verbs.

The main types of request that we will use are:

POST - (**Create**)
GET - (**Read**)
PUT - (**Update**)
DELETE - (**Delete**)

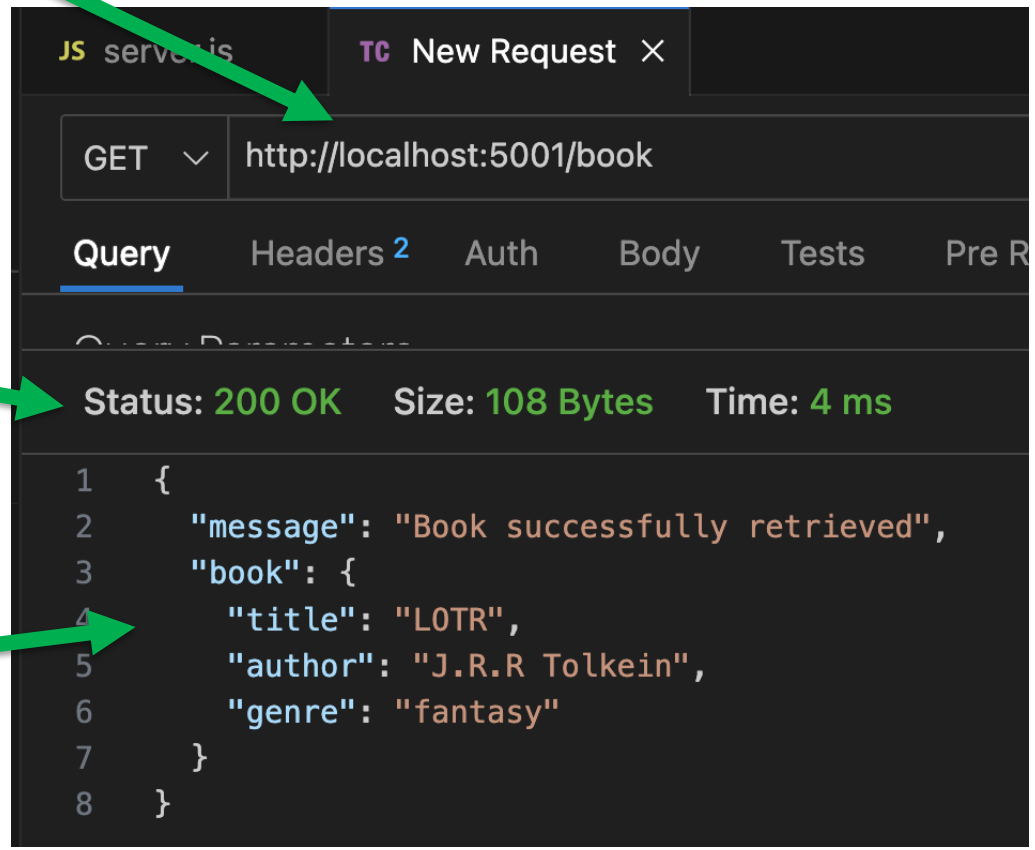
Thunder Client

In the address bar, enter the same URL used earlier in your browser. Click **SEND**.

Status codes are the same codes used for websites (200, 401, 404, 418, 500, etc)

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

You can see your book object data, here.



You now have a basic working server!

We can get information from our server, but we want it to be more sophisticated

Let's create a little array!

We want the ability to add more book details to our server. An array can hold many objects…

Can you remember some array methods?

CRUD

In the server.js file…

add an **empty array**

We will need to set our first **book id**

Move the **book** object out of the **app.get** block and add id – with a value set to **book_id**

```
2   const app = express();
3
4   const books = [];
5   book_id = 1;
6
7   const book = {
8       id: book_id,
9       title: 'LOTR',
10      author: 'J.R.R Tolkein',
11      genre: 'fantasy'
12  };
13
14  app.use('/book', (request, res
```

CODE ALONG

CODENATION

Add to our books array…

```
        title: 'LOTR',
        author: 'J.R.R Tolkein',
        genre: 'fantasy'
    };

    books.push(book);
    book_id += 1;
```

The **push method** is used here to add the **book** to the **books array**.

Increment the id counter, too!

```
app.use('/book', (request, response) => {

    const successResponse = {
        message: 'Books successfully retrieved',
        book: books
    };
```

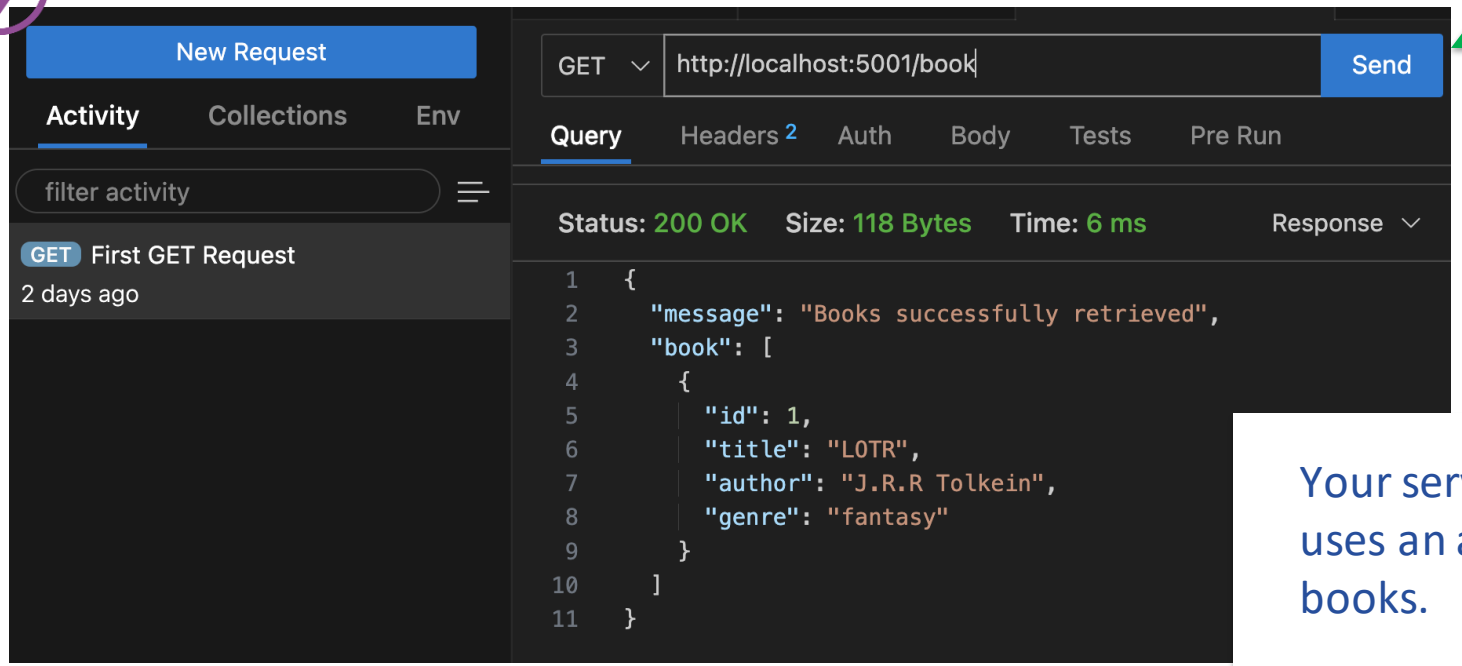
We should also update the response message…

Save and **run** the server up: **node src/server.js**

Did you know that you can rename Thunder Client requests?

In **Thunder Client**: right-click and **rename** the request to '**First GET Request**':

Click **Send** to see the response with the **new id**.



Your server now uses an array of books.

How about adding other books?

We have used a **GET** (app.get) which is a read method.

We can now use a **POST** method to add to our array…

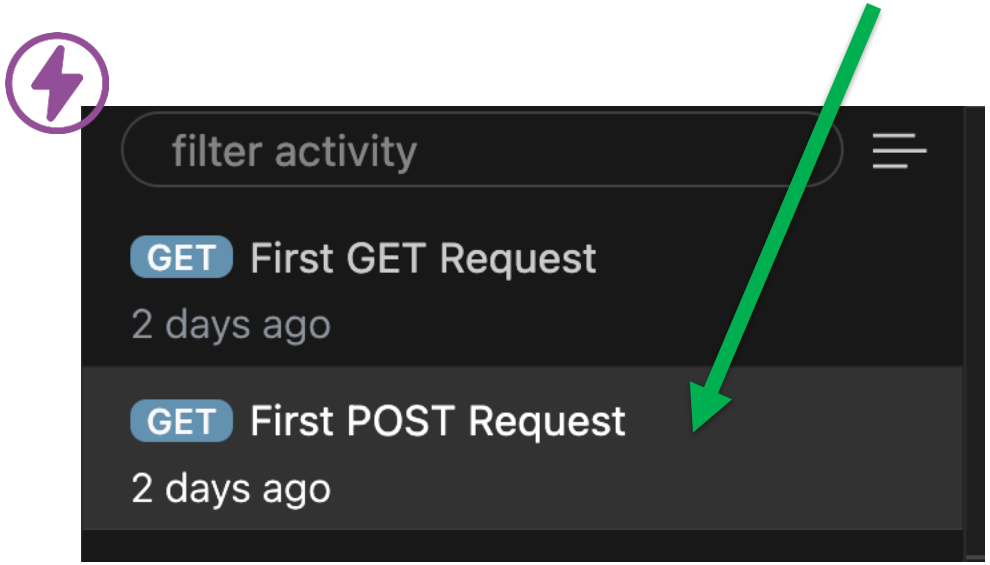
CODE ALONG

CRUD

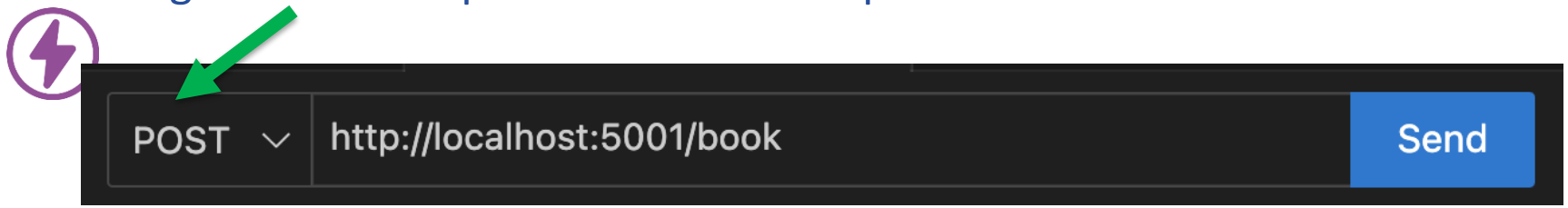
In Thunder Client …

… duplicate the '**First GET Request**' and **rename** it to: '**First Post Request**':

Remember that **Thunder Client** is used for **testing** your code.



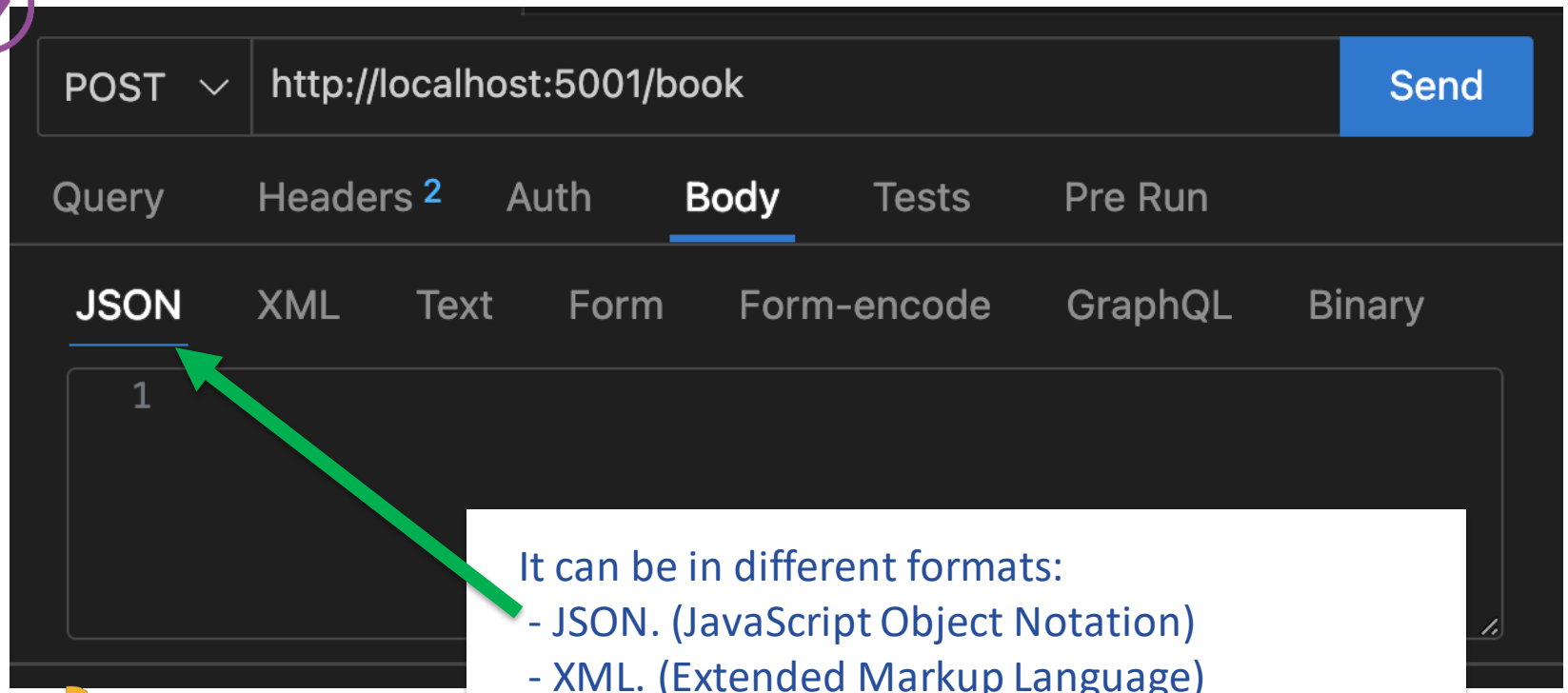
Change this new request into a **POST** request.



CODENATION

Let's take a closer look …

The **body** of this **POST** request is used to pass information in.



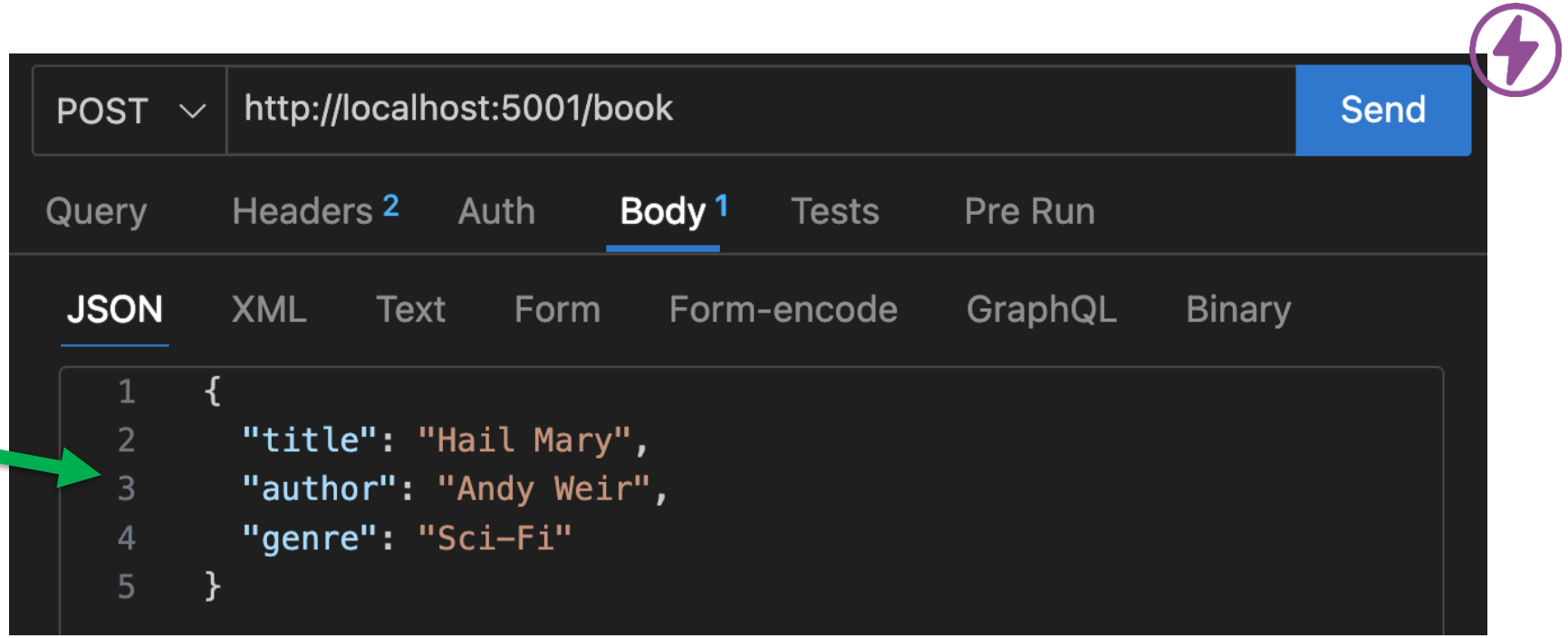
It can be in different formats:
- JSON. (JavaScript Object Notation)
- XML. (Extended Markup Language)
- etc

JSON is now the standard communication method between computers.

It is readable by humans and computers.

Add Your JSON …

Add the details for a new book, in JSON format.



What about the POST?

The code for our **POST** is currently empty so let's add a simple **console.log** in there…

```
28
29  app.post('/book', (request, response) => {
30      console.log(request.body);
31  })
```

Restart the server and **SEND** the **POST** request and we should see the JSON data that we put through the body

```
brianharkins@CN0408 Day1 % node src/server.js
Server is listening on port 5001.
{ title: 'LOTR', author: 'J.R.R Tolkein', genre: 'Fantasy' }
```

CRUD

We can now create a new book object called **newBook**, in the **POST** to add to the books array …

```
29  app.post('/book', (request, response) => {
30      console.log(request.body);
31
32      const newBook = {
33          id: book_id,
34          title: request.body.title,
35          author: request.body.author,
36          genre: request.body.genre
37      };
38
39      books.push(newBook);
40      book_id += 1;
41
42      const successResponse = {
43          message: 'Book successfully added!',
44          book: newBook
45      };
46
47      response.send(successResponse);
48
49  });
```

Add the newBook object. Notice that we are using the request.body for the data…

Add the book to the books array and **Increment** the book id count

Add in a **response** message for this newBook object.

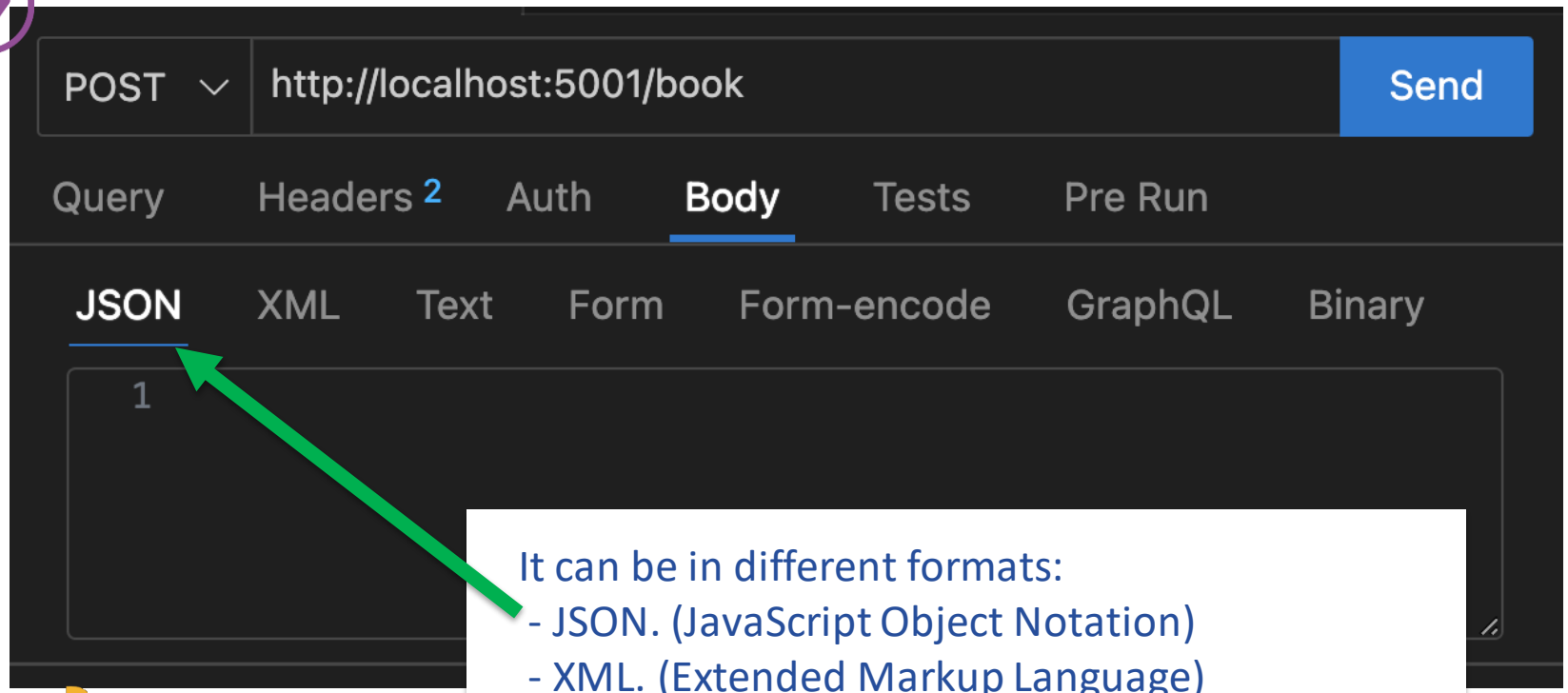
Send the response.

CODE ALONG

CODENATION

Let's take a closer look …

The **body** of this **POST** request is used to pass information in.



It can be in different formats:
- JSON. (JavaScript Object Notation)
- XML. (Extended Markup Language)
- etc

JSON is now the standard communication method between computers.

It is readable by humans and computers.

Let’s Check!

Save and restart the server.

In Thunder Client, run the GET Request.

Then run the POST Request to see the new array.

Status: 200 OK Size: 185 Bytes Time: 4 ms


```
{
  "message": "Books successfully retrieved",
  "book": [
    {
      "id": 1,
      "title": "LOTR",
      "author": "J.R.R Tolkein",
      "genre": "fantasy"
    },
    {
      "id": 2,
      "title": "Hail Mary",
      "author": "Andy Weir",
      "genre": "Sci-Fi"
    }
  ]
}
```

Activity

Add another 2 new books separately to the book array using the **POST** Request in Thunder Client.

What happens to the array if the server is stopped and restarted?

CRUD

- Create ☑
- Read ☑
- Update ☐
- Delete ☐

Our server can now **Create** New Data (POST), **Read** (GET) Data.

But what about **Updating** and **Deleting** Data?

A Starter For Ten …

… we now need to add some code for the Update and Delete code.

Add these two code snippets to get our **Update** (PUT) and **Delete** code

```
47        response.send(successResponse);
48
49    });
50
51    app.put('/book', (request, response) => {
52
53    });
54
55    app.delete('/book', (request, response) => {
56
57    });
58
59    app.listen(5001, () => console.log('Server is listening
```

Let’s write the Delete function first!

(It’s not as tricky as Update)

Add this code to your **Delete** function:

```
app.delete('/book', (request, response) => {
    function findBook(x) {
        return x.title === request.body.title;
    }
    const index = books.findIndex(findBook);

    console.log(index);
});

app.listen(5001, () => console.log('Server is lis
```

The code will find the **index** of an element within the array with the book title that we are looking to delete. **A console log will tell us what it looks like …**

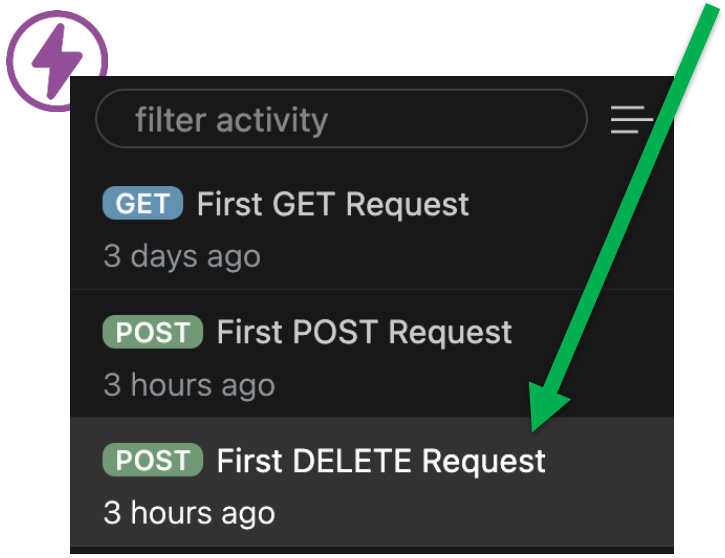
The **findIndex** method will give the **index** of whatever the **findBook** function matches with. If it finds a match, then it returns its **index**, and we can then do something with it.

CODE ALONG

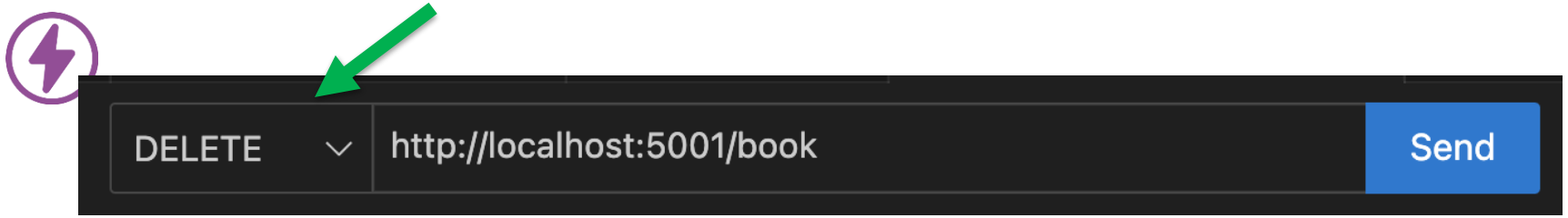
CRUD

In Thunder Client …

… duplicate the '**First POST Request**' and **rename** it to: '**First DELETE Request**':



Change this new request into a **DELETE** request.



CODENATION

Add More JSON …

So that we can test our **Delete** functionality, we need more **book data** …

In Thunder Client, **Add** these two book details using your **POST** request.

(or two of your favourites)

```
{
  "title": "Great Expectations",
  "author": "Charles Dickens",
  "genre": "Novel"
}
```

```
{
  "title": "Hail Mary",
  "author": "Andy Weir",
  "genre": "Sci-Fi"
}
```

Let's Check!

In Thunder Client, run the GET Request.

You should see the **three** books

Status: 200 OK Size: 266 Bytes Time: 2 ms

Response Headers 6 Cookies Results Docs


```
1  {
2    "message": "Books successfully retrieved",
3    "book": [
4      {
5        "id": 1,
6        "title": "LOTR",
7        "author": "J.R.R Tolkein",
8        "genre": "fantasy"
9      },
10     {
11       "id": 2,
12       "title": "Hail Mary",
13       "author": "Andy Weir",
14       "genre": "Sci-Fi"
15     },
16     {
17       "id": 3,
18       "title": "Great Expectations",
19       "author": "Charles Dickens",
20       "genre": "Novel"
21     }
22   ]
23 }
```

Question

Which JavaScript array method would we use to delete an item?

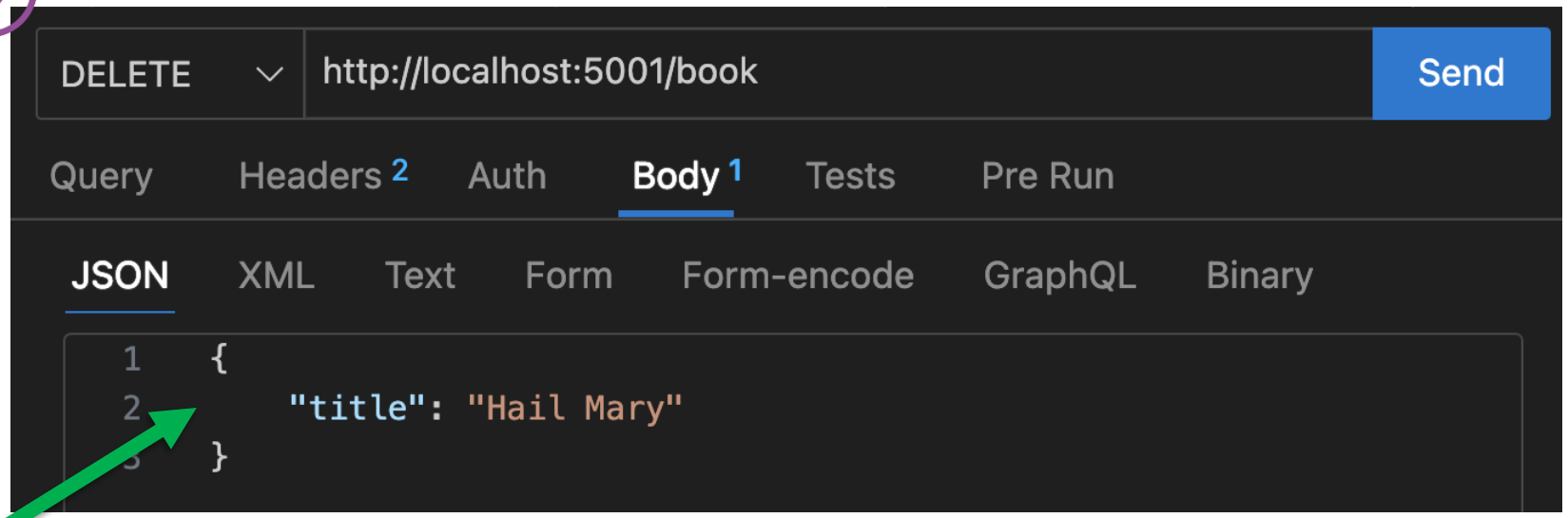
What is the syntax for this method?

Let’s use this method in our **Delete** function …

CRUD

I like to remove it, remove it …

For **Delete**, we only need the **title** in the JSON.



Click Send

Remember that we are **removing** book data by title, so simply enter it into the **JSON text area**.

In the terminal, we should see the **index** of the book to remove – letting us know that it has **found** the book.

What would happen if we spelt it wrong?

CODE ALONG

CODENATION

Splice ‘n Dice

```
55  app.delete('/book', (request, response) => {
56      function findBook(x) {
57          return x.title === request.body.title;
58      }
59      const index = books.findIndex(findBook);
60
61      console.log(index);
62
63      books.splice(index, 1);
64  });
65
66  app.listen(5001, () => console.log('Server is liste
```

Add this **splice** method:

After we have added this splice method (and deleted the book from the array) we need to pop in a **response** – **even if we haven’t found the book …**

Let's Test it …

Restart the server

- Use **GET** to see one book.
- Use **POST** to add two new books.
- Use **Delete** to remove one book (by title)

Useful Responses

Add this **'if else'** code to your **Delete** function:

If the index has been found, do this …

If the index has **not** been found, do this …

```
app.delete('/book', (request, response) => {
    function findBook(x) {
        return x.title === request.body.title;
    }
    const index = books.findIndex(findBook);

    if (index !== -1){
        const successResponse = {
            message: 'Book successfully deleted!',
            book: request.body.title
        };
        books.splice(index, 1);
        response.send(successResponse);
    } else {
        const failureResponse = {
            message: 'Book not found.',
            book: request.body.title
        };
        response.status(404);
        response.send(successResponse);
    }

});
```

Only use splice if there is an index

Let's generate a proper error code, too.

CRUD

- Create ☑
- Read ☑
- Update ☐
- Delete ☑

Our server can now **Create** New Data (POST), **Read** (GET) and **Delete** Data.

But what about **Updating** Data (PUT)?

How Do We Update Data?

We will need to '**find**' a book to **update**, by title…

Copy and paste the **findBook** function from the **Delete** code.

```
app.put('/book', (request, response) => {
    function findBook(x) {
        return x.title === request.body.title;
    }
    const index = books.findIndex(findBook);
});
```

Once we have found the index of the book to **update**, we do a similar thing to the **DELETE** code …

CRUD

Update the Data

```
51  app.put('/book', (request, response) => {
52      function findBook(x) {
53          return x.title === request.body.title;
54      }
55      const index = books.findIndex(findBook);
56
57      if (index !== -1){
58          if (request.body.author){
59              books[index].author = request.body.author;
60          };
61          if (request.body.genre){
62              books[index].genre = request.body.genre;
63          };
64          const successResponse = {
65              message: 'Book successfully updated!',
66              book: request.body.title
67          };
68          response.send(successResponse);
69      }
70  });
```

Update only if there is an index found.

Update if there is an 'author' present.

Update if there is a 'genre' present.

Create and send a **success response**.

CODE ALONG

CODENATION

Don’t Forget the Else …

```
64            const successResponse = {
65                message: 'Book successfully updated!',
66                book: request.body.title
67            };
68            response.send(successResponse);
69        } else {
70            const failureResponse = {
71                message: 'Book not found',
72                book: request.body.title
73            };
74            response.status(404).send(failureResponse);
75        }
76    });
```

Add a failure response message

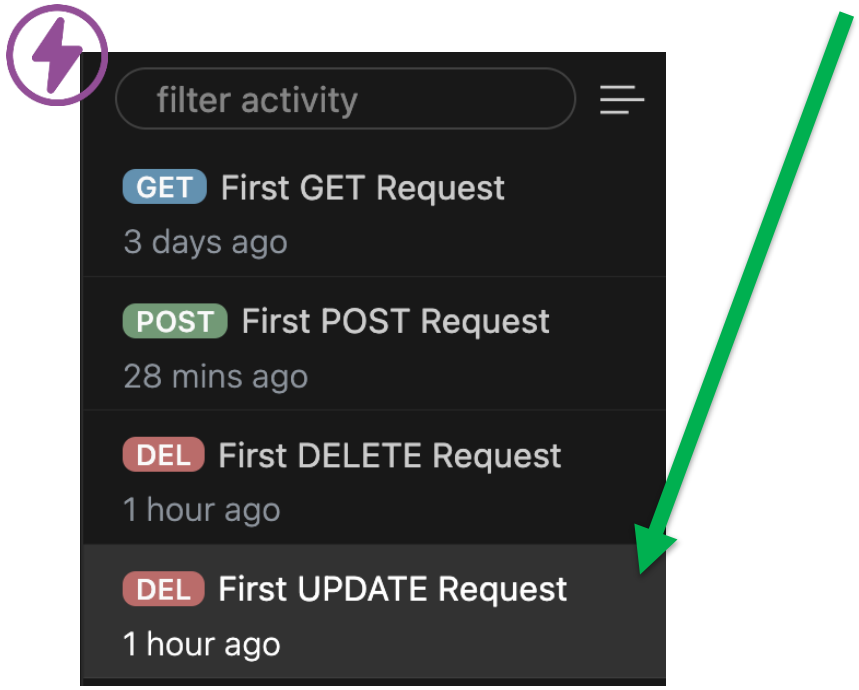
You can chain responses together like this

CODE ALONG

CRUD

In Thunder Client …

… duplicate the '**First DELETE Request**' and **rename** it to: '**First UPDATE Request**':



Change this new request into a **PUT** request.

CODENATION

CRUD

Create ☑
Read ☑
Update ☑
Delete ☑

Well Done!

We now have working code for the 4 CRUD operations.

Learning Objectives

To understand the basics of HTTP Verbs.

To be able to use HTTP Routes in Express.

To be able to use the ThuderClient API client extension.