

COMP4021  
Internet Computing

RESTful APIs

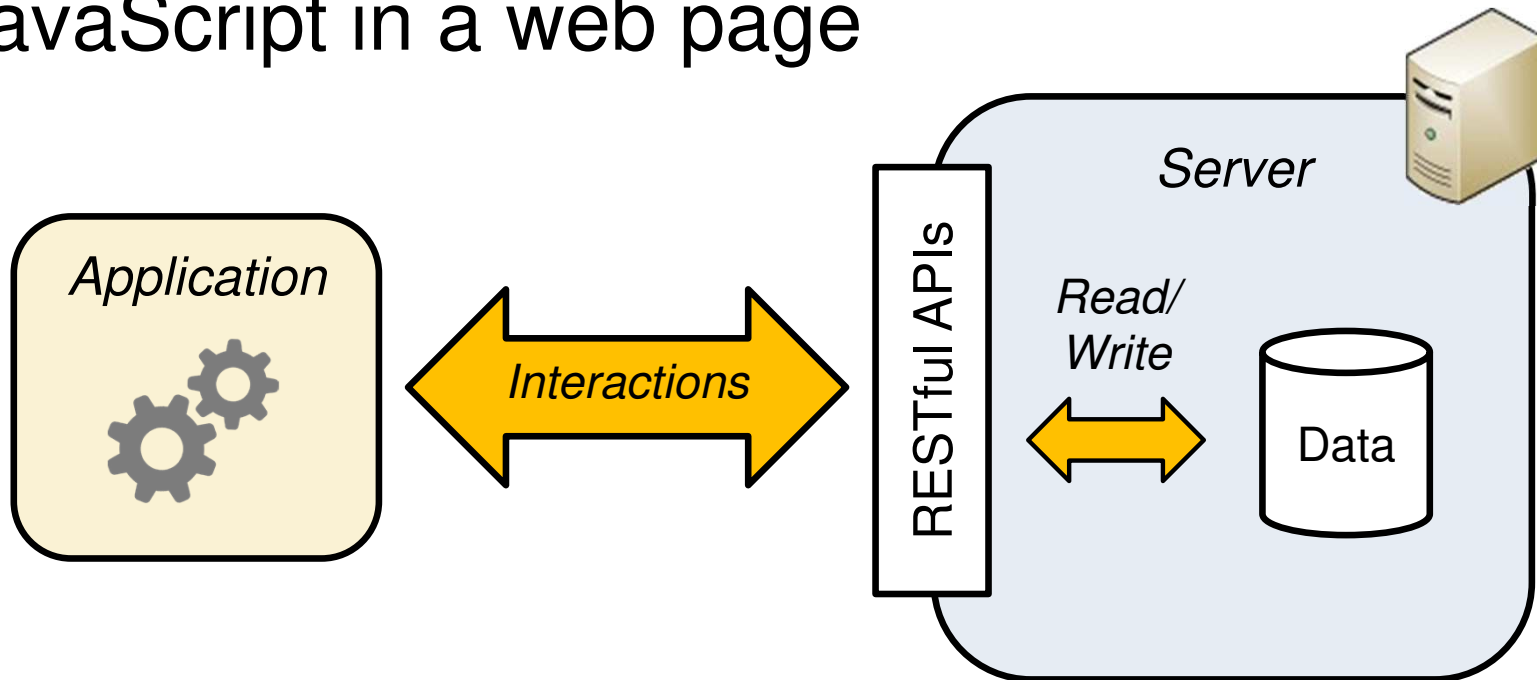
Gibson Lam

# RESTful APIs

- REST stands for **RE**presentative **S**tate **T**ransfer
- It is a set of guidelines that help you create collections of web server endpoints for managing 'resources'
- These endpoints, generally called RESTful APIs, allow developers to use them easily over HTTP

# Overview

- RESTful APIs are usually used by programmers from other applications, such as using JavaScript in a web page



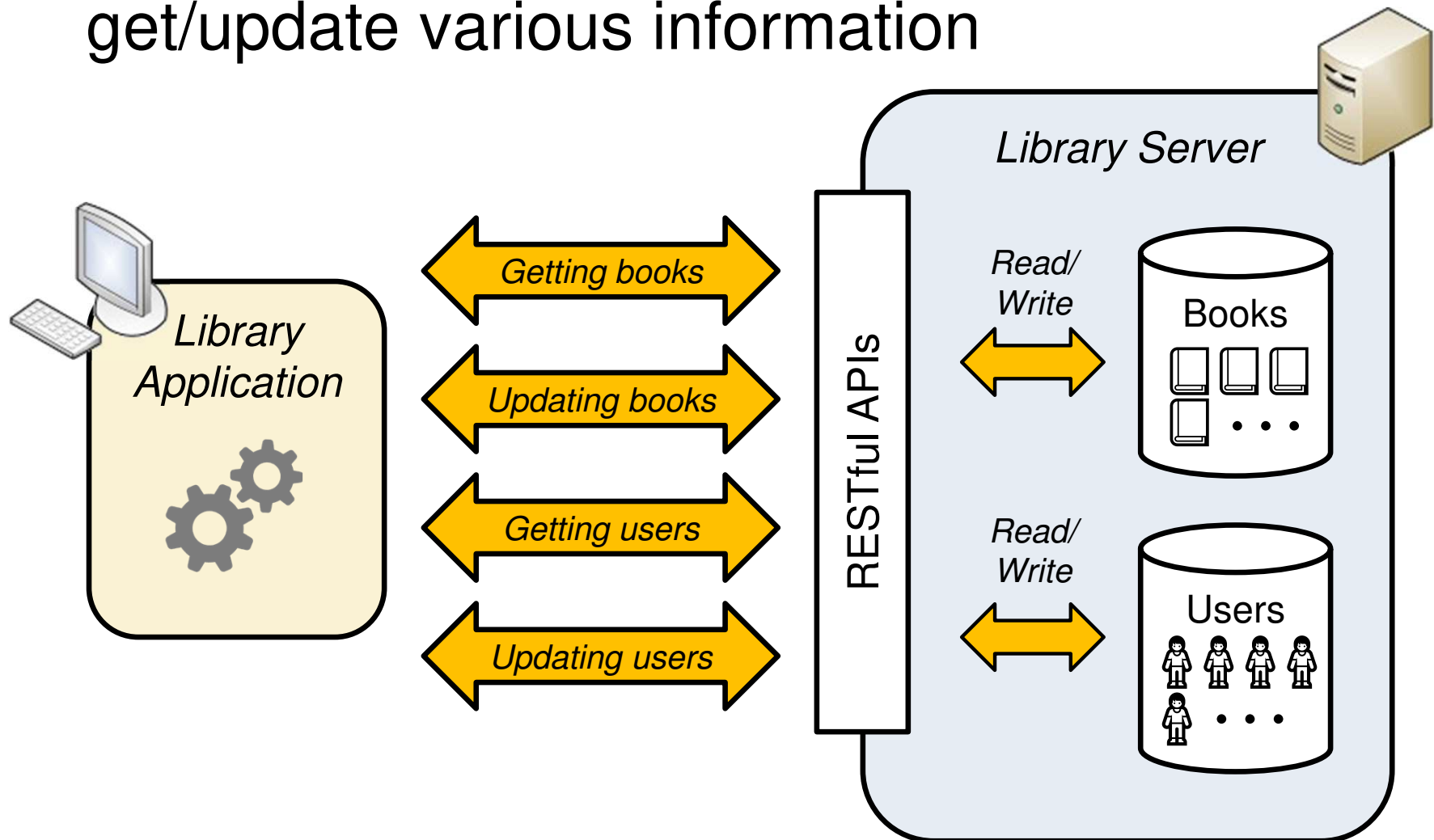
- You don't usually access the APIs as web pages

# Managing 'Resources'

- RESTful APIs are always about managing 'resources', which are typically data stored on a server
  - For example, managing user records and book records stored in a library system
- They don't work on multiple things at the same time and don't work on relationships
  - For example, they don't do things like enrolling students into a course

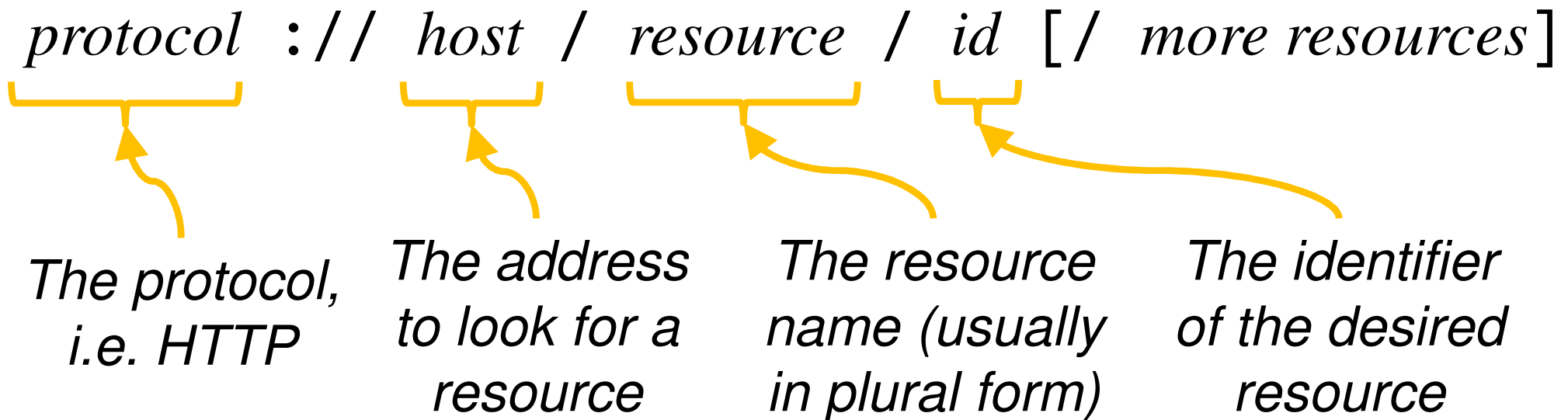
# An Example System

- Here is an example library application that communicates with the library server to get/update various information



# The API Interface

- RESTful APIs use a 'uniform interface', when accessing resources
- When the APIs are used over HTTP, the interface is simply a URL
- It is divided into different parts, like this:



# Example Interfaces

- Getting a book with an id of 2:

`http://books.com/books/2`

- Getting the computer information of a room:

`http://ustrooms.edu/rooms/2465/  
computers/1`

- Getting all students of the CSE major:

`http://mystudents.edu/majors/cse/  
students/`

# Data Transfer

- When getting resources from a RESTful server, they are typically represented by JSON, as shown in this example:

`http://books.com/books/2` 

- However, you can also return things in any format you want

```
{  
  "id":2,  
  "title":"The Snow Queen",  
  "author":"H. C. Andersen",  
  "Year":1944  
  ...  
}
```



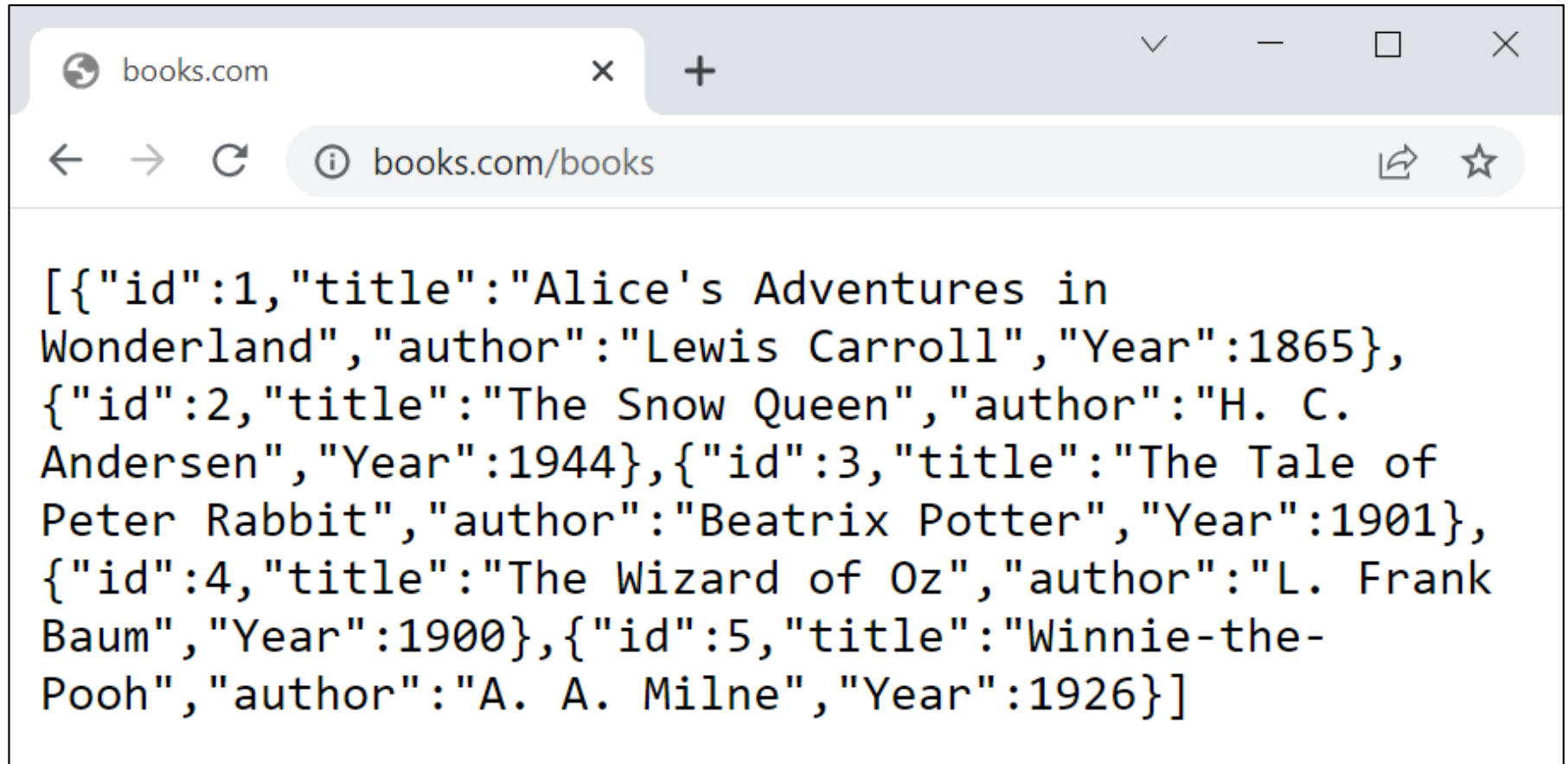
# Using REST in an Express Server

- You can make a RESTful server using Express endpoints
- For example, here is an endpoint to return all books in a library system:

```
app.get("/books", (req, res) => {  
    ...Retrieve the books...  
    res.json(...all books...);  
});
```

# Showing an Example Output


- Since it is a GET request, you can see the output of the endpoint using a browser, like this:



# Getting Individual Resources

- When getting an individual resource using a resource id, you can set up the Express endpoint using a colon ':'

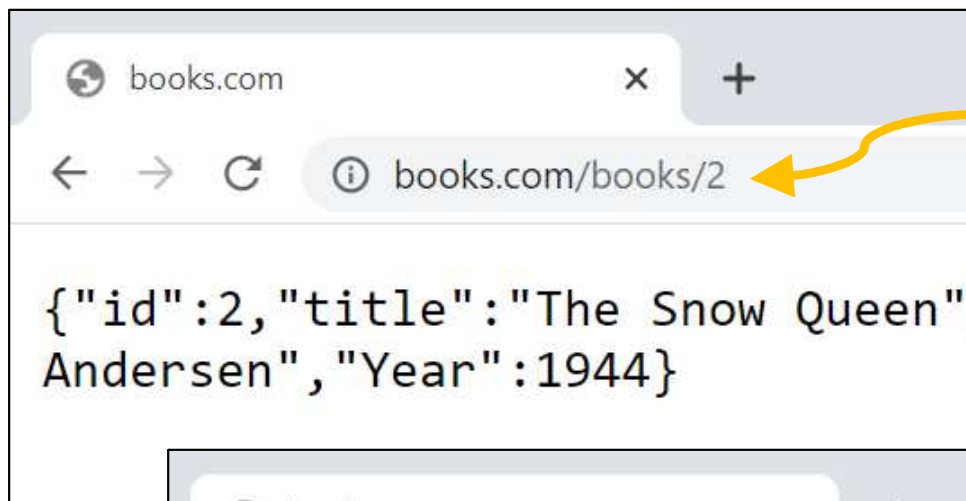
*Getting the id as a parameter from the URL*



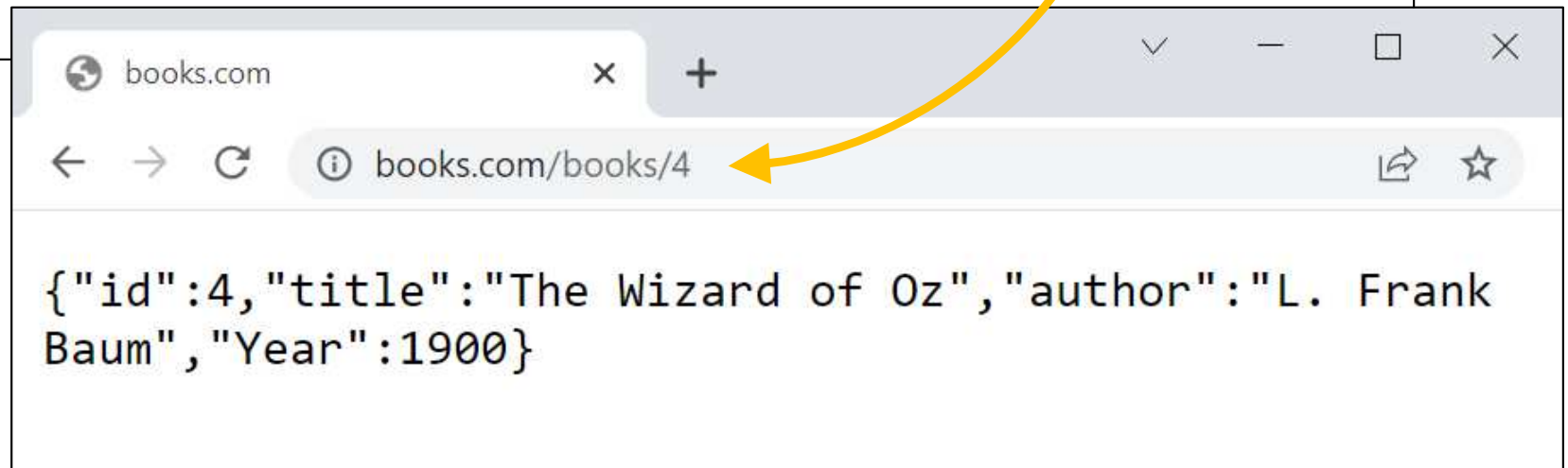
```
app.get("/books/ :bookId", (req, res) => {  
  const bookId = req.params.bookId;  
  ...Retrieve the book using the id...  
  res.json(...the requested book...);  
});
```

# Getting Example Books

- For example, individual books in a library system can be shown in a browser:



*Putting the id at the end of the URL*



# Doing Things With the Resources

- In the previous examples, we only read the resources, you can update them if you want to
- You can do one of the following four operations using RESTful APIs:
  - Reading resources  
(which we have done already)
  - Creating new resources
  - Updating resources, and
  - Deleting resources

# Using HTTP Methods

- A typical web server uses different URLs to serve different purposes
- RESTful APIs work differently; they use HTTP methods to map to the type of things that they perform:
  - The GET method                      for reading
  - The POST method                     for creating
  - The PUT method                      for updating
  - The DELETE method                for deleting

# HTTP Methods and URL

- Even if the same URL is given, a RESTful server may apply two different operations
- For example:
  - If you send a GET request to:  
`http://books.com/books/12`  
you will retrieve the book with an id of 12
  - If you send a DELETE request to the same URL, you will delete the book!

# More Endpoints in Express

- In the next few slides, we will quickly show some additional example endpoints to complete the library's RESTful APIs
- First, you can create new books, e.g.:

*Using the POST request*



```
app.post("/books", (req, res) => {
```

```
  const book = req.body ;
```

```
    ...Create the new book...
```

```
});
```

*The new book  
is sent as  
JSON data*





# Updating Resources

- You can also update the books, e.g.:

*This uses the PUT request*

```
app.put("/books/:bookId", (req, res) => {  
  const bookId = req.params.bookId ;  
  const book = req.body ;  
  ...Update the book with bookId...  
});
```

*The book to  
be updated*

# Deleting Resources

- Finally you can delete any books, e.g.:

*Using the DELETE request*

```
app.delete("/books/:bookId",  
          (req, res) => {  
    const bookId = req.params.bookId ;  
    ...Delete the book with bookId...  
  });
```



*The book to  
be deleted*

# Using HTTP Status Codes

- RESTful APIs use status codes to represent their results
- Here are some examples:
  - 200 OK e.g. successful GET, PUT or DELETE
  - 201 Created e.g. successful POST
  - 400 Bad Request e.g. bad input
  - 404 Not Found e.g. invalid resource id

# Example of Using Status Code

- For example, if you successfully delete a book, you will return 200; or 404 (not found) otherwise


```
app.delete("/books/:bookId", (req, res) => {  
  const bookId = req.params.bookId;  
  if (... bookId exists?...) {  
    ...Delete the book with bookId...  
    res.sendStatus(200);  Successfully  
delete the book  
  }  
  else  
    res.sendStatus(404);  Book not found  
});
```

# Checking Status Using fetch()

- Back at the browser, you can check the status code in the response object when using `fetch()`, as shown below:

```
fetch("/books")  
  .then((response) => {  
    if (response.status == 200)  
      return response.json();  
    else  
      alert("Failed to get the books!");  
  })  
  ...
```

*Checking status code*



# A Library APIs Example

- An example page has been created to test the library's APIs that we have shown so far:

## Library Books RESTful APIs

Get All Books	Get a Book	Create a New Book	Update a Book	Delete a Book
<input type="button" value="Execute"/> Result: <input type="text"/>	Book id: <input type="text"/> <input type="button" value="Execute"/> Result: <input type="text"/>	Book title: <input type="text"/> Book author: <input type="text"/> Book year: <input type="text"/> <input type="button" value="Execute"/>	Book id: <input type="text"/> Book title: <input type="text"/> Book author: <input type="text"/> Book year: <input type="text"/> <input type="button" value="Execute"/>	Book id: <input type="text"/> <input type="button" value="Execute"/>

*The five RESTful APIs' endpoints can be tested using these HTML inputs*