Lab 5 – REST APIs

# Introduction

Welcome to Week 5! …we’re almost there!

This is the last lab of the unit (next week’s labs will be for assignment support). Over the previous weeks you have developed a good understanding of web fundamentals. You can now: create HTML pages, style them with CSS, make them work across devices with Flexbox, and now you can validate form input with Regular Expressions. The last piece of the puzzle is interacting with servers and other devices through APIs. Further to this, you can make API calls within your web pages to create an element of interactivity. This is how all modern websites work and is the difference between a static website and a rich web application.

# Learning Objectives

# To understand what REST APIs are, how they work, and how to interact with them

* To practice using Postman for testing API specifications
* To practice interacting with REST APIs through JavaScript

# Understanding REST APIs

## Exercise 1: Understanding REST API specifications

Familiarise yourself with how RESTful APIs work. The lecture is a good starting point, but do some research until you fully understand the concepts. There are many tutorials and explanations on the web. For example:

* https://restfulapi.net/
* https://www.redhat.com/en/topics/api/what-is-a-rest-api
* https://youtu.be/7YcW25PHnAA

## Exercise 2: Running the server

This exercise will only work on the lab machines. If you are running on your own device, you will first need to install NodeJS from (<https://nodejs.org/en/>). You will need NodeJS for later units also.

1. Unzip the directory to somewhere on your machine
2. Copy the path to this directory by clicking on the address bar and hitting ctrl + c
3. Open your command line application from the start menu.
4. The command line application will open in your home, we can navigate to the project directory by typing “cd <paste your filepath>” (cd stands for current directory)

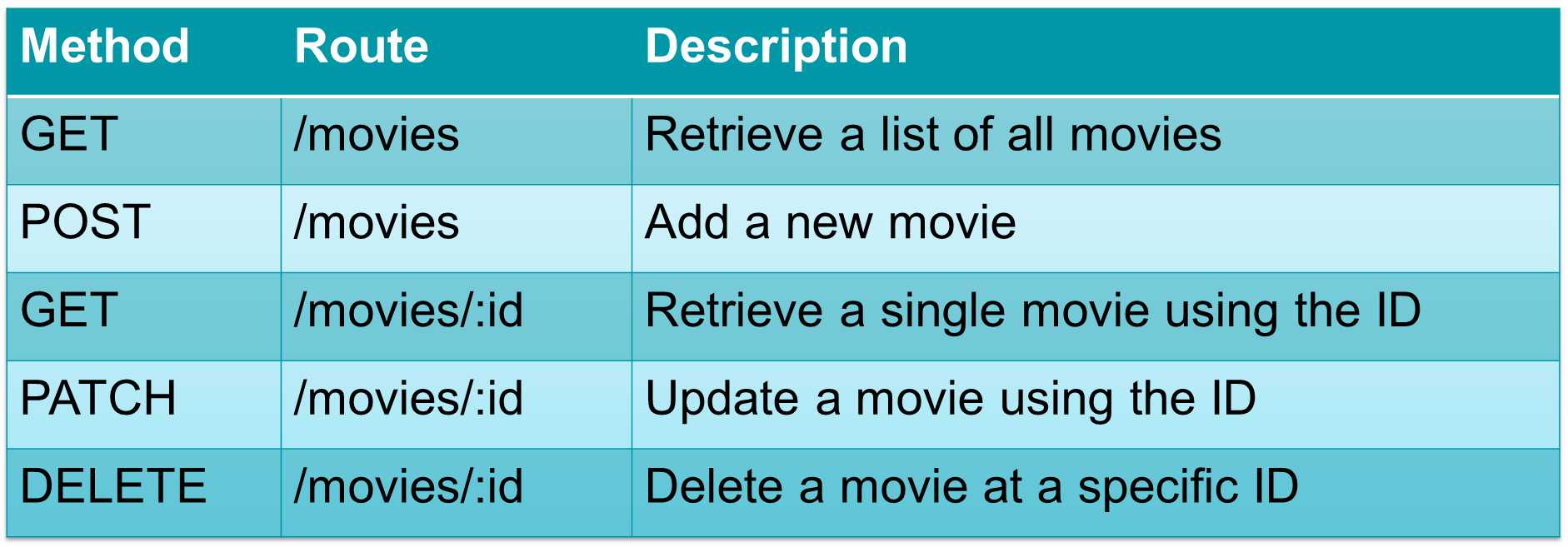


1. You should now see that you are in the directory where you unzipped the server code. If so, then run ‘npm install’ (a directory called ‘node\_modules’ should appear with all of the dependencies).
2. Run ‘npm start’ to run the server. The terminal should print ‘Listening on port 3000…’

## Exercise 3: Testing our understanding with Postman

Again, you will need to install Postman if you are working on your own device. Postman can be installed from <https://postman.com/>

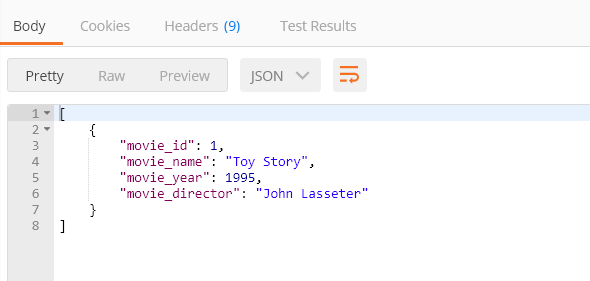
Here is the API specification for the MovieDB API.



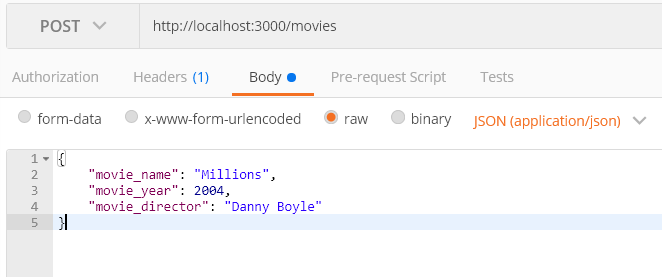
1. Open Postman from your start menu
2. Along the top of Postman, we have all the options necessary for sending HTTP requests. The responses will then appear at the bottom of the application window. Let’s start by retrieving a list of all the movies in the DB (e.g., a GET request to /movies). The server is hosted on our local machine (that command line window) on port 3000. Therefore, we need to specify the URL and port in our request. Copy the below to do this, and click the “Send” button.



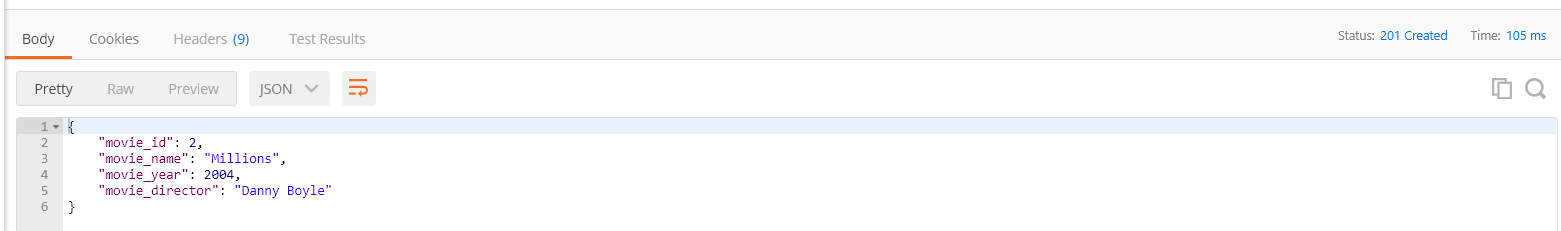
1. In the response window, we can see that an array of JSON objects has been returned. However, we currently only have one movie in the DB.



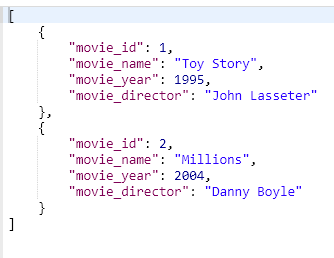
1. Let’s add a new movie. For this we will need to send a POST request to /movies. Select POST from the list of methods to send.
2. The POST request also requires a body to be sent with the data for the new movie. Navigate to the body tab and write copy in the JSON object below. Make sure to also check that you are sending “raw” JSON by selecting these options (see the screenshot)



1. Note that the movie\_name and movie\_director fields are strings, whereas the movie\_year is a number. Click “Send” again and look at the response. The server has responded with a “201 Created” meaning that the movie was successfully added (the screenshot is small, but see the red box for an indication of where to check in Postman)



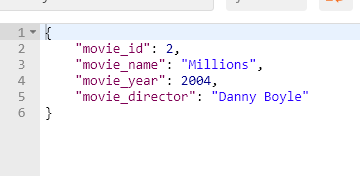
1. Now run the GET request again, can you see the movie in the list?



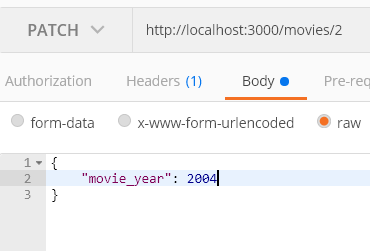
1. Try adding another movie, but with bad data. Try different data types or a year before 1900. Can you break the server, or does it handle all bad data gracefully? (If you can break it, then it’s because of Ash’s shoddy coding).
2. We can also retrieve a single movie by specifying the ID as a path parameter. Send a GET request to /movies/2 to see just your newly added movie.



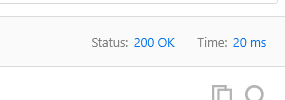
1. Note the response isn’t an array this time. It’s just the single movie



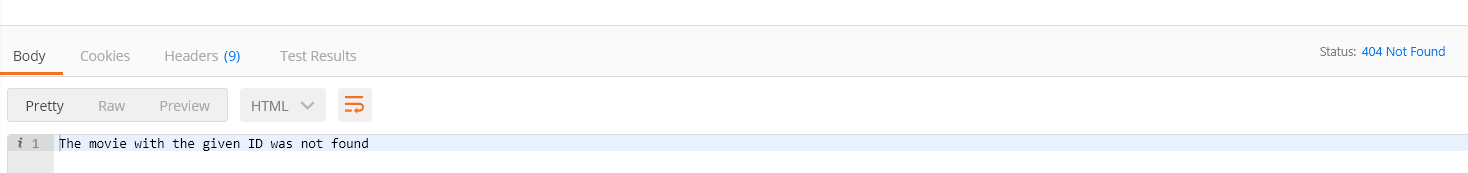
1. We can update the movie using the PATCH method. Update the method to PATCH and keep the URL as /movies/2. Let’s update the year to 2021 in the body of the request



1. There is no body to the response, but we know it was successful because the server returned a “200 OK” response.



1. Double check that the update worked by sending a GET request
2. Finally, we can delete the movie. Change the method to DELETE and keep the URL as /movies/2. Send the request and check that it was successful with a “200 OK” response
3. Test that it has been deleted with a GET request. Note that now we have a “404 Not found” response and the server returns a handy error message



# Interacting with REST APIs using JS

## Exercise 4: Create a HTML form

The next set of exercises replicate the code that was demonstrated in the lecture.

Start by creating a form for adding a Movie to the API. The form should have three inputs and a submit button. Make sure to add all required ID’s so that we can reference in JS

## Exercise 5: Validate the form

Add an event listener which is triggered when the forms submit button is clicked. The function should validate the input using what you learnt in the lab last week. Your validation should check the following:

1. That all fields have been entered
2. All fields should be the right data type
3. The year should be greater than 1900, and less that or equal to the current year (check W3Schools for finding the current year)
4. The movie name and director should be less than or equal to 32 characters

## Exercise 6: Send a POST request to the server

If the form passes your validation checks, you can send to the server. Use the lecture slides to help you. Test that your application is working using Postman.

## Exercise 7: Send a GET request to the server

Using the lecture slides to help you, add a table to your HTML page that is automatically populated with the MovieDB list of movies when the page loads. Edit your POST request so that the table is updated when a new movie is added.

## Exercise 8: Pretty error handling

Add a stylesheet to your web page and create two class styles: one for errors and one for success. Add an empty <div> tag to your web page for providing feedback to users. Using the lecture slides to help you, can you provide feedback to the user on whether their form submission was successful or not. Use the lecture slides to add your class styles depending on the message displayed.

## Exercise 9: PATCH and DELETE

The previous exercises were a lot of copying and pasting from the lecture slides. This exercise is a true test of understanding. Can you add functionality into your application for updating and deleting records in your table and the MovieDB API? (do the delete first as it is easier)

## Exercise 10: Your assignment!

Use Postman to test out the assignment API. The assignment specification has details of the two endpoints that are to be implemented in the assignment. Familiarise yourself with both endpoints.

If there is time, start implementing the API calls within your assignment project.