CS480 Fall24 - Hayes - Project 2, API

2024-11-18 First draft

|  |
| --- |
| Submission instructions and testing information will be updated later.  I may add additional endpoints before submission deadline. Any additional requirements will take only a few minutes, assuming you have understood the prior work you've done.  Due date: December 4, 11:59pm |

[**Description 2**](#_yfks8x4mbrd)

[**Some learning and review before starting 2**](#_fv5xtr4l43e1)

[**Required MongoDB database and collection 2**](#_yoa80qclpiqq)

[**Required Endpoints 4**](#_uld8iyy9slp8)

[**Response requirements 7**](#_ph7buk1js0nl)

[**Infrastructure requirements 8**](#_3dtgtwj6ddcy)

[**Coding Documentation 9**](#_k2ffuys6ean6)

[**Coding hints (?) 10**](#_glnoowuacbb2)

[**Testing 11**](#_sn3oak3h0y5)

[**Submission 12**](#_my1jkxud6vg7)

## Description

For this project, you'll expand upon the API you built previously.

## Some learning and review before starting

What is a REST API?

<https://www.ibm.com/topics/rest-apis>

You'll need to know the basics of creating API endpoints,   
  
RESTful web API design  
<https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>   
(focus on <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design#define-api-operations-in-terms-of-http-methods> )  
  
<https://gearheart.io/blog/restful-api-design-best-practices/>

Review the following pages of the Express JS documentation:

* <https://expressjs.com/en/starter/hello-world.html>
* <https://expressjs.com/en/starter/basic-routing.html>
* <https://expressjs.com/en/guide/routing.html>
* <https://expressjs.com/en/4x/api.html#req.query>

## Required MongoDB database and collection

Create a new database named **cs480-project2**

Add a new collection named **colors**

Add the 6 color documents found at <https://www.sitepoint.com/colors-json-example/#:~:text=%7B%0A%20%20%20%20%20%20%22color%22%3A%20%22black%22%2C%0A%20%20%20%20%20%20%22category%22%3A%20%22hue%22%2C%0A%20%20%20%20%20%20%22type%22%3A%20%22primary%22%2C%0A%20%20%20%20%20%20%22code%22%3A%20%7B%0A%20%20%20%20%20%20%20%20%22rgba%22%3A%20%5B255%2C255%2C255%2C1%5D%2C%0A%20%20%20%20%20%20%20%20%22hex%22%3A%20%22%23000%22%0A%20%20%20%20%20%20%7D%0A%20%20%20%20%7D>

## Required Endpoints

All endpoints start with /**api/v1**

|  |  |
| --- | --- |
| **/actors**  **/films**  **/customers**  **/stores** | these endpoints return all columns for the relevant entity/table  for films, optional query parameter (named query), conducting a case-insensitive search of titles that include the value  e.g. /films?query=Shawshank |
| **/actors/<id>**  **/films/<id>**  **/stores/<id>**  **/customers/<id>** | these endpoints return all columns for the row with primary id equal to the route parameter <id> |
| **/actors/<id>/films** | return all rows from film for the specified actor |
| **/films/<id>/actors** | return all actors for the relevant film |
| **/films/<id>/detail** | return row from view film\_list |
| **/customers/<id>/detail** | return row from view customer\_list |
| **/actors/<id>/detail** | return row from view actor\_info |
| **/inventory-in-stock/<film\_id>/<store\_id>** | return the inventory ids (the result set of calling stored procedure film\_in\_stock) <https://dev.mysql.com/doc/sakila/en/sakila-structure-procedures-film_in_stock.html> |
| **/movies** | returns all fields for movies from mongo sample\_mflix database; limit to 10 documents  optional query parameters to filter on any column  e.g.,  /movies?genre=Drama  /movies?genre=Drama&year=1980 /movies?genre=Drama&year=1980&director=John Ford |
| **/colors** | GET - return all fields for all documents  POST - insert a new item, return result |
| **/colors/<id>** | GET - return all fields for specified document  PUT - update specified document, return result DELETE - delete specified document, return result |

"Why are endpoints plural when the table names are singular?"

Well, sometimes we just have to follow the commonly-accepted standards.

<https://gearheart.io/blog/restful-api-design-best-practices/>

## 

## Response requirements

For this API, we're **not** going to include metadata in the response text; just return the array of data.

For this API, we're not going to deliver any helpful error messages (though you could choose to do that in a production implementation.) In the event of an error, just return an array with one text element: "An error has occurred."

So, most of your API endpoints will return one of the following:

1. an object representing a single row or document,
2. an array of objects representing rows or documents,
3. an empty array (when no items matched the query), or
4. ["An error has occurred."]

NOTE: some operations (the response from the stored procedure call or the Mongo post, put, delete) will return additional data. Just return whatever comes back from those operations - I don't need you to parse out specific elements.

**CORS access**

<https://aws.amazon.com/what-is/cross-origin-resource-sharing/>

Your HTTP response header must also include   
Access-Control-Allow-Origin: \*

## 

## Infrastructure requirements

Your API should be hosted on a new GCP VM.

I recommend you build and test on Replit, then move the needed files to your VM.

You'll need to install Node.js, then "run" the Node project to install all the dependencies.

## Coding Documentation

You need to use two **node.js packages** for connecting with your databases.

**mysql2**

<https://www.npmjs.com/package/mysql2>

What you need was covered in the prior node/rest api tutorial, but you should spend a few minutes reviewing the section on performing queries

<https://www.npmjs.com/package/mysql#performing-queries>

This explains how to avoid sql injection attacks by escaping user-supplied values and/or using the ? or ?? as placeholders (which also escapes the values.)

**mongodb**

<https://www.npmjs.com/package/mongodb>

On the documentation page, review the Quick Start section and the Quick Reference section which has all (most?) of what you need.  
  
This one is also useful  
<https://www.mongodb.com/docs/drivers/node/current/usage-examples/find/>

You may also need to review the express.js documentation

<https://expressjs.com/>

particularly the routing pages

<https://expressjs.com/en/starter/basic-routing.html>

<https://expressjs.com/en/guide/routing.html>

## Coding hints (?)

You're building one web server that will serve data from both your mySQL and MongoDB instances.

The structure of your code might look something like:

// require modules and other setup commands

// connection variables

// endpoints

// function that formats the data before sending

// start listening