79/80

Complete the following Hands-On Test and submit in your **AllCoursework** repository in the /HandsOnTests/HOT3/ folder. Be sure to include your .env file so your instructor can connect to your MongoDB cluster.

**Best Practices**

To receive full credit for this assignment, you must consistently follow all of the below coding standards.

* Indent code using **2 spaces** per level, as per [industry standards.](https://google.github.io/styleguide/jsguide.html#formatting-block-indentation)
* Name all variables and functions using **camelCase.**
* Avoid declaring **global variables** where possible, prefer **local variables** and **function parameters** instead.
* **Don't use var. You will lose 5 points for every use of the var keyword!**
* Use **const** to define variables whenever possible. For example:

|  |
| --- |
| const milesDriven = req.body.milesDriven; |

* Use **let** to define variables only when **const** isn't possible. For example:

|  |
| --- |
| let x = 3; x += 5; |

* Prefer **arrow functions (=>)** for anonymous functions, over the traditional **function declaration** syntax. For example:

|  |
| --- |
| (x, y) => x + y |

* The **function declaration** syntax is permitted for creating **named functions.** For example:

|  |
| --- |
| function add(x, y) { return x + y; } |

* Use [**template strings**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals) instead of string concatenation. For example:

|  |
| --- |
| const fullName = `${firstName} ${lastName}`; |

* Use [**async-await**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function) to support asynchronous database operations.

|  |
| --- |
| async function findAllPets() {  const db = await connect();  const pets = await db.collection('pets').find({}).toArray();  return pets; } |

* Use [**try-catch**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch) to handle exceptions and promise rejections.

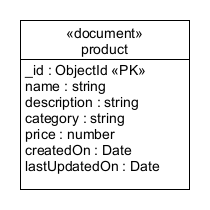
|  |
| --- |
| router.get('/list', async (req, res, next) => {  try {  const pets = await dbModule.findAllPets();  res.json(pets);  } catch (err) {  next(err);  } }); |

**Project Overview**

* Build a full stack CRUD Web Application using MERN
* The main server code for this project must be placed in **server.js**
* Use **config or .env** as demonstrated on the lecture repo and the issue-tracker projects to store/retrieve db connection string information.
* Use **dotenv** as demonstrated to set the **DEBUG** and **PORT** variables.
* Use an Atlas MongoDB database for this project.
* Use [**Postman**](https://www.postman.com/downloads/) to test these routes.

**Products API**

In this assignment you will start creating an API for an ecommerce store. The CRUD routes described below implement the basic functionality needed to view and administer the product catalog.

A UML diagram of the product document is shown on the right.

Implement this API as a route module, located at **/routes/api/product.js**

* Use **async-await** to implement asynchronous database operations.
* Use **joi** to validate the request data.
* Send all data and messages back as **JSON.**
* Use the **200, 400, 404,** and **500** status codes appropriately.
* Set localhost port to 2023 **- 5pts**
* **All** URLs on this test must match the route provided. **5pts**
* Open your database to access from ALL IP Addresses (you can delete after you receive your grade) **-5pts**
* Make sure the .env file is not listed on your .gitignore -**5pts**
* Put all of your database code/logic in the **database.js** as demonstrated in class. **-5pts**
* Use **try-catch** to handle all database errors and promise rejections. **-5pts**
* Build a working frontend with REACT.JS that performs READ operations on all products. **-5pts**

**GET /api/products** \_\_\_\_\_/ **5pts**

* Returns all of the products in the database as a **JSON array.**
* *You do not need to implement searching, filtering, sorting, or paging for this exam.*

**GET /api/products/:productId** \_\_\_\_\_/ **5pts**

* Returns a single product from the database as a **JSON object.**
* Find the product based on the provided **ID.**
* If the ID is invalid or the product is not found, return a **404** response.

**GET /api/products/name/:productName** \_\_\_\_\_/ **5pts**

* Returns a single product from the database as a **JSON object.**
* Find the product based on the provided **name.**
* If the product is not found, return a **404** response.

**POST /api/products** \_\_\_\_**\_/ 5pts -1pt**

* **Inserts** a new product into the database.
* Returns a JSON object containing a **message** and the new **productId.**
* Accept the following fields via the **body** of the request:
  + name : string
  + description : string
  + category : string
  + price : number
* Validate the body of the request using **Joi.** \_\_\_\_\_/ **5pts**

**PATCH /api/products/:productId** \_\_\_\_\_/ **5pts**

* **Updates** an existing product in the database.
* Adds the field lastUpdatedOn to the object. **-5pts**
* Returns a JSON object containing a **message** and the **productId.**
* If the ID is invalid or the product is not found, return a **404** response.
* Accept the following fields via the **body** of the request:
  + name : string
  + description : string
  + category : string
  + price : number
* Validate the body of the request using **Joi.** \_\_\_\_\_/ **5pts**

**DELETE /api/products/:productId** \_\_\_\_\_/ **5pts**

* **Deletes** an existing product from the database.
* Returns a JSON object containing a **message** and the **productId.**
* If the ID is invalid or the product is not found, return a **404** response.