**77/80**

**Testing environment/rules**

* For this test, you may use your notes, textbook, reading materials on Perusall, your completed lab assignments, and the internet as a resource for this Hands-On test.
* This test is being administered as a **take-home test,** and is due back **Monday @ Noon**
* The test must be submitted via a GitHub repository, and you must give collaborator/admin access to your instructor.
* This is an individual project, you are expected to complete all work by yourself without collaboration/input from other students.
* All code must be your own original work. Code found online, on websites like Stack Overflow, Medium, etc. is strictly prohibited. *(Example code found on* [*developer.mozilla.org*](https://developer.mozilla.org) *is permitted though.)*

**How to submit your code**

As we have done with previousassignments, you need to create a new GitHub repository for this project.

Details are listed below:

* Name the repository **"awd1111-exam-2"**
* Make the repository **"private"**
* Add a **README** file
* Add a **.gitignore** file, using the **"Node"** template **(Don't forget this!)**
* Add a **LICENSE** file, using the **"MIT License"**
* Add **paulsmithkc** as a **Collaborator/Admin**

**Create a package.json file for the project**

Open a terminal at the **root** of the repository, and run the **"npm init"** command. *(Or "npm init -y" if you wish.)*

Then open the **package.json** file, make the following edits, and save them.

* **"name": "awd1111-exam-2"**
* **"version": "1.0.0"**
* **"main": "server.js"**
* Add a **script** to start the production server… **"start": "node server.js"**
* Add a **script** to start the development server… **"start-dev": "nodemon -r dotenv/config server.js"**

**Install dependencies**

For this project, you will need to install and use the following dependencies:

**config, debug, dotenv, express, joi, mongodb**

**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.

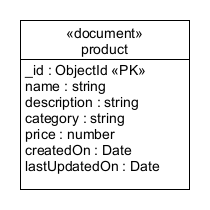
|  |
| --- |
| router.get('/list', async (req, res, next) => {  try {  const pets = await dbModule.findAllPets();  res.json(pets);  } catch (err) {  next(err);  } }); |

**Project Overview**

* Build a small REST API using **Node, Express, and MongoDB.**
* The main server code for this project must be placed in **server.js**
* Put all of your database code/logic in the **database.js** as demonstrated in class.
* Use **config** as demonstrated on the pet-adoption and issue-tracker projects to store/retrieve configuration for both **http** and **db** keys.
* Use **dotenv** as demonstrated to set the **DEBUG** and **PORT** variables.
* Use your local or Atlast 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 **try-catch** to handle database errors and promise rejections.
* 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.

**GET /api/product/list** \_\_\_\_\_/ 10pts

* 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/product/id/:productId** \_\_-3\_\_/ 10pts

* 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/product/name/:productName** \_\_\_\_\_/ 10pts

* 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.

**PUT /api/product/new**

* **Inserts** a new product into the database. \_\_\_\_\_/ 10pts
* 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.** \_\_\_\_\_/ 10pts

**PUT /api/product/:productId**

* **Updates** an existing product in the database. \_\_\_\_\_/ 10pts
* 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.** \_\_\_\_\_/ 10pts

**DELETE /api/product/:productId** \_\_\_\_\_/ 10pts

* **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.