**100%**

**Testing environment/rules**

* For this test, you may use your notes, textbook, reading materials, your completed lab assignments, and the internet as a resource for this Hands-On test.
* The test must be submitted to your **all-coursework/HandsOnTests/HOT1/** GitHub repository
* 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.

**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-1"**
* **"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:

* debug
* dotenv
* express

**Project Overview**

* Build a small web API using **Node** and **Express** which implements the following routes.
* The main server code for this project must be placed in **server.js**
* *For this test, you may choose to implement all of the routes in* ***server.js*** *or break them up into* ***route modules*** *as demonstrated in class.*
* *For this test, you are permitted to borrow some code/logic from your* ***Lab 01-01 or Lab 01-02*** *assignment for this.*
* *Use* [***Postman***](https://www.postman.com/downloads/) *to test these routes.*

**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 [**Number.toFixed()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toFixed) to format numbers for display. For example:

|  |
| --- |
| const number = 4.123; const formattedNumber = number.toFixed(2); |

* Use [**String.padStart()**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/padStart) to pad numbers with zeros or spaces. For example:

|  |
| --- |
| const number = 5; const formattedNumber = number.toFixed(0).padStart(3, '0'); |

**Implement the following routes.**

* **All routes** must match the exact specified path. **5pts**
* Localhost should be on port 5010 - **5pts**
* req.body variable names must match names provided on rubric - **5pts**

**POST /api/mpg/calc (15pts)**

* The following data must be provided as the **body** of the request, and read from **req.body:**
  + milesDriven (sent as a string) \_\_\_\_ / 2.5pts
  + gallonsUsed (sent as a string) \_\_\_\_ / 2.5pts
* Parse and validate the provided input.
  + Use **parseFloat()** to parse a string into a number. (This returns NaN if it was not a valid number.)
  + If milesDriven **is NaN or ≤ 0,** send back an error message with status code 400. \_\_\_\_ / 2.5pts
  + If gallonsUsed **is NaN or ≤ 0,** send back an error message with status code 400. \_\_\_\_ / 2.5pts
* Calculate, log, and send back the average mpg. \_\_\_\_ / 5pts
  + Calculate the average mpg as **milesDriven / gallonsUsed**
  + Log the calculated mpg to the console (using [debug](https://www.npmjs.com/package/debug)), rounded to 2 decimal places using **toFixed()**
  + Also send the calculated mpg back to the client, rounded to 2 decimal places using **toFixed(),** either as plain text or JSON

**POST /api/temperature/convert (15pts)**

* The following data must be provided as the **body** of the request, and read from **req.body:**
  + mode (either "FtoC" or "CtoF") \_\_\_\_ / 2.5pts
  + temp (sent as a string) \_\_\_\_ / 2.5pts
* Parse and validate the provided input.
  + Use **parseFloat()** to parse a string into a number. (This returns NaN if it was not a valid number.)
  + If the mode is neither **"FtoC"** or **"CtoF",** send back an error message with status code 400. \_\_\_\_ / 2.5pts
  + If the temp **is NaN or ≤ 0,** send back an error message with status code 400. \_\_\_\_ / 2.5pts
* Calculate, log, and send back the converted temperature. \_\_\_\_ / 5pts
  + Convert the temperature into the opposite unit using the following formulas:
    - celsius = (fahrenheit - 32) \* (5/9)
    - fahrenheit = (celsius \* (9/5)) + 32
  + Log the converted temperature to the console (using [debug](https://www.npmjs.com/package/debug)), rounded to 2 decimal places using **toFixed()**
  + Also send the converted temperature back to the client, rounded to 2 decimal places using **toFixed(),** either as plain text or JSON

**Implement the following routes.**

**POST /api/income-tax/calc (15pts)**

* The following data must be provided as the **body** of the request, and read from **req.body:**
  + mode (either "Single" or "Married") \_\_\_\_ / 2.5pts
  + income (sent as a string) \_\_\_\_ / 2.5pts
* Parse and validate the provided input.
  + Use **parseFloat()** to parse a string into a number. (This returns NaN if it was not a valid number.)
  + If the mode is neither **"Single"** or **"Married",** send back an error message with status code 400. \_\_\_\_ / 2.5pts
  + If the income **is NaN or ≤ 0,** send back an error message with status code 400. \_\_\_\_ / 2.5pts
* Calculate, log, and send back the calculated income tax. \_\_\_\_ / 5pts
  + Calculate their total income tax using the tax brackets found on the following page:
    - <https://www.nerdwallet.com/article/taxes/federal-income-tax-brackets>
  + Log the calculated income tax to the console (using [debug](https://www.npmjs.com/package/debug)), rounded up to the next dollar using **Math.ceil()**
  + Also send the calculated income tax back to the client, rounded up to the next dollar using **Math.ceil(),** either as plain text or JSON

**POST /api/interest/calc (20pts)**

* The following data must be provided as the **body** of the request, and read from **req.body:**
  + principal (sent as a string) \_\_\_\_ / 2.5pts
  + interestRate (sent as a string, 0-100) \_\_\_\_ / 2.5pts
  + years (sent as a string) \_\_\_\_ / 2.5pts
* Parse and validate the provided input.
  + Use **parseFloat()** to parse a string into a number. (This returns NaN if it was not a valid number.)
  + If the principal **is NaN or ≤ 0,** send back an error message with status code 400. \_\_\_\_ / 2.5pts
  + If the interestRate **is NaN or ≤ 0 or > 100,** send back an error message with status code 400. \_\_\_\_ / 2.5pts
  + If the years **is NaN or ≤ 0 or > 50,** send back an error message with status code 400. \_\_\_/2.5pts
* Calculate, log, and send back the calculated final amount. \_\_\_\_ / 5pts
  + Calculate the final amount using the following formula:
    - finalAmount = principal \* ((1 + interestRate / 100 / 12) \*\* (years \* 12))
  + Log the final amount to the console (using [debug](https://www.npmjs.com/package/debug)), rounded to 2 decimal places using **toFixed()**
  + Also send the final amount back to the client, rounded to 2 decimal places using **toFixed(),** either as plain text or JSON