WEB700 Assignment 3

# Submission Deadline:

Saturday, June 14th @ 11:59 PM

# Assessment Weight:

9% of your final course Grade

# Objective:

Build upon Assignment 2 by updating our code to act as a module, creating a new Express server and adding the building blocks for a custom landing page, as well as an "about" page and custom 404 error page (to be expanded upon in further assignments). Additionally, we will be updating our server.js file to support more dynamic routes and status codes and publishing our solution using [Vercel](https://web700.ca/Resources/vercel-guide).   
  
**NOTE:** If you require a *clean version* of Assignment 2 to begin this assignment, please email your professor.

To see a sample of what the assignment should look like / how It should function when completed, see:   
[https://web700-a3-winter-2025.vercel.app](https://web700-a3-winter-2025.vercel.app/)

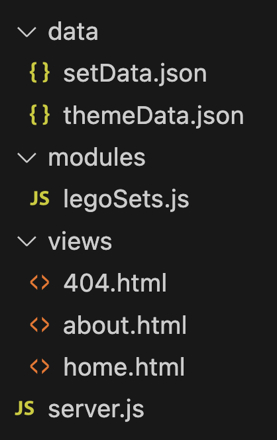
# Specification:

### **Step 1:** Updating Project File / Folder Structure

This assignment will involve updating existing files and adding / deleting others. To begin, open your completed Assignment 2 folder and make the following changes:

* Create a new folder called "**modules**"
* Rename your a2.js file as "**legoSets.js**" and place it in the newly created "**modules**" folder.
* Create a "**views**" folder and add the files:
  + **about.html** (you can copy / paste the contents of this file [here (about.html)](https://pat-crawford-sdds.netlify.app/shared/winter-2025/WEB700/A3/about.html.txt))
  + **home.html** (you can copy / paste the contents of this file [here (home.html)](https://pat-crawford-sdds.netlify.app/shared/winter-2025/WEB700/A3/home.html.txt))
  + **404.html** (you can copy / paste the contents of this file [here (404.html)](https://pat-crawford-sdds.netlify.app/shared/winter-2025/WEB700/A3/404.html.txt))
* Create a new "**server.js**" file

Once complete, your file / folder structure should now look like the following:



### **Step 2:** Updating "legoSets.js" to function as a "module"

Our newly renamed "legoSets.js" file will serve as the main module for interacting with the Lego Data in our server. To achieve this, we need to ensure that:

* Our "testing" code is deleted (this would be all code **outside** of the class definition - leaving only the **class LegoData** **{ … }** definition
* Update your "require" statements for "setData" and "themeData" to use "**..**" instead of ".", ie:

const setData = require("../data/setData");

const themeData = require("../data/themeData");

* Adding the **LegoData** class to module.exports, ie: **module.exports = LegoData;**
* Finally, add the following two lines to the top of your **server.js** file (this will ensure that we can use our module within our main "server" logic):

const LegoData = require("./modules/legoSets");

const legoData = new LegoData();

### **Step 3:** Writing our "server.js" code.

At the moment, our server.js code only has the logic to create a new instance of our LegoData class (exported from our "legoSets" module). In order to properly utilize the new legoData object in our code, we will need to create a functioning web server:

* Begin by following the course website guide to create a [Simple Web Server using Express.js](https://web700.ca/Web-Server-Introduction/simple-web-server-using-expressjs). - this will involve:
  + Creating a package.json file using "npm init"
  + "installing" express using "npm install express"
  + "requiring" express and creating the "app" object in server.js
  + Creating an HTTP\_PORT constant
  + Configuring two "GET" routes: "/" and "/about" to respond with the files: "home.html" and "about.html" respectfully
  + Starting your server by "listening" to the "HTTP\_PORT" constant (**NOTE:** The code to "listen" to HTTP\_PORT must **only** be executed if the "legoSets.initialize()" function **resolves** successfully. Otherwise, output the error to the console)

Once you have a simple web server running with two routes returning the provided html (home.html & about.html), we can concentrate on writing routes that use our legoData object:

* **GET "/lego/sets"**  
    
  This route is responsible for responding with a Lego sets (array) from our legoData object, according to the following specification:  
  + If there is a "theme" query parameter present, respond with Lego data for that theme, ie "/lego/sets?theme=technic" will respond with all sets in your collection with the "technic" theme
  + If there is not a "theme" query parameter present, respond with all of the unfiltered Lego data
  + If any errors occurred, return the error message with the "404" status code
* **GET "/lego/sets/:set\_num"**  
  This route is responsible for responding with a single Lego Set from our legoData object, according to the following specification:
  + Returns the Lego set with a "set\_num" value that matches the value after "/lego/sets/". For example, "/lego/sets/028-1" should return the Lego set with set\_num 028-1, etc.
  + If any errors occurred, return the error message with the "404" status code
* **Custom "404" Error**

Here, we must implement a custom "[404 error](https://webprogrammingtoolsandframeworks.sdds.ca/Advanced-Routing-Middleware/middleware#404-errors)". However, instead of returning text, respond with the 404 status code and the "/views/404.html" file

### **Step 4:** Updating our Navigation Menu Links.

You will notice that the navigation bar included with the sample .html files includes two broken (href="") links (to "Set" and "Theme"). As a final development step, update these links in all 3 files (ie: "404.html", "about.html", and "home.html") to link to a specific set as well as the full list of sets, filtered by a given theme. For example:

* <a class="nav-link" href="">Set</a> would be updated to the following (using set 0013-1, for example)  
    
  <a class="nav-link" href="/lego/sets/0013-1">Set</a>
* <a class="nav-link" href="">Theme</a> would be updated to the following (using theme "tech" for example)  
    
  <a class="nav-link" href="/lego/sets?theme=tech">Theme</a>

### **Step 5:** Deploying your Site

Finally, once you have tested your site locally and are happy with it, it's time to publish it online.   
Please reference the "[Vercel Guide](https://web700.ca/Resources/vercel-guide)" for more information / instructions.

# Assignment Submission:

1. Add the following declaration at the top of your **server.js** file:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* WEB700 – Assignment 03

\*

\* I declare that this assignment is my own work in accordance with Seneca's

\* Academic Integrity Policy:

\*

\* https://www.senecapolytechnic.ca/about/policies/academic-integrity-policy.html

\*

\* Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*

\* Published URL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Compress (.zip) your assignment folder and submit the .zip file to My.Seneca under   
   **Assignments** -> **Assignment** 3

# Important Note:

* **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
* After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
* Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.