

Sprint Two – Project Two

Build a Binary Search Tree webservice using express.

Introduction and Summary

Hello all, and welcome to the final sprint for this semester. There will be two projects in this sprint, the first one you may complete in a group of up to three people, the second one you must complete on your own (these two projects will be assigned separately). The first project, you will make a search engine website that a user can send search terms and get back results. For the second project, you will create a web service that accepts an array of numbers and gives back a JSON tree representation of those numbers. The steps for each project will first be summarized into easily digestible chunks in the sections of their respective project documents, each document will then go into details on each step to elaborate on the project requirements. Please read the entire summary for each project carefully to get an idea of the scope of the project and the core requirements before continuing into the implementation details.

Project Two

This project must be completed individually. The central idea for this project is that we're going to create a service that takes, as a request parameter, an array of numbers, and returns as a response a JSON representation of a binary search tree featuring the numbers from the array. The summary for this project is below:

Step 1. Create an interface where the user can enter a series of numbers.

Step 2. Create an express application that can service a request containing that series of numbers.

Step 3. Have your express service take those numbers, and insert them, one by one, into a Binary Search Tree. Return a JSON representation of this binary search tree as your response.

Step 4. Your project **MUST** include at least three unit tests in it.

Step 5. You **MUST** save the input and output data to a database of your choosing.

BONUS: Make the tree your return to the user a balanced binary search tree.

So, the plan for this project is that we're going to create a simple node/express application that has at least three routes.

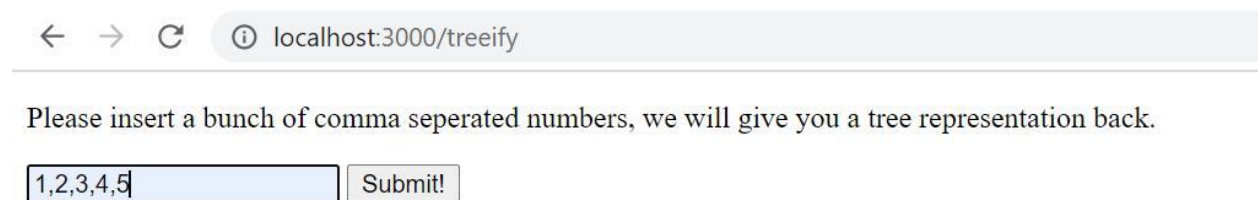
The first route is going to serve some html out to the client to give them a simple interface to use, where they can enter an array of numbers *somehow*, a submit button to submit them to the second route for processing, and a show previous button which will show previous trees saved in our database.

The second route is going to take the list of numbers, and it is going to generate a binary search tree from them, by inserting them, one at a time, into the tree – it should then return this tree, in a JSON representation, as it's response. In figure 3 below I show the output of the first route, as you can see, it is a simple interface to allow the user to enter some data, in figure 4 I show what the response JSON looks like for that input. Lastly the inputted numbers and outputted tree should be saved to a database of your choosing, which is where the third route comes in.

The third route is simply going to spit the data from your database to the webpage. This can just simply be the input array/output of the previously processed trees, you can even just represent the results as a HTML table if you wish, or as JSON output, any reasonable representation of the data is fine.

You may wish to add a homepage or improve the aesthetics of this project (particularly if you want it in your portfolio), but at a minimum that is all that is required, three simple routes, one that shows an interface, one that gives the computed results and one that shows the previously computed results.

Note: The tree that I show in my example here is self-balancing, however, you **do not** have to create a balanced version of the tree, any output that is technically a binary search tree (that is, the value of a given node's left children are always less than its own value, and the values of its right children are always greater than its value) will satisfy the minimum requirements for this project. However, it is considered a bonus objective to have the tree be self-balancing. One final thing, you **must** include at least three unit tests for this project. I'd recommend using the Jest testing framework. You can have the tests cover whatever you'd like, but an obvious thing would be your tree insertion function.



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/treeify'. Below the address bar, there is a text input field containing the numbers '1,2,3,4,5' and a 'Submit!' button to its right. The text 'Please insert a bunch of comma seperated numbers, we will give you a tree representation back.' is positioned above the input field.

Figure 3 - Input Interface Page

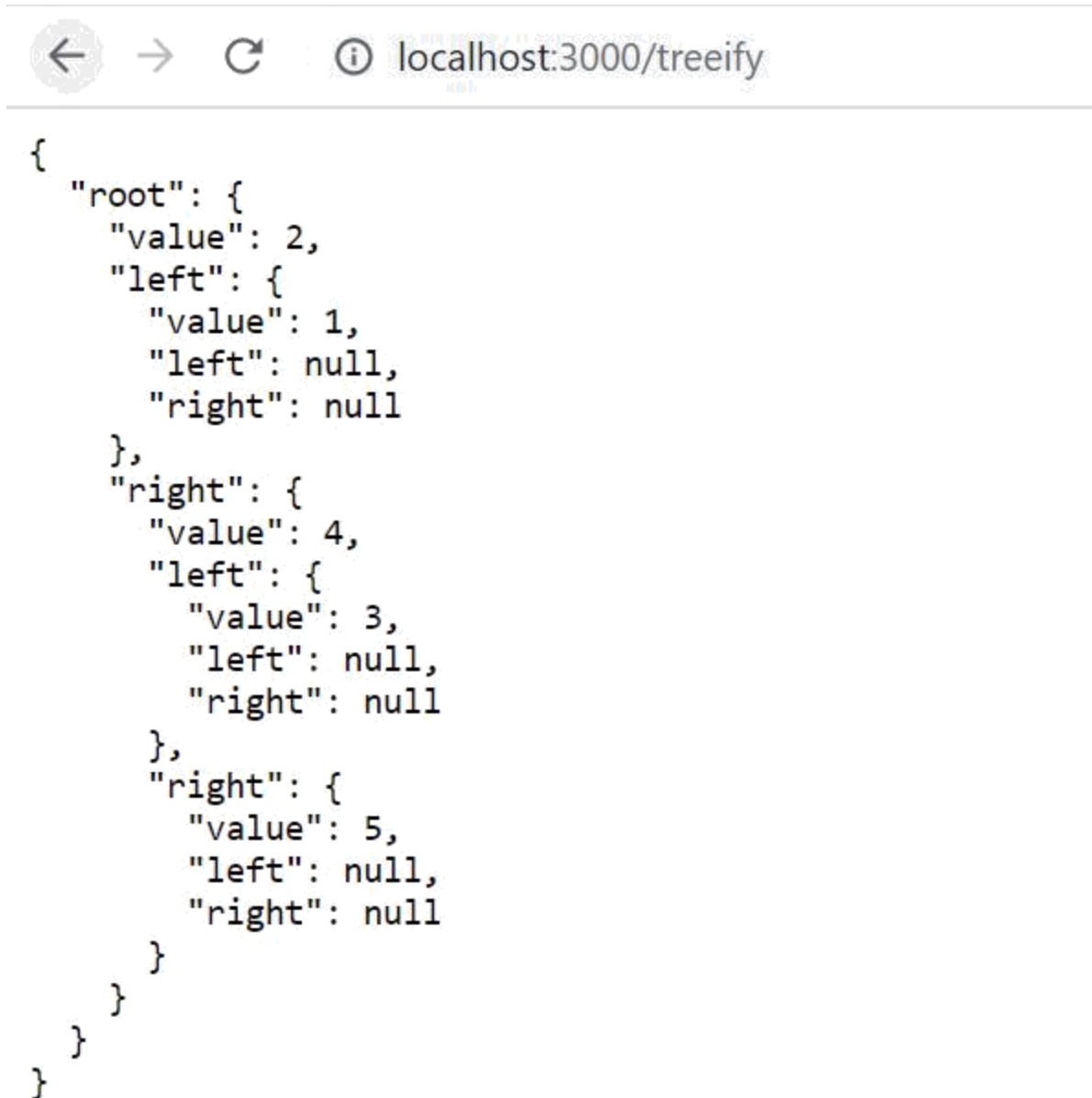


Figure 4 - Tree Result Page

Note, this is technically an implementation of a microservice – you could imagine other programs using your tree-creating API to create binary search trees from an array. It may seem simple, but there are many such simple APIs out there on the internet – and sometimes we have internal microservices that do simple things in real companies for various reasons. It may seem contrived, but this sort of pattern is worth getting familiar with in today's technological landscape.