# Quiz Section 1: Server

In this section, we will debug a very simple web application and log our debugging process.

```
git clone https://gitlab.cs.washington.edu/cse331-25sp/materials/sec01.git
```

We will assume that you have already installed Git Bash, NPM, and VSCode on your machine. Bash provides an environment where we can type commands to run programs. NPM (node package manager) reads and runs commands from a `package.json` file that describes the organization of our coding project. VSCode is an editor and is optional; you can replace it by any other editor of your choice. Perform the following steps to get started with the code provided for this section:

1. Navigate to an appropriate directory on your machine in the terminal (using `cd`). Then, run the command

   ```
   git clone https://gitlab.cs.washington.edu/cse331-25sp/materials/sec01.git
   ```

   This will copy the starter code into a new subdirectory of the current directory called `sec01`.

2. Change into the `sec01` directory (`cd sec01`). Then, run the command

   ```
   npm install --no-audit
   ```

   This will have `npm` download all of the dependencies referenced in the `package.json` file and store them locally (in the `node_modules` subdirectory) so that you can call them from your own code.

   Do not forget to include the `--no-audit` parameter. If you do forget, you will see some scary errors claiming that there are critical vulnerabilities in the code you downloaded that require fixing. This is not true. (If you want to confirm, you can run `npm audit --production` to see that there are no vulnerabilities in the code we are including in the app. `npm audit` is confused and thinks that some of the developer scripts are part of the production code, when they are not.)

3. Start the app by running the command `npm run start`, opening your browser, and navigating to `http://localhost:8080`. You should see a simple web page with a textbox and 3 buttons.

## Task 1 – Fix Fibonacci

In this problem, we will fix the function `fib` in `src/routes.js` so that it correctly returns the $n$-th Fibonacci number, where $n$ is a parameter supplied by the user as a query parameter.

(a) Run `npm run start` and navigate to `http://localhost:8080` and click the button `Calculate Fibonacci`. You will see that the message returns `undefined`.

Navigate to `https://comfy.cs.washington.edu/service/hw1-practice` to access the Debugging Log and add a new entry. In the new log entry, document this error and write down the start time of the debugging session. Then look through the console logs and see if you can find the bug. If you can, fix it.

(b) Navigate to `http://localhost:8080` and input a number in the textbot and click the button `Calculate Fibonacci`. Verify that the correct Fibonacci number is printed.

(c) Once you have fixed the bug, fill out the rest of log entry with your findings. Be sure to answer the questions:

What experiments did you run to fix the error?

What was the defect?

**Once you have added a log entry, be sure to click 'Save Log' to save your log entry. Note: For the practice website, the log will not actually save since it is just practice.**

## Task 2 – Range of Fibonacci Numbers

In this problem, we will implement the function `calculateFibonacciSeries` which caluclates all the Fibonacci numbers from a start number to an end number.

(a) Implement the function `calculateFibonacciSeries` in `src/routes.js`. This function has a single query parameter `range` which is a string of the form `start,end` where `start` and `end` are integers representing the positions in the Fibonacci sequence. See the wiki page for more information on the Fibonacci sequence.

The function should:

- Compute the Fibonacci numbers corresponding to the given range, from the start-th Fibonacci number to the end-th Fibonacci number (both inclusive). The 0th Fibonacci number is 0, the 1st Fibonacci number is 1, and the $n$-th Fibonacci number is the sum of the $(n-1)$-th and $(n-2)$-th Fibonacci numbers.
- Return the result in the following format: {`result: [fib1, fib2, ...]`}.

For example, if the range is `3,7` the function should return the 3rd, 4th, 5th, 6th, and 7th Fibonacci numbers, which are 2, 3, 5, 8, and 13. So the function should return {`result: [2, 3, 5, 8, 13]`}. If the range is `4,4` the function should return {`result: [3]`}.

(b) Navigate to `http://localhost:8080` and input a range in the textbox in the format `start,end` (e.g. `0,10`) and click the button `Find Fibonacci Numbers In Range`. Verify that the correct Fibonacci numbers are printed.

(c) Try inputting an invalid range in the textbox (e.g. `10,0` or `5,`) and click the button `Find Fibonacci Numbers In Range`. Verify that the correct behavior occurs. If your implementation does not handle this case correctly, fix it.

(d) Once you have implemented the function, be sure to Journal any bugs you encounted (if any) and how you fixed them.