

# CSCI 4131 – Internet Programming

## Homework Assignment 5 - Introduction to Node.JS

**Due Date: 11:59pm Friday April 1st (no fooling!)**

**Late Submissions accepted with Penalty after Due Date through Sunday April 1<sup>st</sup> at 5:59am**

**Submissions after 5:59am April 1<sup>st</sup> will not be accepted**

**This is an individual assignment. Do your own work using only the materials and instructions we have provided you for you for the assignment – as explicitly specified in the Syllabus. See the instructor if you have questions.**

### 1 Description

The objective of this assignment is to introduce web-server development with [Node.js](#). We will provide most of the client-side code and some of the server-side code for this assignment to you, and you are required to add/complete certain functions to complete the assignment. Node.js is basically JavaScript running a Web-server. It uses an event-driven, non-blocking I/O model. So far, in this course we have used JavaScript for client-side scripting. For this assignment, we will use JavaScript for server-side scripting. Essentially, instead of writing the server code in Python like in HW4, we will develop a basic web-server using JavaScript. There are 9 pages in this assignment specification.

In this assignment, use either JavaScript or [jQuery](#) to request data using Asynchronous JavaScript and XML (AJAX) and manipulate the Document Object Model of the Webpage making the AJAX request. [AJAX](#) is used on the client-side to create asynchronous web applications. As discussed in class and the assigned reading, it is an efficient means of requesting data from the server, receiving data from the server, and updating the web page without reloading the entire web-page.

If you want to use jQuery, which is a JavaScript library, a good tutorial to start with is available at w3schools at the link: <https://www.w3schools.com/jquery/> , and your zyBook has chapters that cover jQuery thoroughly as well.

### 2 Preparation and Provided Files

**I. The first step will be to get Node.js running on CSE lab machines or on your personal machine (see this link to install node.js on: [your personal machine](#)). *You can use node.js on the CSE labs machines as follows:***

1. Log into a CSE lab machine. This can be done with [VOLE](#) or [SSH](#).
2. Open the terminal and type the following command to add the Node.js module:  
`module add soft/nodejs`

3. The next step is to check the availability of Node.js. Type the following command to check the version of Node.js on the machine:  
`node -v`
4. This will display the current installed version if node has loaded correctly.

## II. The second step is to create a Node.js project for this assignment as follows:

Open a terminal on a CSE lab machine, then:

1. Create a directory named `<x500id_hw05>` by typing the following command:  
`mkdir yourx500id_hw05`
2. Go inside the directory by typing the following command:  
`cd yourx500id_hw05`
3. Having a file named **package.json** in Node.js project makes it easy to manage module dependencies and makes the build process easier. To create **package.json** file, type the following command:  
`npm init`
4. This will prompt you to enter the information. Use the following guideline to enter the information (The things that you need to enter are in bold. Some fields can be left blank.):

```
package name: (yourx500id_hw05) yourx500id_hw05
version: (1.0.0) <Leave blank>
description: Assignment 5
entry point: (index.js) <Leave blank> (We will provide an index.js file
for your use)
test command: <Leave blank>
git repository: <Leave blank>
keywords: <Leave blank>
author: yourx500id
license: (ISC) <Leave blank>
```

5. After filling in the above information, you will be prompted to answer the question: “Is this ok? (yes)”. Type **yes** and hit enter.
6. Now copy all the files present that are provided for this assignment to this directory: **yourx500id\_hw05**
7. List of all the available files in your HW5 directory using the Unix/Linux `ls` command should display similar to the following in your terminal window

- Highest level directory

```
nguy3817@csele-vole-48:/home/nguy3817/Desktop/HW5_SOLUTION_SPR22 $ ls
client/  contacts.json*  createServer.js*  package.json*
```

- Client directory

```
nguy3817@csele-vole-48:/home/nguy3817/Desktop/HW5_SOLUTION_SPR22/client $ ls
AddContact.html*  index.html*  MyContacts.html*
```

- Set the permissions to all your files in your directory and client folder to **777**. You can use the following Unix/Linux commands when you're in the directory with the files listed above:

- chmod -R 777 .**

- You can use **ls -al** to verify that all your files and directories are read, write, and executable from all users.

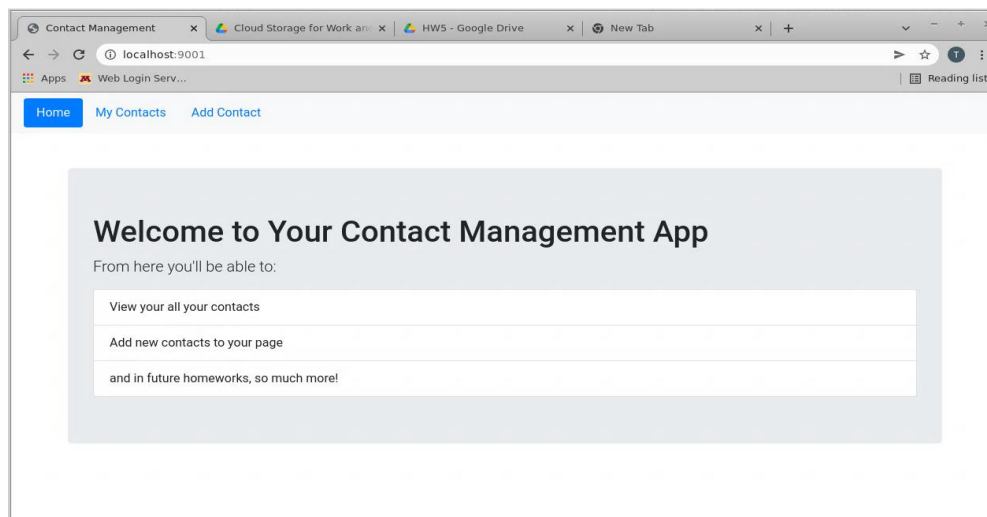
```
nguy3817@csele-vole-48:/home/nguy3817/Desktop/HW5_SOLUTION_SPR22 $ chmod -R 777 .
nguy3817@csele-vole-48:/home/nguy3817/Desktop/HW5_SOLUTION_SPR22 $ ls -al
total 55
drwxrwxrwx  3 nguy3817 CSEL-student   6 Mar 17 17:32 ./
drwx----- 27 nguy3817 CSEL-student  55 Mar 15 20:39 ../
drwxrwxrwx  2 nguy3817 CSEL-student   5 Mar 12 17:18 client/
-rwxrwxrwx  1 nguy3817 CSEL-student  813 Mar 16 22:24 contacts.json*
-rwxrwxrwx  1 nguy3817 CSEL-student 3000 Mar 15 19:52 createServer.js*
-rwxrwxrwx  1 nguy3817 CSEL-student  278 Mar 12 16:48 package.json*
nguy3817@csele-vole-48:/home/nguy3817/Desktop/HW5_SOLUTION_SPR22 $
```

- The project setup is now complete, and you are ready to start the server.

### III. To start the server, type the following command: `node createServer.js`

This starts the server and binds it to port 9001 (or whatever port you have assigned to your server). Now, using in your browser's URL bar (i.e., address bar), type: <http://localhost:9001> (or replace 9001 with whatever port you have assigned to your server)

The following page should be displayed (shown below and shown again in the screenshots on pages that follow):



The following files are provided for this assignment:

1. **createServer.js**: This file contains the partially complete code for the node.js server.
2. **client/index.html**: Home page for this application.
3. **client/MyContacts.html**: Page which displays the list of contacts for a specified category.
  - You need to fill in the TODO which would send a **GET** request to the Node.JS server via AJAX to fetch the data in the file **contacts.json** and then dynamically add the data to display a table on the **MyContacts.html** page.
4. **client/AddContact.html**: Form to add details about new events.
  - When the form is submitted it will send a **POST** request with the data entered on the form to your Node.JS server.
5. **contacts.json**: This file contains lists of events in JSON format, separated by categories.

### 3 Functionality

**Note:** It is advisable to complete the code changes for the server before changing the code for the client. All the server endpoints (APIs) can be tested using [POSTMAN](#) or [CURL](#).

### Client

All the resources related to the client have been provided in the client folder. The client folder has three HTML files (**index.html**, **MyContacts.html**, and **AddContact.html**).

**MyContacts.html** has a table (**id=contactTable**) whose body is empty. You need to add code to the TODO section that dynamically populates the contents of the table after getting the list of events (a string containing the items in the table in JSON format) from the server. You need to implement the following functionality in **MyContacts.html** file:

1. Request a list of a specified category's entries from the **getContacts** endpoint of your Node.js server using AJAX with the GET method.
2. Upon successful completion of the asynchronous AJAX **GET** request, your Node.js server will return the list of the specified category's entries.
3. Use the response returned to dynamically add rows to the table with the **id contactTable** present in **MyContacts.html** page (Create a JSON object out of the list returned and then build/render an HTML table to display the entries in the contact table).
4. You can use jQuery, JavaScript, or a mix of both to achieve this.

## Server

When the server starts, it listens for incoming connections on port 9001. This server is designed to handle only **GET** and **POST** requests.

### GET requests:

1. The server has been designed to serve four different HTML pages to clients:  
**index.html**, **MyContacts.html**, **AddContact.html**.
2. The server can also read and write to the list of event entries (in JSON format) by accessing the **contacts.json** file.
3. GET request for the **index.html**: The code for this has already been provided to you in **createServer.js** file where the server is listening on the endpoint **/** and **/index.html**.  
**You do not need to add any code for this.**
4. GET request for the **MyContacts.html** page:
  - a. When the **My Contacts Tab** is clicked on the browser, a request is sent to the server to fetch the **MyContacts.html** file.
  - b. You need to write code in **createServer.js** to listen for requests to the Server's endpoint **/MyContacts.html** and return the file **client/MyContacts.html** to the client
5. GET request to **getContacts**:
  - a. You need write code to listen on an endpoint for the **GET** request from **MyContacts.html** (the request will be seeking the contents of the **contacts.json** file for a given category)
  - b. You need to write code in **createServer.js** to fetch json data from the categorys in the **contacts.json** file and return the json data to **MyContacts.html** (which will then be parsed and displayed by **MyContacts.html** in table format) when a category is selected.
  - c. Your server should only return a given category's contacts for any request.
  - d. The contacts must be displayed in the order of oldest to newest on that category's tab.
  - e. **BONUS EXTRA CREDIT:** Instead of displaying contacts in the order you added them, **SORT** the contacts by **Contact Name**, and display them in sorted order instead.
6. GET request for the **AddContact.html** page:
  - a. When the **Add Contact Tab** button is clicked on the browser, a request is sent to the server to fetch the **AddContact.html** file.
  - b. You need to write code in **createServer.js** to listen for requests to the endpoint **/AddContact.html** and return the file: **client/AddContact.html** to the requesting client.

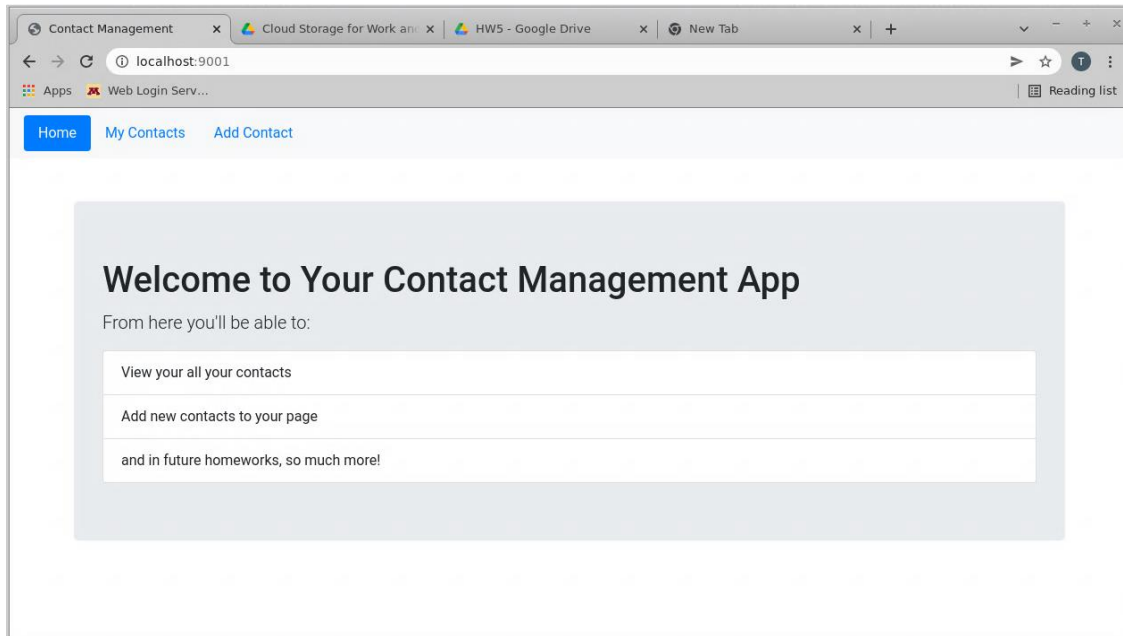
7. **GET request for any other resource:** If the client requests any resource other than those listed above, the server should return a 404 error. The implementation is already provided in the code we've provided for you.
8. **POST requests:**
- The server should process the form data posted by the client. The form we've provided in the file **AddContact.html** enables a user to enter details about a new contact and update the list of contacts in **contacts.json**. The user enters the **Contact Name, Category, Location, Contact Information, Email, Website Title, and Website URL** in the form.
  - **Website Title** will be the clickable text for the **Website URL**.
  - Details for one contact is pre-populated in the **contacts.json** file. Your job is to add code that appends the details of a new contact sent via a **POST** of the data entered on the form to this file and redirect the user to the **MyContacts.html** page after successful addition of the new contact.
  - To accomplish this, your server needs to listen for requests to the **/postContactEntry** endpoint for a **POST** request from the **AddContact.html** file.
  - You need to write code to
    - i. read the data "posted" (i.e., the data the user has entered on each field) to the form)
    - ii. add the new information to **contacts.json** file
    - iii. redirect to the file **MyContacts.html**.

The code for redirection is 302.

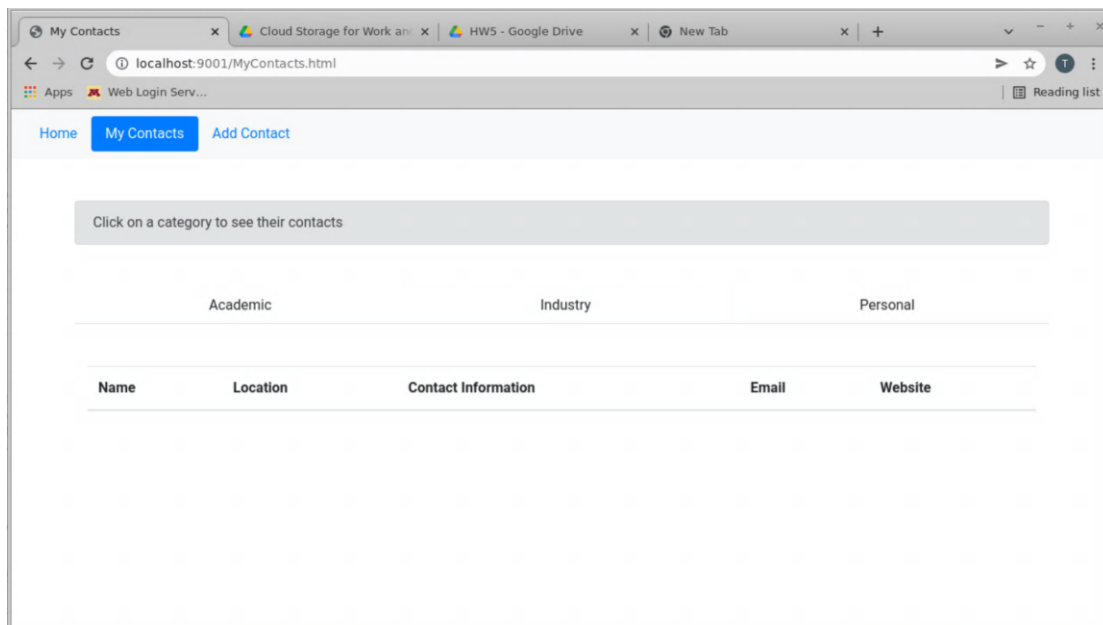
Ensure that the newly added data does not change the format of the **contacts.json** file (i.e. there are no new fields added or existing fields removed).

## 4 Screenshots

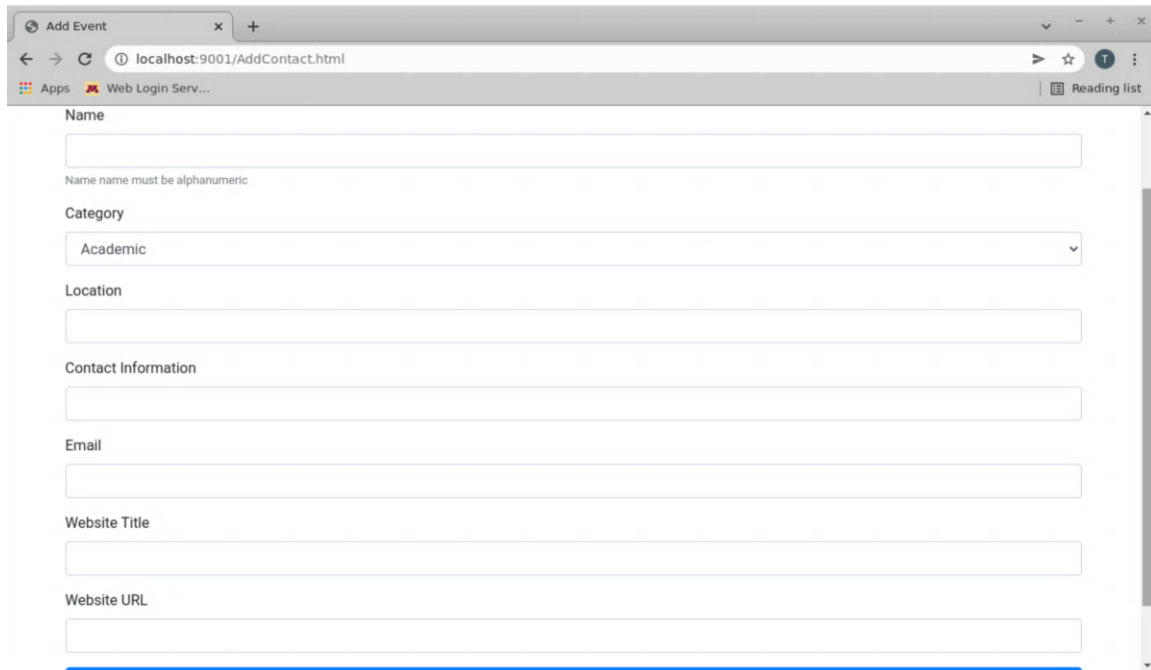
**index.html** (Should be displayed when you type: <http://localhost:9001> (or whatever port your server is using if it is not 9001) in your browser's URL bar after starting your Node.js server)



Initial display for **MyContacts.html** (displayed when user selects the **My Contacts** navigation tab)



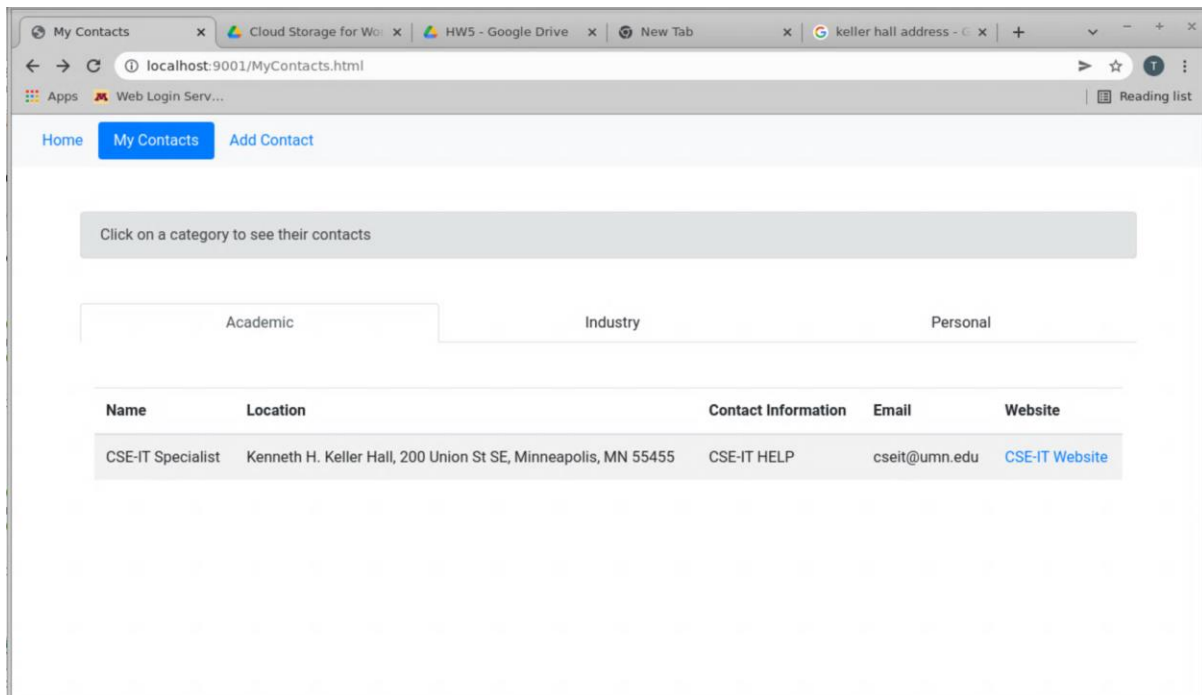
Add details for a new contact (Form in **AddContact.html** displayed when Add Contact is selected).



The screenshot shows a web browser window with the address bar displaying 'localhost:9001/AddContact.html'. The form contains the following fields and elements:

- Name**: A text input field with a placeholder message below it: 'Name name must be alphanumeric'.
- Category**: A dropdown menu currently set to 'Academic'.
- Location**: A text input field.
- Contact Information**: A text input field.
- Email**: A text input field.
- Website Title**: A text input field.
- Website URL**: A text input field.

**MyContacts.html** page after adding a new contact AND clicking the Academic tab (after completed - form is submitted with the newly entered information shown in the last row of the Academic table on click of that Academic tab)



The screenshot shows the 'My Contacts' page in a web browser. The 'My Contacts' tab is selected in the navigation bar. Below the navigation bar, there is a message: 'Click on a category to see their contacts'. Three category tabs are visible: 'Academic', 'Industry', and 'Personal'. The 'Academic' tab is selected, and a table of contacts is displayed below it.

| Name              | Location   | Contact Information | Email         | Website                        |
|-------------------|--|---------------------|---------------|--------------------------------|
| CSE-IT Specialist | Kenneth H. Keller Hall, 200 Union St SE, Minneapolis, MN 55455 | CSE-IT HELP         | cseit@umn.edu | <a href="#">CSE-IT Website</a> |



## 5 Submission Instructions

Zip your entire project directory - and the name of the zipped folder should be named:  
**yourx500id\_hw05.**

## 6 Evaluation

Your submission will be graded out of 100 points on the following items:

1. **MyContacts.html** is successfully returned by the server (**15 points**).
2. **AddContact.html** is successfully returned by the server (**15 points**).
3. Client successfully gets the list of contacts from the server. The contacts are dynamically added to the table present in the **MyContacts.html** page. (**30 points**)
4. POST endpoint successfully adds the details of the new contact entry to the **contacts.json** file in the right category (**30 points**).
5. User is redirected to the **MyContacts.html** page after successful addition of a new contact (**10 points**).
6. **Bonus:** On clicking any tab on the Contacts Page (**MyContacts.html**), the contacts displayed are sorted in lexicographically ascending order by **Contact Names** using server side scripting (**10 points bonus**).