CSCI 4131 – Internet Programming

Homework Assignment 5 - Introduction to Node.JS

Due Date: 11:59pm Saturday, April 6th

Late Submissions accepted with Penalty after Due Date through Monday April 9th at 11:59pm

Submissions after 11:59 April 9th will not be accepted

This is an individual assignment. Do your own work – 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.

In this assignment, use FETCH and manipulate the Document Object Model of the Webpage making the FETCH request. FETCH 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. *There are 10 pages in this assignment specification.*

2 Preparation and Provided Files

I. The first step will be to get Node.js running on CSE lab machines or your personal machine. This can be accomplished on CSE lab 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.

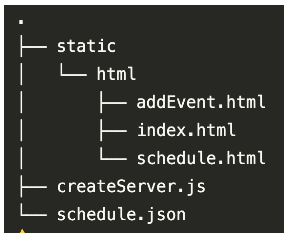
II. The second step is to create a Node.js project for this assignment as follows:

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: (createServer.js) <Leave blank> (We will provide an
createServer.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. Listing (**`tree`**) all the available files in your HW5 directory should display similar to the following:



```
.
├── static
│   └── html
│       ├── addEvent.html
│       ├── index.html
│       └── schedule.html
├── createServer.js
└── schedule.json
```

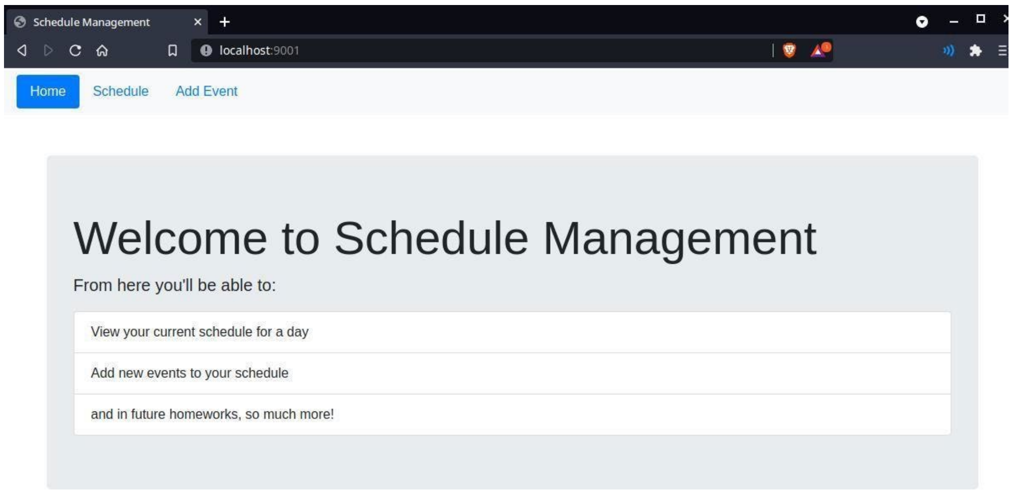
8. The project setup is now complete, and you are ready to start the server.

III. To start the server, type the following command:

```
node createServer.js
```

This starts the server and binds it to port 9001. Now, using in your browser's URL bar (i.e., address bar), type: **[http:// localhost:9001]**

The following page should be displayed (below, and shown again in the screenshots below):



Welcome to Schedule Management

From here you'll be able to:

- View your current schedule for a day
- Add new events to your schedule
- and in future homeworks, so much more!

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/schedule.html`**: Page which displays the list of events for a day.
   - You need to fill in the TODO which would send a **`GET`** request to the Node.JS server via FETCH, requesting that server reads and returns the data in the file **`schedule.json`** and then add the JavaScript or jQuery necessary to dynamically add the data to display a table on the **`schedule.html`** page.
4. **`client/addEvent.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. **`schedule.json`**: This file contains lists of events in JSON format, separated by day of occurence.

3 Functionality

Note: We advise you 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`.

The functionality for the client and server is specified on the following pages

Client

All the resources related to the client have been provided in the client folder. The client folder has four HTML files (**`index.html, schedule.html,`** and **`addEvent.html`**).

`schedule.html` has a table (**`id=scheduleTable`**) 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 **`schedule.html`** file:

1. Request a list of a day's event entries from the **`getSchedule`** endpoint of your Node.js server using FETCH with the `GET` method.
2. Upon successful completion of the asynchronous FETCH **`GET`** request, your Node.js server will return the list of event entries.
3. Use the response returned to dynamically add rows to the table with the **`id scheduleTable`** present in **`schedule.html`** page (Create a JSON object out of the list returned and then build/render an HTML table to display the entries in the schedule). Note the format of each column in the provided images, notably that **`info`** contains a link to the **`url`**.
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 three different HTML pages to clients: **`index.html, schedule.html, addEvent.html`**.

2. The server can also read and write to the list of event entries (in JSON format) by accessing (reading from and writing to) the file: **`schedule.json.`**

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 **`schedule.html`** page:
    a. When the **Schedule Tab** is clicked on the browser, a request is sent to the server to fetch the **`schedule.html`** file.
    b. You need to write code in **`createServer.js`** to listen for requests to the Server's endpoint **`/schedule.html`** and return the file **`client/schedule.html`** to the client

5. <u>`GET` request to `getSchedule`</u>:
    a. You need write code to listen on an endpoint for the `GET` request from `schedule.html` (the request will be seeking the contents of the `schedule.json` file for a given day)
    b. You need to write code in `createServer.js` to read json data from the day in the `schedule.json` file and return the json data to `schedule.html` (which will then be parsed and displayed by `schedule.html` in table format) when a day is selected.
    c. Your server should only return a single day's events for any request. The filtering should be done in createServer.js (it should not be done on the front end/client).
    d. The events must be displayed in ascending order on the events' start time.

6. <u>`GET` request for the `addEvent.html` page</u>:
    a. When the **Add Event Tab** button is clicked on the browser, a request is sent to the server to fetch the `addEvent.html` file.
    b. You need to write code in `createServer.js` to listen for requests to the endpoint `/addEvent.html` and return the file: `client/addEvent.html` to the requesting client.

7. <u>`GET` request for any other resource</u>: 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 file createServer.js that we've provided to you.

8. <u>`POST` requests:</u>
    - The server should process the form data posted by the client. The form we've provided, in the file `addEvent.html,` enables a user to enter details about a new event and update the list of events. The user enters the **Event Name**, **Day**, **Start Time, End Time, Phone Number, Location, Extra Information,** and **URL** in the form.
    - Details for a few events are pre-populated in the `schedule.json` file. Your job is to add code that appends the details of a new event sent via a `POST` of the data entered on the form to this file and redirect the user to the `schedule.html` page after successful addition of the new event. This information must be maintained in ***sorted order*** by `start time`.
    - To accomplish this, your server needs to listen for requests to the `/postEventEntry` endpoint for a `POST` request from the `addEvent.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 `schedule.json` file ***in sorted order***
        iii. redirect the file `schedule.html`.
        iv. **The code for redirection is 302.**

Please ensure that the newly added data does not change the format of the `schedule.json` file (i.e. there are no new fields added to **Name** through **Extra Information** or existing fields removed).

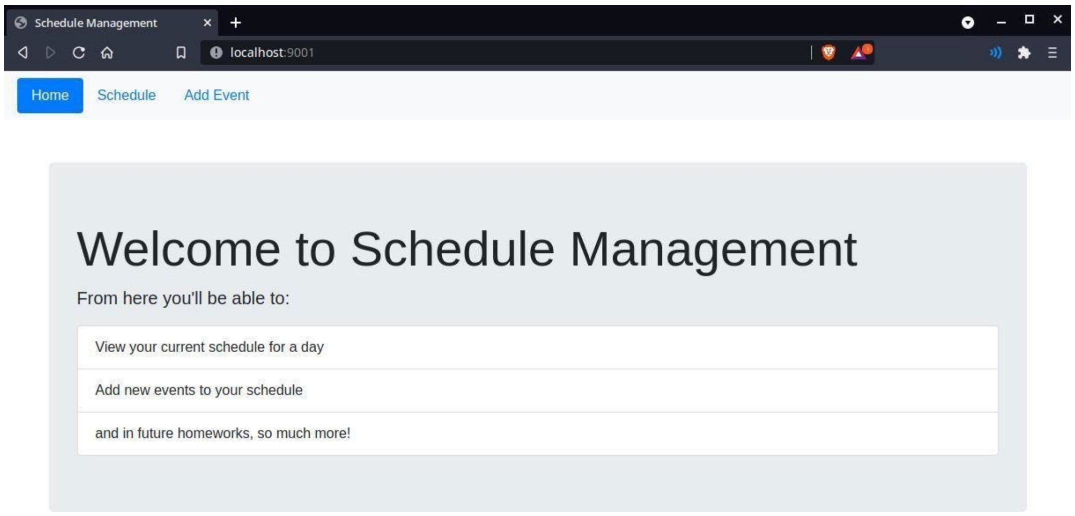
Tasks for **bonus (Hints, it is a bonus!!! – *1 more hint in Evaluation Section*)**

Add a new endpoint to your server `eventInterferes` which responds to a `GET` request of a potential new event. This must take a `day, start,` and `end time`. This will return a potentially empty list of events which occur within the new event's time.

1. The frontend must have a new button in `addEvent.html` which sends a request to the `eventInterferes` endpoint with `day, start,` and `end time` information.
    a. If no events interfere - a new html element appears on the html page signifying no interference.
    b. If any events interfere - a list of events which interfere must be displayed in a new html element.

4 Screenshots

`index.html` (Should be displayed when you type: [http:// localhost:9001](http://localhost:9001) in your browser's URL bar after starting your Node.js server) – ***change port on server if running on CSELabs machine***

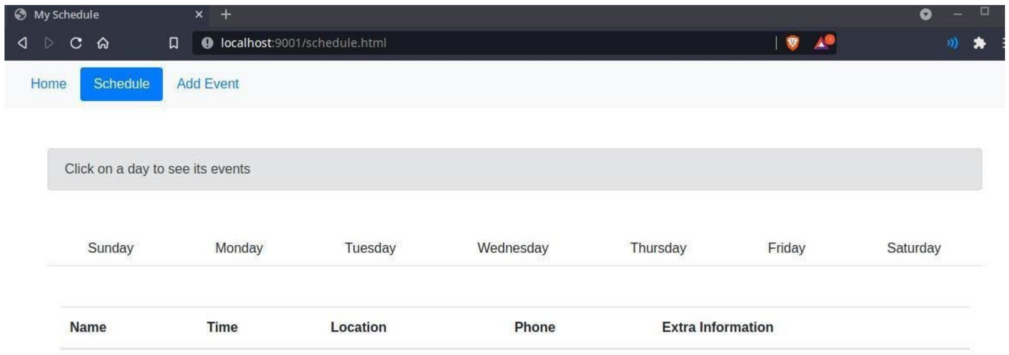


Welcome to Schedule Management

From here you'll be able to:

- View your current schedule for a day
- Add new events to your schedule
- and in future homeworks, so much more!

Initial display for **`schedule.html`** **(**displayed when user selects schedule menu item**)**

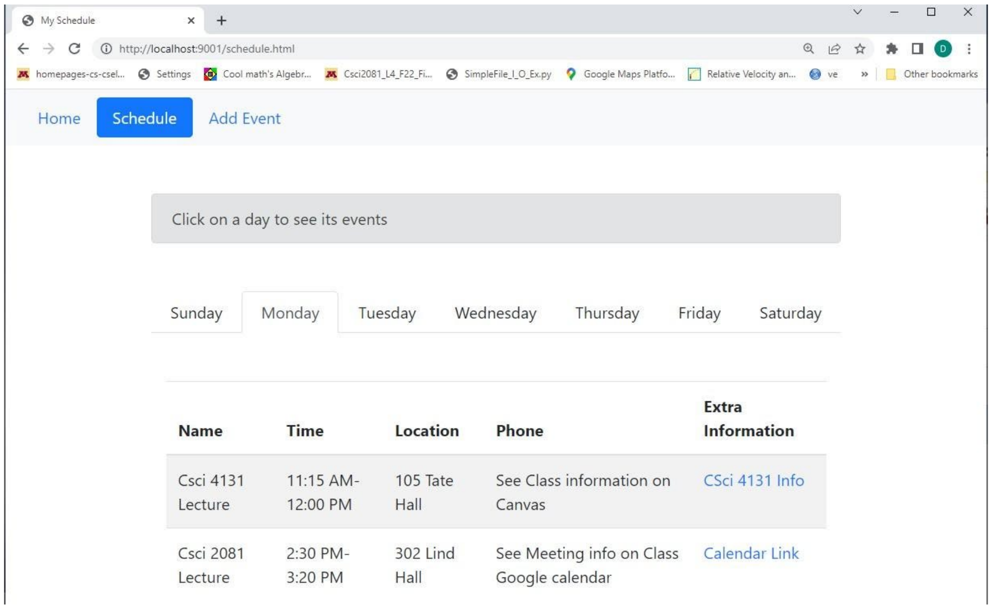


Click on a day to see its events

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday

Name	Time	Location	Phone	Extra Information

Selection of day in **`schedule.html`**

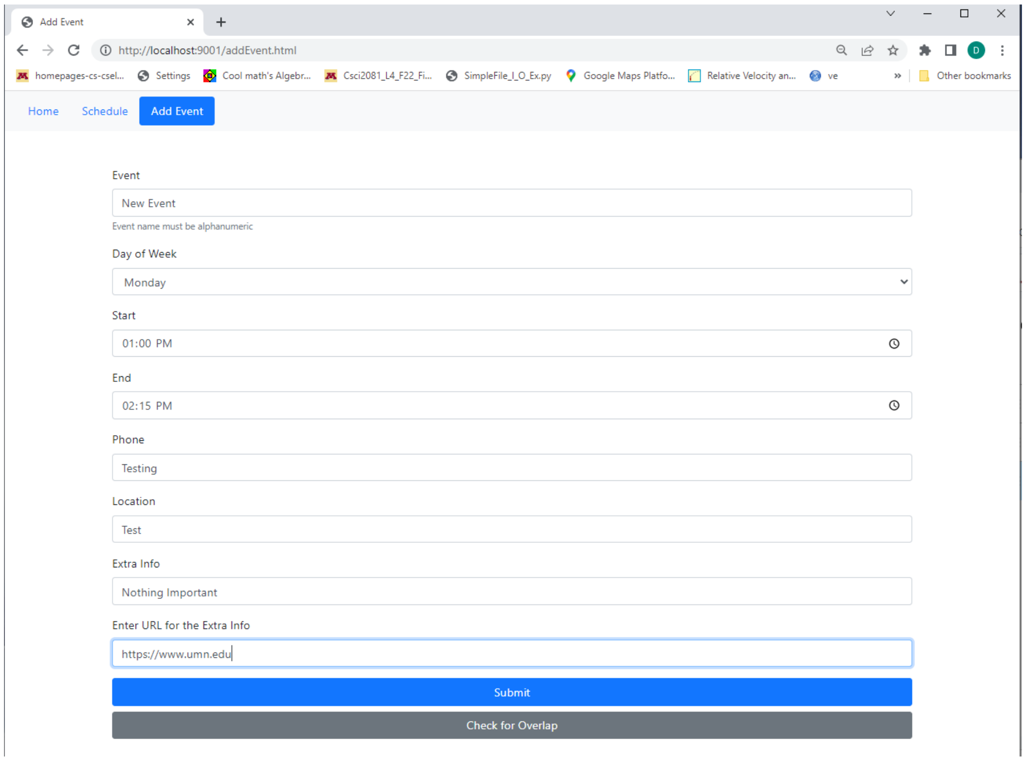


Click on a day to see its events

Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday

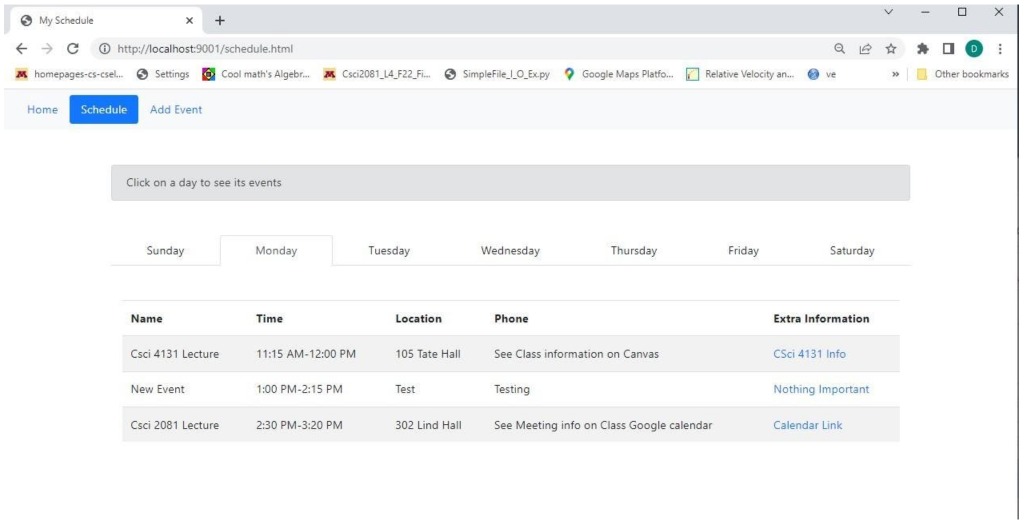
Name	Time	Location	Phone	Extra Information
Csci 4131 Lecture	11:15 AM-12:00 PM	105 Tate Hall	See Class information on Canvas	CSci 4131 Info
Csci 2081 Lecture	2:30 PM-3:20 PM	302 Lind Hall	See Meeting info on Class Google calendar	Calendar Link

Add details for a new event (Form displayed when Add Event is selected – and we have filled out the form).



Event	New Event
	Event name must be alphanumeric
Day of Week	Monday
Start	01:00 PM
End	02:15 PM
Phone	Testing
Location	Test
Extra Info	Nothing Important
Enter URL for the Extra Info	https://www.umn.edu

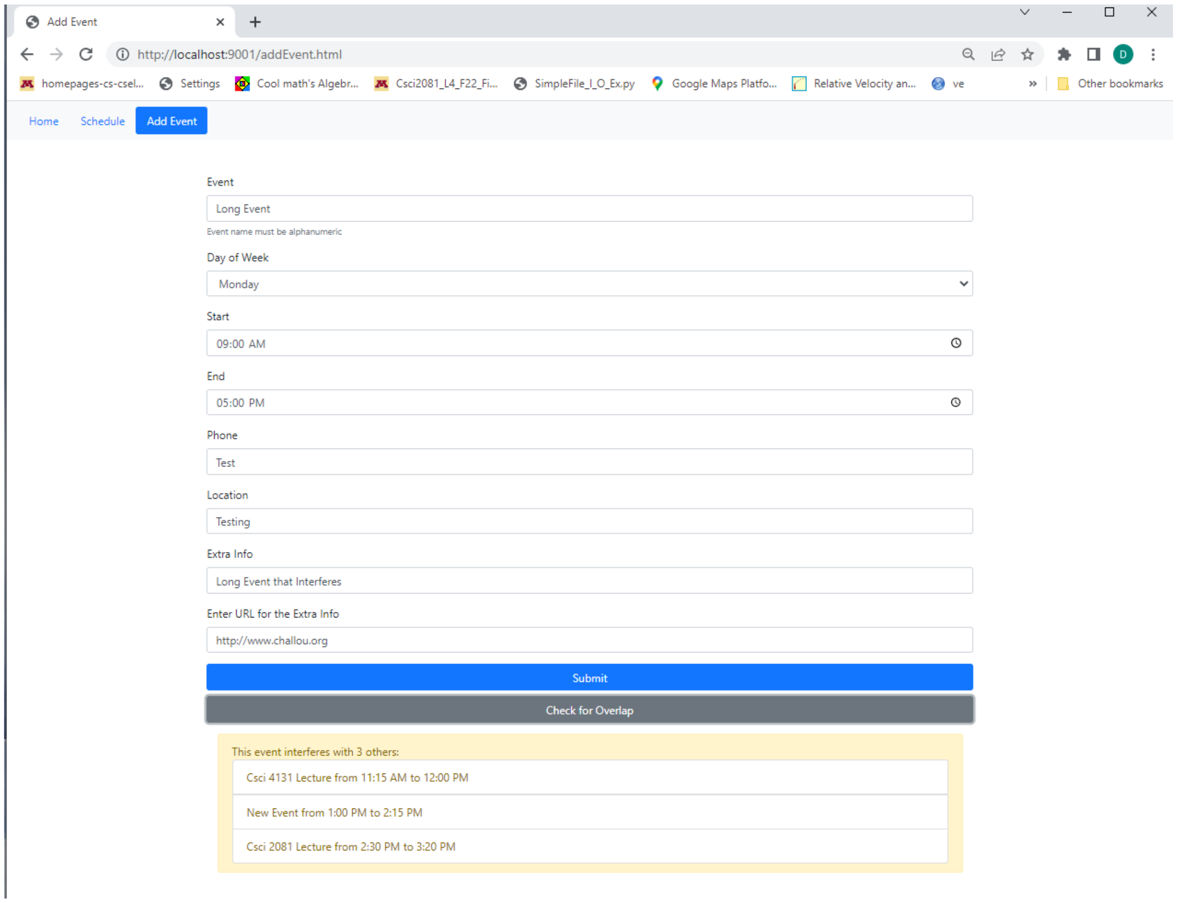
`schedule.html` page after adding a new event (after completed form above is submitted with the information shown in the middle row of the events displayed below)



Click on a day to see its events

Name	Time	Location	Phone	Extra Information
Csci 4131 Lecture	11:15 AM-12:00 PM	105 Tate Hall	See Class information on Canvas	CSci 4131 Info
New Event	1:00 PM-2:15 PM	Test	Testing	Nothing Important
Csci 2081 Lecture	2:30 PM-3:20 PM	302 Lind Hall	See Meeting info on Class Google calendar	Calendar Link

`addEvent.html` when events intersect with an existing one.



Event: Long Event

Event name must be alphanumeric

Day of Week: Monday

Start: 09:00 AM

End: 05:00 PM

Phone: Test

Location: Testing

Extra Info: Long Event that Interferes

Enter URL for the Extra Info: http://www.challou.org

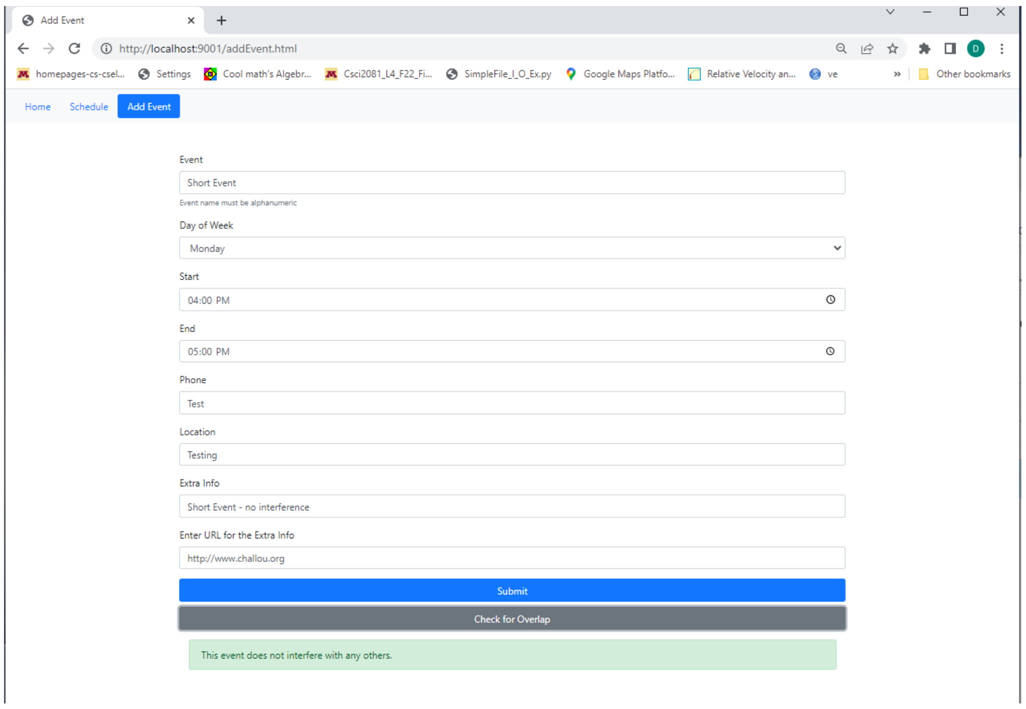
This event interferes with 3 others:

Csci 4131 Lecture from 11:15 AM to 12:00 PM

New Event from 1:00 PM to 2:15 PM

Csci 2081 Lecture from 2:30 PM to 3:20 PM

`addEvent.html` when **no** events intersect with an existing one.



Event

Short Event

Event name must be alphanumeric

Day of Week

Monday

Start

04:00 PM

End

05:00 PM

Phone

Test

Location

Testing

Extra Info

Short Event - no interference

Enter URL for the Extra Info

http://www.challou.org

Submit

Check for Overlap

This event does not interfere with any others.

5 Submission Instructions

Zip or Tar your entire project directory - and the name of the zipped folder should be your `Your_x500id_hw05`.

6 Evaluation

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

1. **`schedule.html`** is successfully returned by the server (**15 points**).
2. **`addEvent.html`** is successfully returned by the server (**15 points**).
3. Client successfully gets the list of events from the server. The events are dynamically added to the table present in the **`schedule.html`** page. (**30 points**)
4. `POST` endpoint successfully adds the details of the new event entry to **`schedule.json`** file (**30 points**).
5. User is redirected to the **`schedule.html`** page after successful addition of a new event (**10 points**).
6. Bonus: **`addEvent.html`** has the required functionality to gather intersection information from the server. (**10 points bonus**).

Hint: *We use the node.js **moment** module to check for interferences!*