CSCI 4131 – Internet Programming
Homework Assignment 7 - Final Project

Due: Saturday, May 4th at 11:59 pm

Late Submissions (with Penalty) accepted through Wednesday, May 8th at 11:59 pm
We will not accept any submissions after May 8th, so submit what you have by that time - otherwise, you will receive a zero for this assignment

Description

Assignment 6 introduced you to using a relational database with Node.js. The Project will extend your work in Homework 6 to include delete and edit functionality. This new functionality will enable you to create new events for your schedule, edit them to fix typos and other errors, view them on an HTML page, and finally delete them. With CRUD completed, you'll walk away with a fully functional full-stack application at the end of the semester.

- **A note on extra credit:** While there are extra credit opportunities in this assignment, you won't be able to score above 100 on this project, as extra credit does not impact your other grade categories. The extra credit here is mostly provided as a way to guide you toward other topics and tools that you may find interesting if you have the time and desire to look into them. *We will not be assisting with extra credit during office hours.*

*The following are **some of the resources** you should use to familiarize yourself with Express:*

- Essential
  - Installing Express
  - Hello world example of Express
  - Basic routing in Express
  - Serving static files in Express
- Additional References
  - Express website
  - FAQ
  - Routing in Express
  - API Reference

The following are resources you should review to get familiar with SQL, MYSQL and MYSQL/Node.js

- Your zyBook!!! Chapters 11 and 14
- https://www.w3schools.com/sql/
- https://www.w3schools.com/sql/sql_ref_mysql.asp
- https://www.w3schools.com/nodejs/nodejs_mysql.asp
- Optional SQL/MYSQL: Chapter 13 Sebesta
- Using MySQL With Node.js

Preparation and Provided Files

1 Setup

There are no additional files provided. You will work with your Homework 6 submission and extend it.

If you didn't complete Homework 6 or didn't properly complete the core components of Homework 6, you should do that first. If you've got a grade back for Homework 6, make sure to fix any major errors that you may have lost points for. **A significant portion of the final grade for this project is based on functionality that should have been completed in Homework 6. If you didn't get a good grade back for Homework 6 and chose not to fix any errors, you'll lose those points again on the project**.

Make sure you can set up and run your server on VOLE or the CSE lab machines. Look into port forwarding if you want to make development much easier for yourself.

2 Functionality Overview

Your website will have 5 pages :

1. A **Welcome** Page
2. A **Login** page
3. A **Schedule** Page
4. An **Add Event** page
5. An **Edit Event** page (this is new!)

All html pages will need to use Pug templates either by being converted using a converter or written that way.

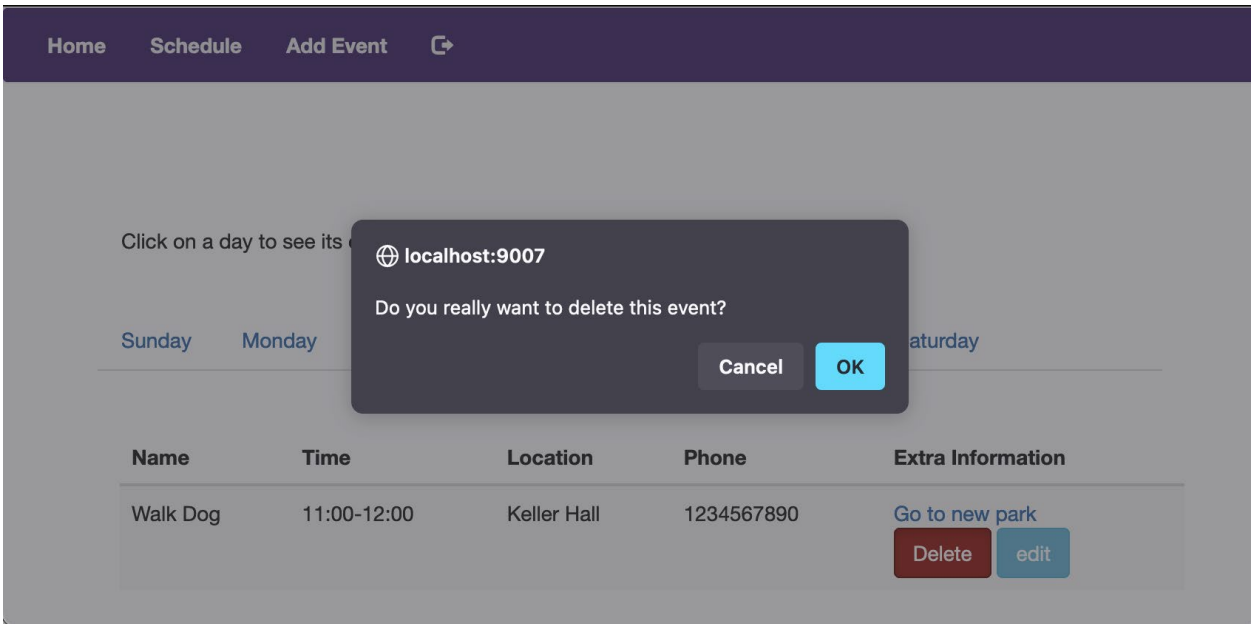
A logout button is required to end the current session once a user has logged in and should be present and functional on:

- The **Schedule** page
- The **Add Event** page
- The **Edit Event** page

NOTE: For this assignment you will need to develop the entire website including frontend (HTML pages, CSS, Javascript) and backend (Express server) + MySQL database.

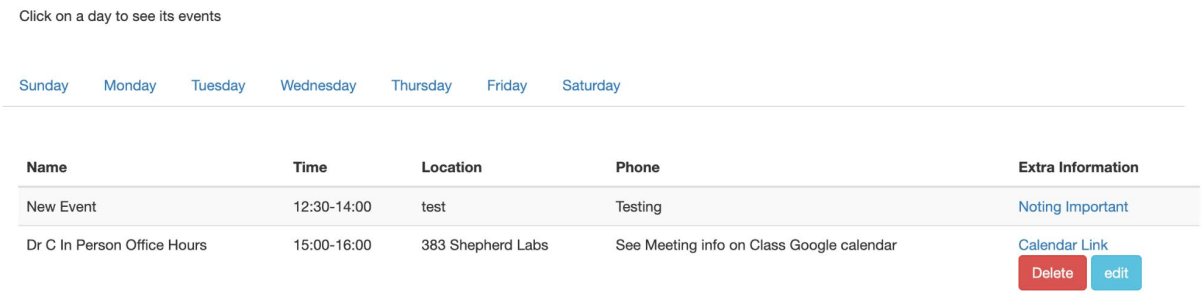
The pages below specify the functionality we have provided, and the functionality you must develop

3 Schedule Page Delete Button



You'll want to make some small changes to both the schedule page itself, and to the server.

3.1 Schedule Page client changes



The schedule page should now have buttons added to each row. By default, these buttons should not be visible.

When a user hovers over a row with an event on it, the button should be shown *inside* the row.

The button should meaningfully convey that it will delete the associated event the user is currently hovering over.

When the user clicks the delete button, the user should be asked if they're sure they want to

delete the event (preferably via a window.confirm event) and if they state they are sure, a DELETE request with the event's ID should be sent to the server.

The browser should then handle the server's response, removing the row from the displayed page.

It is up to you how you wish to handle displaying the button, but you should have some client-side JavaScript that handles the DELETE request and DOM manipulation on a successful response back.

3.2 Server-side deletion

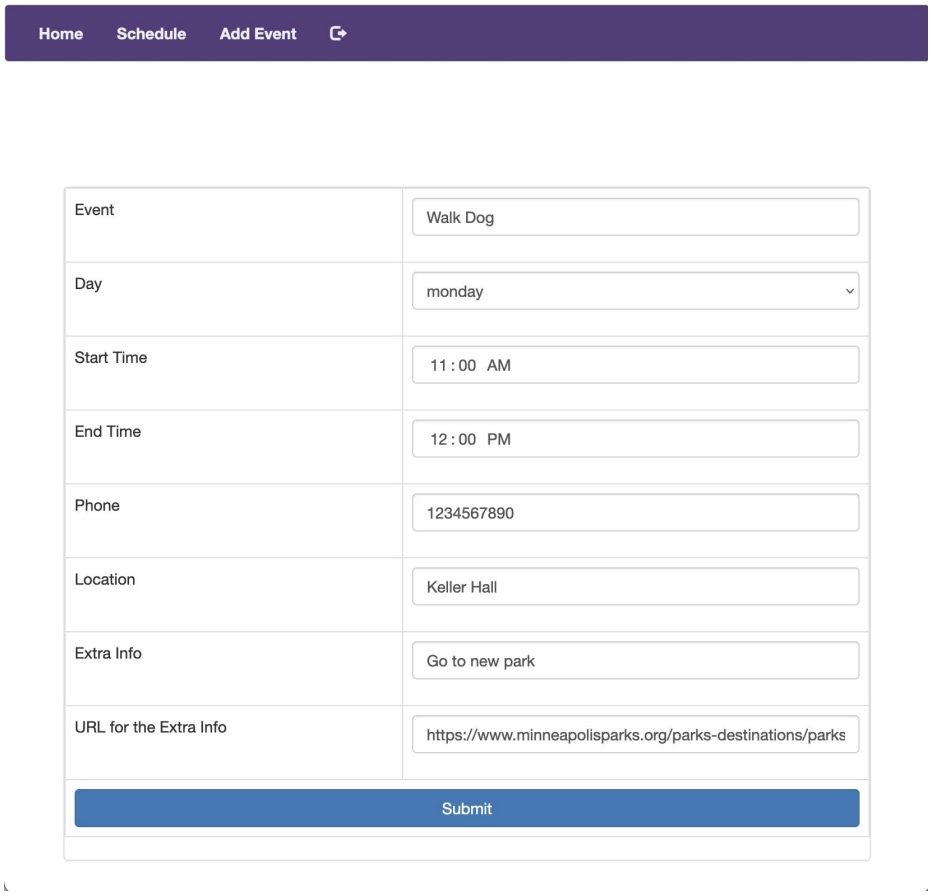
You'll also need to update the server to accommodate a DELETE request.

Namely, when a DELETE request is made, the server should first check if an event exists with the ID passed to it, if it does, the server should delete that row in the database, and on a successful row deletion, return a 200 OK response to the server.

What if there isn't an event with the requested ID? In that case, the server should return a 404 response, and *not* delete anything in the database.

For the delete request, make sure that the ID of an event is being passed in as a path parameter, and not in the body of the request.

4 Editing an event



A client should also be able to update any of the events that they've created. Maybe their schedule changed and they need to update it, or there was a typo for a location and they don't want to see Heller Kall on the page. Regardless, you need to create an edit page.

You need to do two main things for this portion of the project. Create an edit page, and hook it up to the server.

4.1 Edit page client changes

For the edit page, you'll want to add another button that only shows on hovering a row on the schedule page. You can place it right next to the delete button if you want. This button should trigger a GET request to edit the event. An example of the request (that you don't *need* to follow) is: "GET /schedule/edit/123" where 123 is the ID of the event being edited.

When that request is made, the user should be routed to the edit page, which is rendered server side. This edit page will make a POST request to the server with a body filled with fields that need to be updated. Additionally, the form should be prepopulated with the current values of the schedule, so that the user can make changes to the event and see what the original values were before changing them. This prepopulation of values should NOT use JavaScript, rather the pug template for the edit form should

accept the row values for an event, and then render the form dynamically to include those row values as current values. (Hint: you *could* reuse the addEvent form you have from HW6)

An example request the client could make (again, that you don't *need* to follow) is "POST /schedule/edit/123" with the changes in the request body (and form encoded).

After a POST request is made, if it's successful, the user should be redirected back to the main schedule page. If it fails, there should be some notification to the user that something went wrong.

4.2 Edit page server changes

The server should now handle edit requests by first handling a GET request for the edit page with event ID. In this case, the Pug template for the edit page should be filled out, with the event information for the event matching the request's event ID.

If the event does not exist, respond with a 404 response.

The server should also handle POST requests and, in doing so, **update** the existing row (if it exists) with the new values. After a successful POST, the user should be redirected to the main schedule page.

If any errors exist, the user should be notified that the POST request failed with a 422 status and information on why the request failed. You can use the information provided in the error message in your notification to the user on the client side. For now, return a 422 response in case the database query also fails.

5 Extra Credit

Remember, that extra credit for this project won't spill over into other categories where you may have lost points. However, you can use this as an opportunity to play with some new packages and tools that are used widely in industry, and which may be interesting to you.
There are three options, one that exists primarily on the server side, one that involves cloud deployments, and one that works on the styling. Each extra credit opportunity is worth 10 points *for this project grade only*.

Remember, the TAs and Dr. Dan will NOT be helping you on any of the extra credit opportunities. This is for you to look into online and troubleshoot on your own.

5.1 Login Create an Account

So far there's a set of predefined accounts that can log in and create scheduled events. What if we wanted users to be able to create their own accounts?

To play around with this concept, we'll do a very rudimentary "create an account" feature. Here's what you should do:

1. Create a new template for creating an account, name it something like new_account.pug (this is a suggestion, name it whatever you want). Make sure it has the following fields: email, password, and confirm_password. Set up client side JS to validate that the password and password confirmation fields are identical before allowing submission to the server. This form should then make a POST request to create a new account.
2. Set up a route on your server to do a GET request which returns the rendered account creation template. If the user is already signed in and attempts to access the account creation page, redirect them to the main schedule page.
3. Set up an additional route that handles the POST request from the form. This route takes in the user's email and password, and then attempts to insert a new row into the accounts table in the database. If it is successful, log the user in and redirect them to the main schedule page. If there's an error, return them back to the form with an error message.
    a. Make sure that the same email can't be used for multiple accounts
    b. You do NOT need to validate that the email exists (in that you can send email to the email address sent)
    c. A lot of your database logic can be borrowed from other code available to you, but you may want to make some changes to your database table, it is up to you how you approach this challenge.

5.2 Sass integration

Bootstrap is fine, but honestly it looks really generic. One way to improve the look of your full stack app is to add some small customizations using Sass (Syntactically Awesome Style Sheets).

For this feature, you'll need to serve bootstrap through your server. Make updates to your Pug templates to get a static stylesheet for bootstrap from your server. This will probably involve you serving assets using the static files feature for Express: https://expressjs.com/en/starter/static-files.html

To install Sass use

>npm i -g sass

Also install bootstrap

> npm i bootstrap@3.3.7

From there create a file `yourx500id.scss` *make sure you submit this file with your project!*

Check out the tutorial for using Sass with Bootstrap here and follow along: https://getbootstrap.com/docs/5.0/customize/sass/

For now, just try to play around with colors (https://getbootstrap.com/docs/5.3/customize/color/)

and generate a new bootstrap css file which has a custom color palette.

From there make sure that your newly generated bootstrap CSS is being used by the front end pug templates, and validate this by visiting your webpages, and validate that you can see the new colors you have set. Feel free to make modifications to the Pug templates, adding class names and colors to further customize your project!
You'll realistically need to make additional changes to your PUG templates. The scaffolding you have now uses Bootstrap 3.3.7, but that's 7 years old now! Try modernizing the application to use Bootstrap 5.3.0 which is the latest version of Bootstrap (and it also supports Sass better!).

>npm i bootstrap@5.3.0-alpha3

The command above installs the latest version of Bootstrap. If you swap it in for Bootstrap 3.3.7 without making any other changes, you'll most likely have additional issues with your styling that you'll need to fix. Keep that in mind as you try to play with Sass and Bootstrap.

Bonus challenge for no additional extra points: Try playing around with CSS animations, and add some animations to your login form!

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations

This is purely to give you the opportunity to try JavaScript-free animations on your application. Consider it an opportunity to be more creative with your final project if you have time.

5.3 Cloud Deployment

As a student you have access to a lot of cloud computing resources for *free*. The Github student pack gives you free computing resources on Heroku and Microsoft's Azure, as well as a free domain for a year from NameCheap and Tech Domains. Amazon Web Services also has free computing resources for students!

Your mission, if you choose to accept it, is to take your full stack application that runs on localhost, and instead have it run in the cloud. This involves setting up your own MySQL database on the cloud (or port forwarding to your database running on the CSE machines), deploying your server to AWS/ Azure / Heroku / Linode / DigitalOcean / Google Cloud / Some other provider of your choice, and then being able to test that server by accessing it from your browser.

You can furthermore use a free domain for your application to make it easier for anyone to access it!

Make sure to take down your deployment after grading has been completed, cloud costs can quickly surprise you, and we don't want you to get a bill from your cloud provider next summer when the year's worth of student credits potentially expire.

If you successfully get your server deployed, submit an additional file named myCloudDeployment.txt which lists where your server is located, how to access it, and any other information we may need in order to properly grade your extra credit. Make sure to test your deployment to make sure we can access it, if we can't test your website on the cloud, you won't get extra credit for this assignment.

You have the choice of which cloud provider you want to use. You can use a variety of tools (like Docker containers, Terraform / Cloudformation templates, etc) to accomplish this task, but for any tools that you use, make sure you document their usage in your myCloudDeployment.txt file.

Submission Instructions

PLEASE ENSURE TO TEST YOUR CODE ON CSE LAB MACHINES.

You will need to submit all the files used to develop the website. This includes all the HTML, CSS, JavaScript, package.json, index.js and any other files.

Towards this end, make a copy of your working directory: `yourx500id_hw06`. Rename the copied folder as **yourx500id_express**.

Create a README file inside yourx500id_express directory. This README file should include: Your x500id, acc_login, and acc_password values from insert_into_accounts_table.js file. Finally, compress (e.g., tar or zip) the **yourx500id_express** directory and submit it **via Canvas.**

We will use the acc_login and acc_password values to login to your website. Ensure that these values are correct and can be used to access your website.

Please remove the **node_modules** folder from your submission! Do not leave it in your submission! Also make sure that naming conventions are followed as outlined in previous sections!

AND, submit something no later than the late submission deadline to ensure you get credit for any functionality you have working. Submissions after the late submission deadline will not be accepted, and will be assigned a grade of ZERO.

Evaluation

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

1. Submission instructions are met. **SPECIFICALLY** - remember to remove the **node_modules** folder **(5 points, really 5 points!)**
2. Homework 6 functionality works as intended (creation and viewing events). **(40 points)**
3. The Delete button is visible to the user on row hover on an event on the schedule and sends a DELETE request using the Fetch API **(10 points)**
4. DELETE request works as described on the server side, and properly deletes the row if it exists **(15 points)**
5. The Edit page is successfully populated with the selected row's attributes (including row id). The edit page is also accessible via a button on the schedule page. **(10 points)**
6. POST requests properly update the row in the database, and successfully redirect the user back to the schedule page **(15 points)**
7. All HTML code is written in a pug template and properly rendered by the server for each request. You will lose **all points for this item** if any portion of your server just returns an html file that isn't rendered by Pug. **(5 points)**
8. **The BONUS Create Account Functionality (10 point BONUS)**
9. **The BONUS Sass Integration (10 points BONUS)**
10. **The BONUS Cloud Deployment (10 point BONUS)**