Homework 3:
Adding Interesting Dynamic Behavior

CSCI 4131: Internet Programming (Fall 2022)

Deadline

The deadline for this assignment is October 28th at 11:55pm. It can be turned up to 24 hours late for a 10% deduction.

1 Change Log

- V1.0 (2022-10-13) Initial Version.

2 FAQS

- None yet.

3 Introduction

As a finale for our exploration of front-end technologies we will be adding 3 more core features to the website:

- An HTML form (with some css/js validation and details included) This will ultimately help bridge us to our backend technologies and allows you to practice your HTML form features.
- Contacts table sorting features. This will let you practice more advanced dom-manipulation patterns and show off what you can really do in JS.
- A small file restructuring that should make things a lot less chaotic.
- A weather widget that will let you practice your front-end web api usage skills and the fetch API.

3.1 Learning Goals

This assignment is designed with a few learning goals in mind. By doing this assignment you will:

- Practice working with html forms for making complex web requests
- Practice more advanced DOM manipulation features, including reading data out of dom elements and restructuring dom elements
- Further explore the "bridge" between HTML and JS through the DOM and make sure you've got a good handle on how to think about DOM coding.
- Gain experience building deliberate web requests and validating their correctness using the browser inspection features.
- Gain experience with the Fetch API
- Practice manipulating json data
- Practice reading API documentation to understand how to work with a remote API.

4 Requirements

This assignment builds upon your HW2 submission. While there will be some overlapping requirements (I.E. valid HTML and valid CSS) *most* of the requirements remain independent. If you lost points on HW2 and worry it will cause you to lose points on HW3 please reach out to course staff so we can help you resolve those issues before you undertake the changes for HW3.

The requirements for HW3 are broken into 4 parts:

1. General structural overhaul
2. Contacts form Sorting
3. New Contact Me page
4. Weather Widget

You can do these in any particular order, but you may find this order matches the order we've covered material particularly well.

4.1 General Structural Overhaul

No long-running software development would be complete without occasionally looking back at what you did and thinking "Golly, what a mess I've made!" In HW2 the "mess" was not using CSS. In HW3, the "mess" is our ad-hoc file organization. While file organization might seem irrelevant, it's important to remember that *we've been building a website the whole time* and as we learn more about URLS we have to recognize that *the way we organize our files is also the way we organize our URLS.* For small websites having little to no structure is fine, but as we grow this website larger we're going to need more organization.

restructure your files to match the following organization
(starting from a root folder for your project)

- myAboutMe.html
- myContacts.html
- myWidgets.html
- contactMe.html (you will make this later...)
- resources/
  - css/
    - myStyle.css
    - Any other css files you are using (passwordWidget.css, timeWidget.css, weatherWidget.css etc.)
  - fonts/ (optional – only if you're using a custom font-file)
    - any font files you are using
  - images/
    - any image file you are using
  - js/
    - contactMe.js (you'll be building this later)
    - passwordWidget.js
    - timeWidget.js
    - myContacts.js (you'll be building this later)

    * weatherWidget.js (you'll be building this later)
    * any other javascript files you are using

After making this change, make sure to update your links to the new file locations. This shouldn't take too long, but it is something that's worth doing carefully to make sure you don't miss a link.

IMPORTANT NOTE this changes how we expect you to zip files together for submission. We will expect that the zip file you submit has no top-level folder. On most computers this means entering the folder, selecting all files you wish to submit and then right-clicking to zip them (rather than clicking on the main folder and zipping that up).

4.2 Contacts form Sorting

This requirement asks you to make some parts of the contacts form sortable.

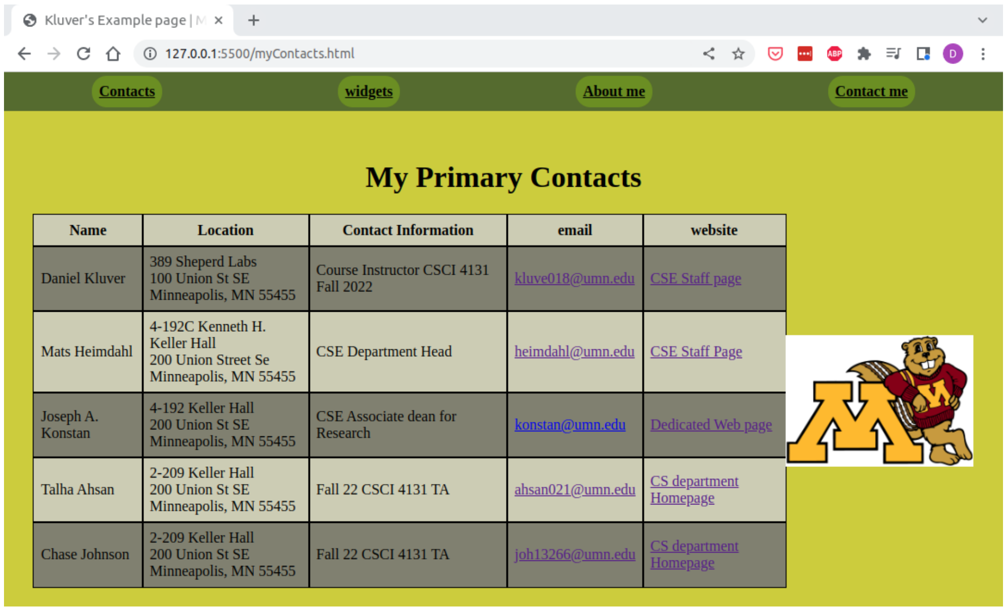
- You should begin by making a file "myContacts.js" and placing it in the resources/js folder.
- Move any existing embedded javascript from your myContacts page to this new javascript file. You are allowed to keep inline-javascript for things like *onmouseover* handlers which you configured in html rather than using javascript event binding. All other javascript (however) should be moved to this file.
- Add new javascript to this file that resorts the table either by name or email. Then either add events in javascript or html so that clicking on the email header causes the table to resort by email, and clicking the name header causes the table to resort by name.
- **Important** You must not do this by manually sorting and then somehow storing the sort-orders into javascript variables, or ad-hoc elements on the table. The sorting must be done in javascript as well as the table re-organization.
- **Important** If your table is already sorted by name or email change the initial order of rows in your HTML to not be sorted.
- **Important** If your table is the same when sorted by name and email, please add one more row of data such that the two sort orders are distinct.
- The sorting must be repeatable – so you can click between name and email sorting as many times as you want without reloading the page.
- After sorting, behavior1 and behavior2 (The preview image and thumbnail images) must continue to work, and must be correctly attached to the data in the table – so the correct images should "follow" the updated rows.
- You must write your own javascript for this – you cannot use a pre-made software library
- For extra credit make all columns sortable. For full extra-credit do this without meaningful code re-use (I.E. use a single function for all sorting, using a parameter to change which column is sorted on)

4.2.1 Hints and Pointers

- This is a pretty common task on the internet. I wouldn't recommend searching "how to sort a table in javascript" the most common answer is on several websites and in my opinion is poorly written and fails to demonstrate several key ideas about this task. The second most common answer online is elegant but unreadable. Regardless, remember that I expect you to solve this problem (Write this javascript) yourself, and referencing (even without code copy) a pre-existing solution to this problem (without attribution of your source) is scholastic dishonesty.

- This task is intimidating at first, but it's surprisingly easy **IF AND ONLY IF** you make good use of HTML and Javascript features. Think of this as a test of how efficiently and effectively you're thinking about HTML and the DOM. If you're hacking this together without really *understanding* DOM manipulations this will be quite difficult.
- It's best to think about this as a few key steps:
  1. pull the table row elements out of the DOM and into a javascript list
  2. sort the javascript list using normal sorting functions
  3. Use DOM manipulation commands to put the row elements back into the table in the correct place and order.
- If you have a DOM element you can search within it's children using methods `elem.getElementsByTagName()`, `elem.getElementsByClassName()`, `elem.querySelector()` and `elem.querySelectorAll()` to perform DOM searches only within the descendants of this element.
- If you have the DOM Element for a `<table>` you can access the `element.rows` attribute(not method) to get an `HTMLCollection` of the table row elements. (`https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableElement/rows` or `https://www.w3schools.com/jsref/coll_table_rows.asp` for more info)
- If you have the DOM element for a `<tr>` you can access the `elem.cells` attribute(not method) to get an `HTMLCollection` of td/th elements. (`https://www.w3schools.com/jsref/coll_table_cells.asp` for more info)
- Note both of these attributes return `HTMLCollection` objects, which behave a lot like lists (you can loop over them with `collection.length` and indexing `Collection[i]`) but they are not themselves lists. You may want to copy this data into lists so you can use things like the list sort method for sorting.
- Remember the list sort method requires a comparator function to describe how you wish to sort data.
- To re-arrange elements inside a given DOM node it is enough to use methods like the `append(newChild)` method – if given a node that's already in the tree dom-insertion methods always move the element. (So you don't need to worry about removing the rows from the tree before adding them in somewhere else).
- If you're using a `<tbody>` tag remember to append the `<tr>`s tags into the `<tbody>` – not back into the `<table>` itself.

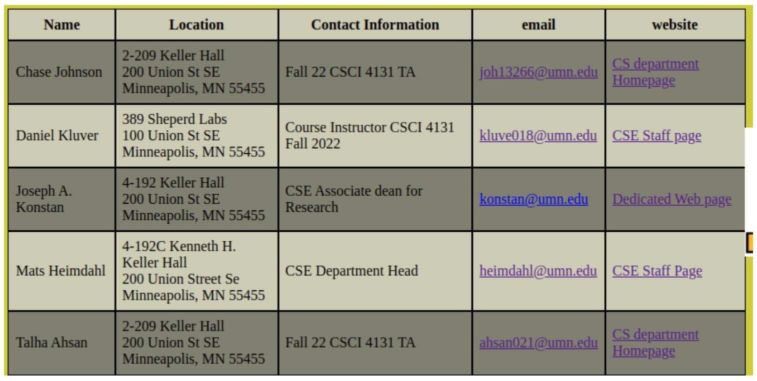
4.2.2 Images



My Primary Contacts

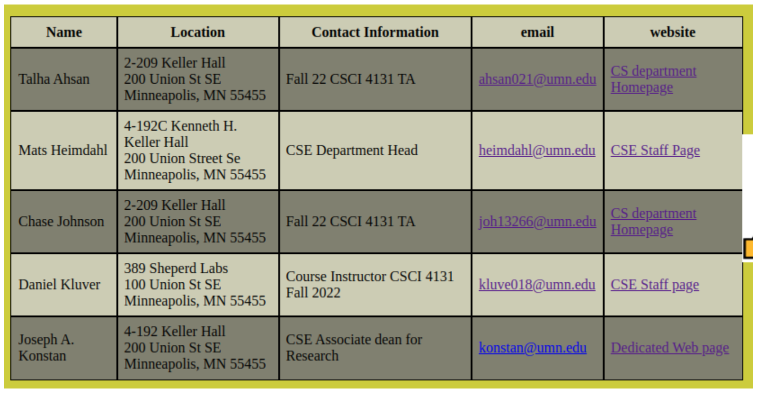
Name	Location	Contact Information	email	website
Daniel Kluver	389 Sheperd Labs 100 Union St SE Minneapolis, MN 55455	Course Instructor CSCI 4131 Fall 2022	kluve018@umn.edu	CSE Staff page
Mats Heimdahl	4-192C Kenneth H. Keller Hall 200 Union Street Se Minneapolis, MN 55455	CSE Department Head	heimdahl@umn.edu	CSE Staff Page
Joseph A. Konstan	4-192 Keller Hall 200 Union St SE Minneapolis, MN 55455	CSE Associate dean for Research	konstan@umn.edu	Dedicated Web page
Talha Ahsan	2-209 Keller Hall 200 Union St SE Minneapolis, MN 55455	Fall 22 CSCI 4131 TA	ahsan021@umn.edu	CS department Homepage
Chase Johnson	2-209 Keller Hall 200 Union St SE Minneapolis, MN 55455	Fall 22 CSCI 4131 TA	joh13266@umn.edu	CS department Homepage

Contacts page when loaded. Also shown is the link to the new page (later in this document)



Name	Location	Contact Information	email	website
Chase Johnson	2-209 Keller Hall 200 Union St SE Minneapolis, MN 55455	Fall 22 CSCI 4131 TA	joh13266@umn.edu	CS department Homepage
Daniel Kluver	389 Sheperd Labs 100 Union St SE Minneapolis, MN 55455	Course Instructor CSCI 4131 Fall 2022	kluve018@umn.edu	CSE Staff page
Joseph A. Konstan	4-192 Keller Hall 200 Union St SE Minneapolis, MN 55455	CSE Associate dean for Research	konstan@umn.edu	Dedicated Web page
Mats Heimdahl	4-192C Kenneth H. Keller Hall 200 Union Street Se Minneapolis, MN 55455	CSE Department Head	heimdahl@umn.edu	CSE Staff Page
Talha Ahsan	2-209 Keller Hall 200 Union St SE Minneapolis, MN 55455	Fall 22 CSCI 4131 TA	ahsan021@umn.edu	CS department Homepage

Contacts table after clicking the header for name – notice that it's sorted by name now.



Name	Location	Contact Information	email	website
Talha Ahsan	2-209 Keller Hall 200 Union St SE Minneapolis, MN 55455	Fall 22 CSCI 4131 TA	ahsan021@umn.edu	CS department Homepage
Mats Heimdahl	4-192C Kenneth H. Keller Hall 200 Union Street Se Minneapolis, MN 55455	CSE Department Head	heimdahl@umn.edu	CSE Staff Page
Chase Johnson	2-209 Keller Hall 200 Union St SE Minneapolis, MN 55455	Fall 22 CSCI 4131 TA	joh13266@umn.edu	CS department Homepage
Daniel Kluver	389 Sheperd Labs 100 Union St SE Minneapolis, MN 55455	Course Instructor CSCI 4131 Fall 2022	kluve018@umn.edu	CSE Staff page
Joseph A. Konstan	4-192 Keller Hall 200 Union St SE Minneapolis, MN 55455	CSE Associate dean for Research	konstan@umn.edu	Dedicated Web page

Contacts table after clicking the header for email – notice that it's now sorted by email address.

4.3 New Contact Me page

This requirement asks you to create a new page. The idea for this new page is a "Contact me" page that would be open to the public to send the webpage owner (you) a message, or equivalently post a message to an internal messaging system. In future assignments we'll look at receiving and storing this information in a back-end but for now we'll just look at how to build the form itself. The requirements for this page are split into three key components: HTML, Validation, Javascript.

4.3.1 HTML

These requirements can be accomplished using HTML and CSS alone. At this stage you will be building a functional page, patching it into the rest of your pages, and it will be able to act as a form and submit data to our "testing" API.

- Create a page "contactMe.html"
- This page should have the same general design and styling as the other html pages – including the required top-bar navigation feature.
- All other pages should have their required top-bar navigation feature updated to include a link to the contact me page.
- This page should have a unique title from the other pages.
- This page should have a contact form – this should be 50% of the page width and centered on the page. The form must be visually distinct from the page background, ideally by using a different background color.
- The contact from should have an appropriate header element saying "Contact Me".
- The page should have an HTML form on it with the following editable fields:
  - a "Title" field that accepts text (think about this as the subject of an email)
  - An "Email" field that accepts text
  - A "Username" field that accepts text
  - A "Link" field that accepts text
  - A "Category" field that uses radio-buttons to select between three options "Question", "Comment", or "Concern".
  - A "Message" field which accepts multi-line text.
- There should also be a submit button. When the submit button is clicked the form should send a post request to "https://www-users.cselabs.umn.edu/ joh13266/contactMe.php" which is a testing server we have put together for this assignment.
  - Note – this testing server tends to be slow the first time it's used after a sufficient delay. This is normal, just give it a minute to boot-up if it's being slow.
  - Submitting the form and getting to this page should require no javascript. This can all be done with pure HTML/CSS.
  - The body of the POST request generated should be in standard "form-encoded" format (this is the default) and should contain values for the following form elements "postTitle", "email", "username", "link", "category", "message" (which contain values from the fields listed earlier). Note that these field-names are case-sensitive and must match exactly to show up.
  - the category field should have values "question", "comment", or "concern" (lowercase) based on which radio button was selected.

  - When the form is correctly configured the testing website will show a table and report what you had entered for title, username, email, link, category, and message. You can use this to validate that information is being sent correctly.
- Each editable field should have an appropriate label element. When you click the label element the correct input element should be selected. Likewise, each radio button must have an appropriate label.
- The radio buttons should be in a field set element with appropriate description See `https://www.w3schools.com/tags/tag_fieldset.asp` for more information on this html element.

4.3.2 Form Validation

You can accomplish these behaviors using HTML5 form validation alone, or general-purpose javascript form validation. Do not allow the form to be submitted unless the following is true:

- title, email, username, category, and message are all required fields. If there are no values for these elements do not allow the form to be submit.
- The email field must at least *look* like a valid email address, at a minimum containing an at-sign. If this is not true do not allow the form to be submit.
- The link field must at least *look* like a valid url, at a minimum it should have a schema, then `:/` then the rest of the URL.
- (hint – if you make good choices on the input tag setup, all of this is easy to do in html alone.)

4.3.3 Javascript

This must be done in javascript, although you can modify the css/html of your page to enable this if you need to.

- Create a file "contactMe.js" and place it appropriately in the resources/js folder.
- Feature 1: auto-suggest username
  - When the email input is changed, if the new value is a valid-looking email (has an `@` sign) and the username field is currently empty, automatically update the username field with the part of the email before the `@` sign. I.E. if I type in `kluve018@umn.edu` as my email and have not filled in a username, `kluve018` should populate the email field.
  - This should not happen if there is already a username entered.
  - This should be repeatable – so if we fill in the email it should fill in a username, then if we empty the username field and update our email the username field should be **re-filled**
- Feature 2: clarification box.
  - Whenever the user indicates the message type "concern" a message should appear in a red box somewhere on the screen that reads "In order to best address your concern, please make sure to list the specific way in which I have wronged you."
  - If the user switches away from concern to Question or Comment, the box should go away until such a time that the user selects "Concern" again
  - You can use whatever CSS/HTML you find reasonable to make the notification look good, so long as it has a clear outline (ideally red, but if your whole page is super red, a

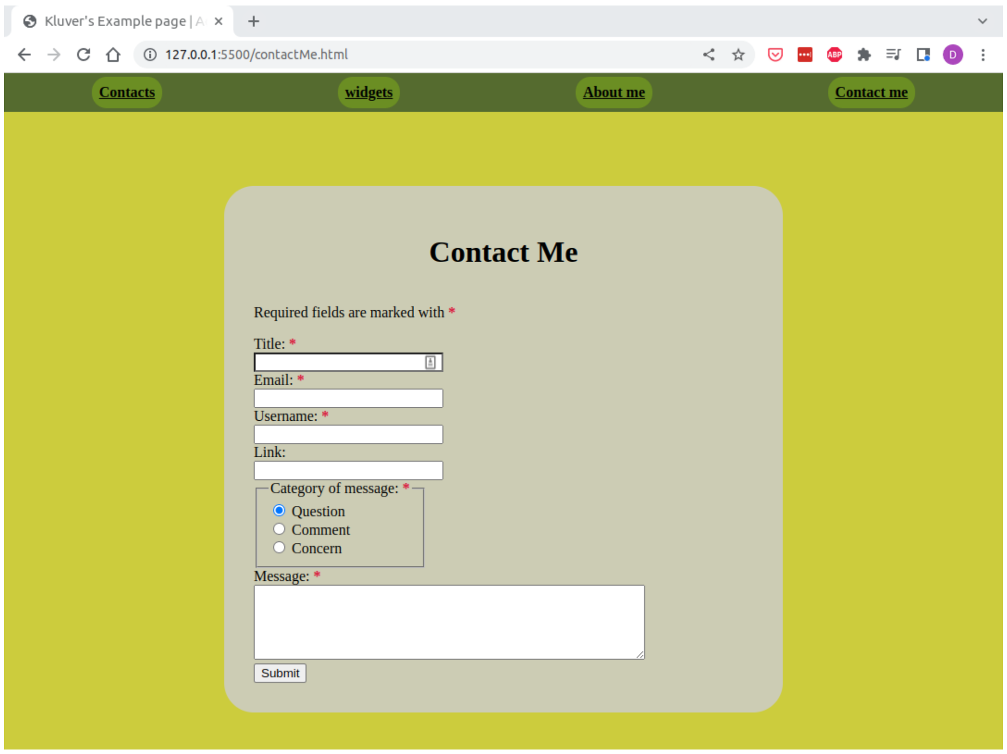
different color can be substituted). However, the dynamic behavior (Adding/removing the notification element) must be done with javascript.

- Use only event-handlers declared in javascript for this problem, do not use event-handlers configured in html.

4.3.4 Hints and Pointers

- Develop this one part at a time. There's no point working on validation when the form isn't submitting all fields as intended.
- Save css styling for towards the end – get the HTML required worked out first.
- There are two events that are commonly used with input elements the "input" event and the "change" event. The input event normally triggers each time a minor modification is made (I.E. each keystroke in an input field) whereas the change event triggers only when the user finishes changing a field (I.E. after they click out of the field into another field). For this problem you should only need the change event.
- The validation for this problem can be done entirely in HTML and without much additional effort if careful input elements are used. Try to solve the form validation without any JavaScript, and only use JavaScript as a last-resort here.

4.3.5 Images



Contact Me

Required fields are marked with *

Title: *

Email: *

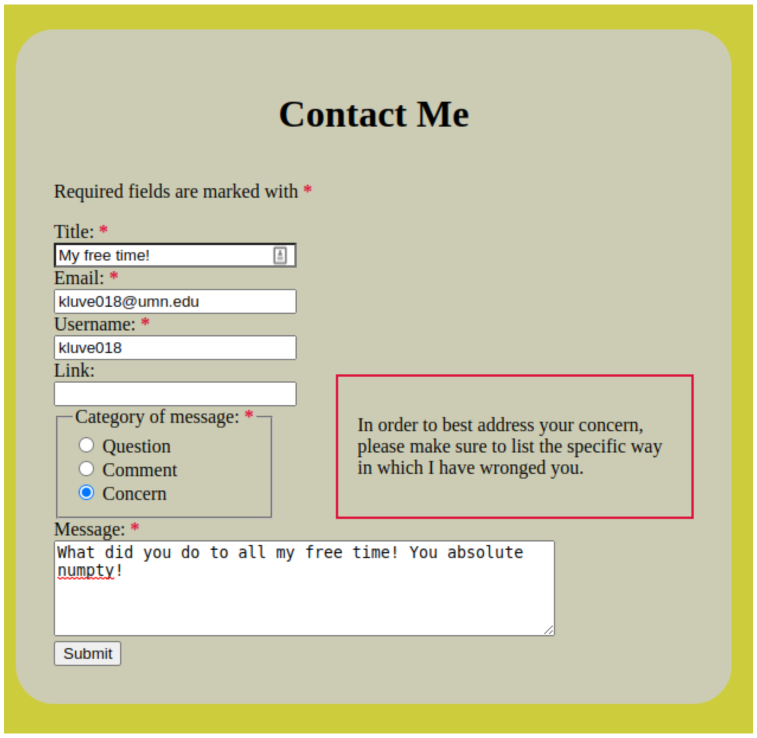
Username: *

Link:

Category of message: *
- [✓ Question]
- Comment
- Concern

Message: *

The contact me page



Contact Me

Required fields are marked with *

Title: *
My free time!

Email: *
kluve018@umn.edu

Username: *
kluve018

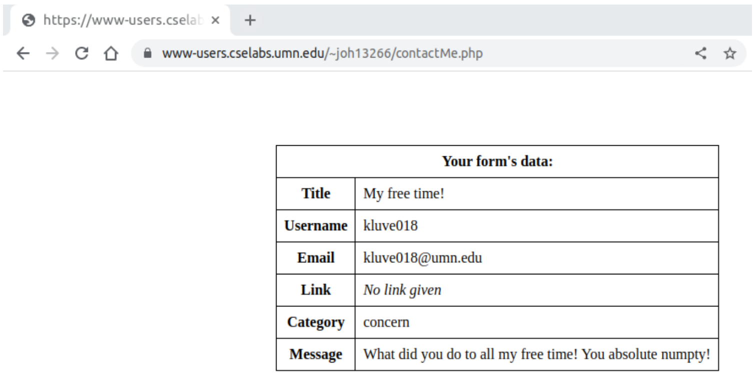
Link:

Category of message: *
Question
Comment
[✓ Concern]

In order to best address your concern, please make sure to list the specific way in which I have wronged you.

Message: *
What did you do to all my free time! You absolute numpty!

The contacts page with some data. Note that there's a message because concern is selected.



Your form's data:	
Title	My free time!
Username	kluve018
Email	kluve018@umn.edu
Link	*No link given*
Category	concern
Message	What did you do to all my free time! You absolute numpty!

After submitting the form we're showed our selections back to us. Note the URL – this is not part of our website, but is being returned by the form-tester utility.

4.4 Weather Widget

For this requirement you must add a weather widget to your widgets page. This widget has two intractable components an "Auto-find me!" button (behavior provided) that uses built-in browser APIs to get the user's approximate location (latitude and longitude) and a "Find the weather button" (you build this part) which uses the open-meteo and sunrise-sunset APIs to get the current weather conditions, temperature, predicted time of sunrise, and predicted time of sunset for the currently entered location.

- Begin by downloading the provided weatherWidget.css, weatherWidget.js, and weatherWidget.html files. Add this widget to your widgets page.
  - You are allowed to modify the weatherWidget.css as well as the provided html to suit your website better. In particular if you took our advice in HW2 and made a simple class for these widgets you may not need weatherWidget.css at all, or you may only need it for one or two elements.
  - If you do change the html provided, be aware that the provided javascript relies on certain ids. If these IDs change you will need to update the id elements.
  - If you do change the provided html, do not remove the "data provided by" links. We are legally, and ethically, required to have these attributions to our data sources.
- You should place the weather widget in whatever counts as the "bottom" of your widget page layout – for me that's the bottom and it ends up being centered. for you that might be bottom-left or bottom-right depending on how you made the widgets page be a grid.
- Before moving forward verify that you can get your latitude and longitude with the auto-find me button. This may require clicking on a popup somewhere on your screen to give your browser permission to share this information with your website. If so, that should only need to be done the first time. If you cannot get this part working, email us (as it's provided in complete form, we're not really expecting trouble here).
- When the user presses the find weather button the following should occur:
  - the user's location should be read from the latitude and longitude inputs.
  - Two separate, but concurrent, web requests should be made to the open-meteo and sunset-sunrise APIs to get the weather conditions, and sunrise/sunset times for the user's location. These requests should occur simultaneously, and not in sequence.
  - Whenever we get a response from open-meteo the widget should be updated with the current temperature, and current weather conditions. This should appear below the buttons, but above the attribution links.
  - Whenever we get a response from sunset-sunrise the widget should be updated with the current sunset and sunrise times. This should appear below the buttons, but above the attribution links.
  - The two pieces of information (weather and sunset/sunrise) should be shown asynchronously from each other based on whichever website was faster – if sunset-sunrise responds first, the sunset/sunrise time show up first, if the weather info returns first, the temperature and weather info should show up first. Commonly these will show up nearly simultaneously to human perception.
  - Regardless of the order – the weather information should show up above the sunrise/-sunset time.
  - We've provided elements for you to put these values in – you do not have to use them as is. You are free to modify the weatherOutput html element to suit your personal opinion about how this output should look.
- You must use the fetch API for this request, not any of the other ways of asynchronously making web requests.
- If an error occurs at any point, an error message should show up in the html element with id "weatherStatus". We will be looking for code to handle errors, but we do understand that this code might be untested (as errors are rare and really hard to test directly)

4.4.1 Open-meteo API

The open meteo API provides weather data. For more information about this API endpoint see `https://open-meteo.com/en/docs` As this API supports a WILD amount of options, some basic information will be provided here:

The base API endpoint (the url without query parameters) is: `https://api.open-meteo.com/v1/forecast` This resource expects GET requests, and needs query parameters to be configured to tell it what weather information is being requested. query parameters that we recommend:

- latitude (the user's latitudinal location)
- longitude (the user's longitudinal location)
- current_weather (do you want the current weather to be predicted. Give this value "true")
- temperature_unit (what temperature unit do you want). I've set this to "fahrenheit".
- windspeed_unit (what units do you want windspeed in) I've set this to "ms" (miles per second) although since we don't use this information you can probably skip this query parameter.
- timezone (what timezone should prediction-times be reported in) I've set this to "`America%2FChicago`" which decodes as "America/Chicago" which matches Minneapolis timezone. Again probably not essential for our purposes.

Taken together, we can craft a URL that gives us the current weather on the UMN: `https://api.open-meteo.com/v1/forecast?latitude=44.9740&longitude=93.2277&current_weather=true&temperature_unit=fahrenheit&windspeed_unit=ms&timezone=America%2FChicago`

Response An error is indicated by the open-meteo API by returning a non 200 status code and an error message in json. Otherwise text describing a json object will be returned as the response to this request and status code will be 200.

On a success, a wide variety of information is returned. You can find this documented in the open-meteo documentation, or you can get some example json (such as from the url above) and look at it's structure directly. Either way you might find a website like `jsonformatter.org/json-pretty-print` helpful as json is not often formatted for easy reading. Make sure you know how to identify and extract the current weather's temperature and weather code. The weather code is an int that indicates current weather conditions according to a common lookup table. You can find open-meteo's version of this lookup table in the documentation, and you can find my javascriptified version of this lookup table in the provided code for this problem.

If you're struggling to understand how to structure a URL to make requests to this API, or process the returned json – please reach out over slack in the homework questions channel. So long as we're just discussing the API behavior of the open-meteo API anyone can help any other student. (Just don't share specific code for this problem)

4.4.2 sunset-sunrise

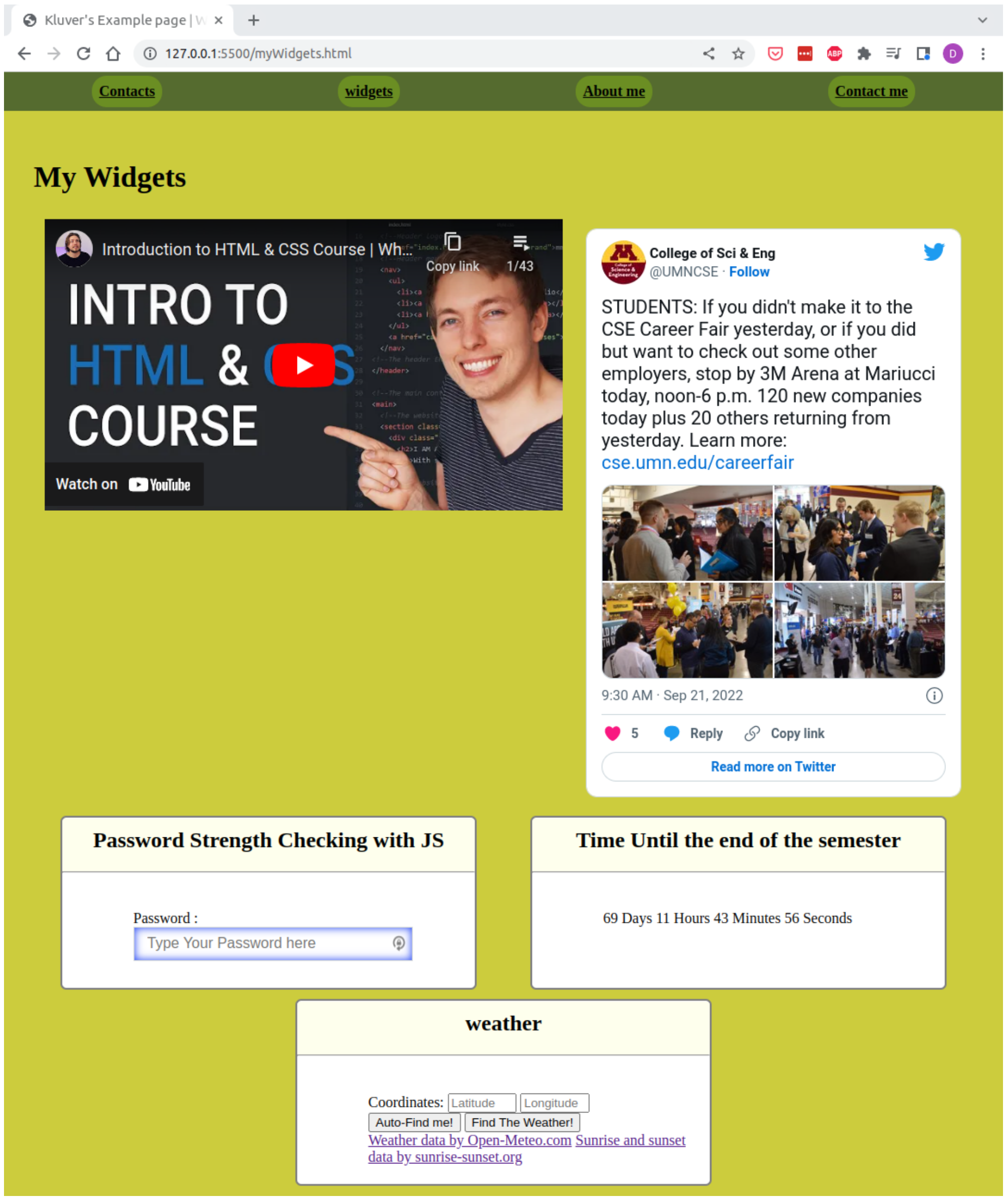
The sunset-sunrise API provides predicted time of sunset and sunrise for a given location (and a given day). For more information about this API endpoint see `https://sunrise-sunset.org/api` Unlike the open-meteo API this has a more understandable API with non-overwhelming documentation and examples. Therefore, we will largely be leaving it up to you to read the official documentation to learn how to make requests to this API.

Some notes/hints:

- Like open-meteo the API is based on a single end-point which uses GET requests and is configured using query parameters. The query parameters and their behavior are well documented.
- We want the current date.
- The callback parameter is for an old-school idea called jsonp. While jsonp is a fascinating work-around to the problem of asynchronous web requests before we had proper APIS, if you're using the fetch API you don't need to use jsonp and can ignore this parameter
- The formatted parameter confused us a bit so we'll discuss it here. If you set formatted=1 (the default) the sunrise and sunset times will be provided in a nicely formatted string, and a useless timezone (unless you're in Greenwich,UK and need to know when the sun will set in Minneapolis for some reason). Therefore we recommend using formatted=0 – this makes the API return times in standard ISO 8601 format. This can be decoded (and put into a local timezone) using javascript Date operations.
  - `new Date(string)` will decode a string in many standard formats including ISO 8061
  - `date.toLocaleTimeString()` will create a readable string in the local time zone for the time indicated by a date object.
- sunset-sunrise API returns json.

Unlike open-meteo, decoding the process to make requests and parse responses from sunset-sunrise is part of the work of the assignment, so we'll ask that you not ask for public advice on how to structure URLs for this or parse the response object. If you find that you need advice here feel free to email course staff or ask in office hours.

4.4.3 Images


Kluver's Example page | W
127.0.0.1:5500/myWidgets.html
Contacts
widgets
About me
Contact me

My Widgets

Introduction to HTML & CSS Course | Wh...
Copy link
1/43
INTRO TO HTML & CSS COURSE
Watch on YouTube

College of Sci & Eng
@UMNCSE · Follow

STUDENTS: If you didn't make it to the CSE Career Fair yesterday, or if you did but want to check out some other employers, stop by 3M Arena at Mariucci today, noon-6 p.m. 120 new companies today plus 20 others returning from yesterday. Learn more: cse.umn.edu/careerfair

9:30 AM · Sep 21, 2022

5
Reply
Copy link
Read more on Twitter

Password Strength Checking with JS

Password : Type Your Password here

Time Until the end of the semester

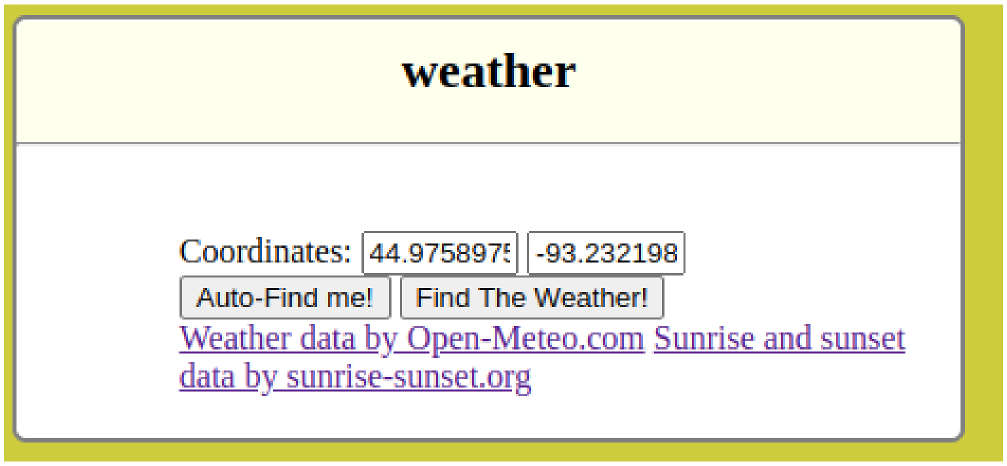
69 Days 11 Hours 43 Minutes 56 Seconds

weather

Coordinates: Latitude Longitude

Weather data by Open-Meteo.com Sunrise and sunset data by sunrise-sunset.org

Widgets page with additions

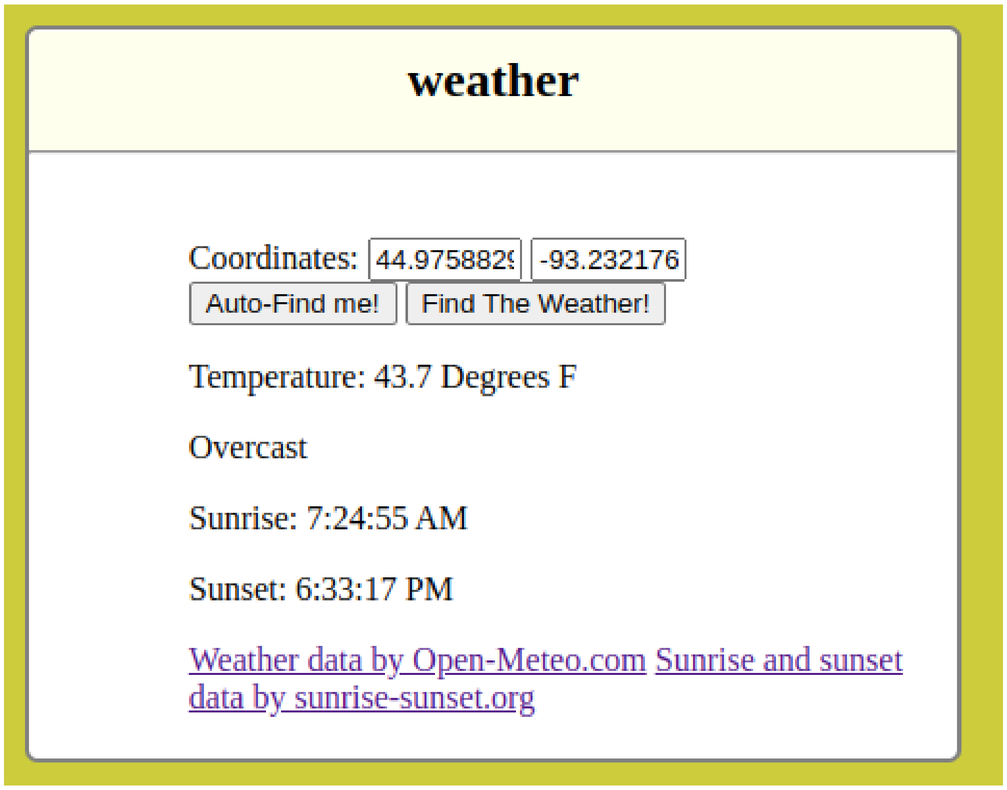

weather

Coordinates: 44.975897 -93.232198

Auto-Find me! Find The Weather!

Weather data by Open-Meteo.com Sunrise and sunset data by sunrise-sunset.org

The weather widget after clicking on the “Auto-Find me!” button


weather

Coordinates: 44.975882 -93.232176

Auto-Find me! Find The Weather!

Temperature: 43.7 Degrees F

Overcast

Sunrise: 7:24:55 AM

Sunset: 6:33:17 PM

Weather data by Open-Meteo.com Sunrise and sunset data by sunrise-sunset.org

The weather widget after clicking the “Find The Weather!” button. Note – you’ve been given the freedom to layout this information differently if you think this doesn’t look good Just make sure you have the right information available.

5 Grading information and requirements recap

- (20 points) Overall (and other minor changes)
  - (6 points) Autograder. This will check if files are structured correctly in the submission. We will override these upon manual review, but we currently cannot foresee a situation where you fail these tests and can't fix it.
  - (0 points) Manual review of the file organization. *sometimes* the tests fail, so we'll double check anyone who fails tests.
  - (4 points) adding the new page link on all 4 pages.
  - (3 points) Each page passes HTML validation without errors (except those caused by youtube/twitter widgets)
  - (2 points) Each page passes CSS validation without errors (except those caused by youtube/twitter widgets)
  - (5 points) All CSS/JS/HTML code is human readable well-tabbed, and well-formatted per their language-specific code-style norms and standards.
- (20 points) Contacts Sorting
  - (3 points) all main javascript code for the contacts table is in myContacts.js
  - (5 points) clicking the name field resorts the table by name.
  - (5 points) clicking the email field resorts the table by email.
  - (EC 3 points) All fields are sortable using a single javascript function
  - (3 points) after resorting the behaviors from HW2 still work.
  - (4 points) sorting is repeatable and can be swapped between sorts without issue.
- (30 points) Contact Me page
  - (4 points) Page is named correctly and has appropriate visual style. A main visual element marks the form which is 50% screen width and centered. A header element marks the form and is centered.
  - (3 points) The form element has correct configuration to send data to our testing backend server
  - (9 points) All 6 editable fields are correctly encoded using reasonable html elements and correct attributes for sending data to our testing backend server.
  - (2 points) radio buttons are correctly configured and exclusive with each other – also they are in a fieldset.
  - (3 points) title, email, username, category and message are required the form can't be submitted without them.
  - (3 points) email and webpage validation is correct/reasonable and the form can't be submitted if they are invalid.
  - (3 points) username suggestion feature works as documented
  - (3 points) concern notification message works as documented
- (30 points) Weather Widget
  - (3 points) Widget is in-place in the widgets page in a reasonable place, widget auto-find me feature still works, and all code is in the correct javascript file.
  - (3 points) Requests to open-meteo and sunrise-sunset APIs execute concurrently with each other.

- (12 points) open-meteo data is requested correctly, processed reasonably, and put into the widget in a reasonable way.
  - Correct API URL is built
  - fetch API is used correctly to get json object
  - json is processed correctly to get weather and status code
  - data is output in a reasonable format.
  - Basic error handling is present (small issues allowed here as this code is hard-to-test)
- (12 points) sunrise-sunset data is requested correctly, processed reasonably, and put into the widget in a reasonable way.
  - Correct API URL is built
  - fetch API is used correctly to get json object
  - json is processed correctly to get sunrise and sunset time
  - data is output in a reasonable format with no timezone issues.
  - Basic error handling is present (small issues allowed here as this code is hard-to-test)

6 Submission information.

NOTE – with the autograder we have some enhanced requirements

Your process for submitting should be as follows:

- Check your work against the grading rubric above, or if you want to be more careful, against the requirements listed earlier in the PDF.
- Open the folder that serves as the root of your website.
- select all files and zip those files together (rather than zipping the main folder)
- Upload it to gradescope
- If you're failing tests, check to understand why – it's 100% of the time going to be either files that are not named as expected, or an issue in how you prepared your zip file.
- Resubmit as many times as you need.
- Download your submission from gradescope and unzip it. Make sure everything still works and that you haven't accidentally introduced relative URLs into your assignment.