PER SCHOLAS

ALAB 316.4.1:
Form Validation

Version 1.0, 6/26/23
Click here to open in a separate window.

Introduction

This lab provides you with instructions for implementing a variety of validation requirements using a combination of built-in HTML validation and DOM event-driven JavaScript validation. It is up to you to decide which is the best tool for each job.

Objectives

- Create robust form validation using built-in HTML validation attributes and DOM event-driven JavaScript validation.

Resources

This lab uses CodeSandbox. If you are unfamiliar with CodeSandbox, or need a refresher, please visit our reference guide on CodeSandbox for instructions on:

- Creating an Account.
- Making a Sandbox.
- Navigating a Sandbox.
- Submitting a Sandbox link to Canvas.

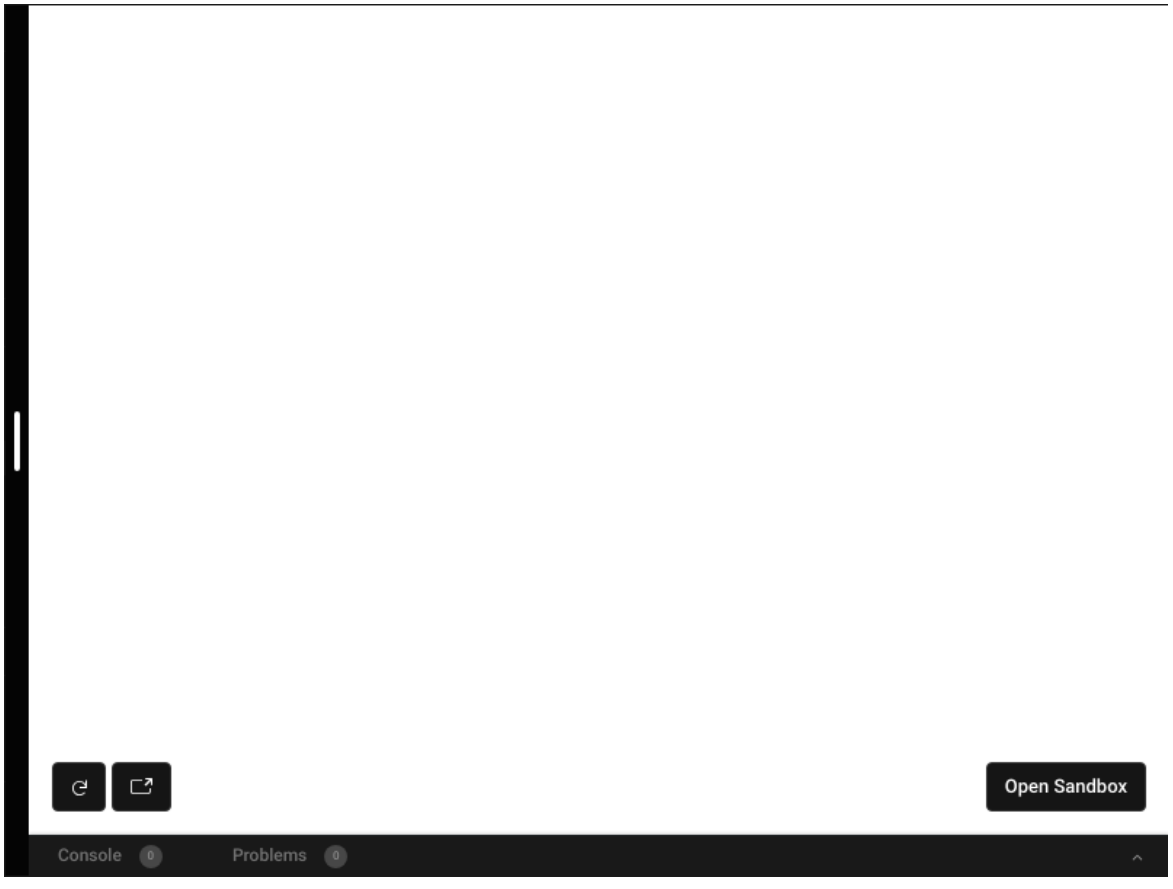
Submission

Submit your completed lab using the **Start Assignment** button on the assignment page in Canvas.

Instructions

You will begin with a pre-configured CodeSandbox:

- **ALAB Form Validation**



Fork the CodeSandbox above to get started.

Your goal in this lab is to demonstrate your knowledge of form validation techniques using the variety of tools available to you. Working with user-entered data is a cornerstone of web development, so it is important to understand how you can control and verify that data.

Part 1: Introduction

This lab provides a CodeSandbox, linked above, with a pre-built and pre-styled HTML register/login page that contains two separate forms.

Currently, these forms have no validation! Your job is to add validation so that the forms adhere to the requirements outlined below. You can choose to implement this validation using any combination of HTML validation attributes and JavaScript event listeners that you want, as long as it meets the requirements.

Explore the HTML structure that has been provided. You can make changes to the HTML (and CSS), as long as they do not subtract from the original functional intent of the page.

An HTML element with id errorDisplay has been provided as a convenient method of showing error text to the user. In order to show or hide errorDisplay, you must modify its display style attribute.

You can place any text or HTML into errorDisplay.

Part 2: General Requirements

To reiterate, these requirements can be completed using any combination of HTML validation attributes and JavaScript event listeners that you want. Consider the right tool for each job before you begin working on it.

1. **General Requirements:** Whenever any of these validation requirements fail, **an appropriate error should be communicated to the user** (in most cases, the actual requirement listed below serves as a good error message), and focus should return to the input element that the error originates from. **If any requirements fail, the form should not submit.**

Part 3: Registration Form Validation Requirements

For the Registration Form section of the page, implement the following validation requirements:

1. **Registration Form - Username Validation:**
   - The username cannot be blank.
   - The username must be at least four characters long.
   - The username must contain at least two unique characters.
   - The username cannot contain any special characters or whitespace.

2. **Registration Form - Email Validation:**
   - The email must be a valid email address.
   - The email must not be from the domain "example.com."

3. **Registration Form - Password Validation:**
   - Passwords must be at least 12 characters long.
   - Passwords must have at least one uppercase and one lowercase letter.
   - Passwords must contain at least one number.
   - Passwords must contain at least one special character.
   - Passwords cannot contain the word "password" (uppercase, lowercase, or mixed).
   - Passwords cannot contain the username.
   - Both passwords must match.

4. **Registration Form - Terms and Conditions:**
   - The terms and conditions must be accepted.

5. **Registration Form - Form Submission:**
   - Usually, we would send this information to an external API for processing. In our case, we are going to process and store the data locally for practice purposes.
   - If all validation is successful, store the username, email, and password using localStorage.
     - If you are unfamiliar with localStorage, that is okay! Reference the documentation's "Description" and "Examples" sections to learn how to implement it. If you run into issues speak with a peer or one of your instructors.
     - Consider how you want to store the user data, keeping in mind that there will be quite a few users registering for the site. Perhaps you want to store it with an array of user objects; or maybe an object whose keys are the usernames themselves.
     - **Valid usernames should be converted to all lowercase before being stored.**
     - **Valid emails should be converted to all lowercase before being stored.**
   - Clear all form fields after successful submission and show a success message.

6. **Registration Form - Username Validation (Part Two):**
   - Now that we are storing usernames, create an additional validation rule for them...
   - Usernames must be unique ("that username is already taken" error). Remember that usernames are being stored all lowercase, so "learner" and "Learner" are not unique.

Part 4: Login Form Validation Requirements

For the Login Form section of the page, implement the following validation requirements:

1. **Login Form - Username Validation:**
    - The username cannot be blank.
    - The username must exist (within localStorage). Remember that usernames are stored in all lowercase, but the username field accepts (and should not invalidate) mixed-case input.

2. **Login Form - Password Validation:**
    - The password cannot be blank.
    - The password must be correct (validate against localStorage).

3. **Login Form - Form Submission:**
    - If all validation is successful, clear all form fields and show a success message.
    - If "Keep me logged in" is checked, modify the success message to indicate this (normally, this would be handled by a variety of persistent login tools and technologies).

Part 5: Completion

Test your validation thoroughly! Try to break things!

- Remember that each successful registration should be stored; therefore you should be able to login with a variety of account credentials.
- When you are done testing your own code, swap sandboxes with a partner and test theirs!
- When each of you are finished testing, share your results.
- Discuss with your partner the differences and similarities between your two approaches. Remember that there is rarely a strictly "correct" or "incorrect" way to solve a problem in development, but there are (almost always) more efficient approaches!

Remember to submit the link to your finished sandbox using the submission instructions included at the beginning of this document.