**html**

<div id="mocha"></div>

<form class="test-form">

<input name="first-name" type="text" data-validation="alphabetical" />

<input name="age" type="text" data-validation="numeric" />

</form>

**script.js**

(function () {

'use strict';

const validationRules = new Map([

['alphabetical', /^[a-z]+$/i],

['numeric', /^[0-9]+$/]

]);

// Implementation

function createValidationQueries(inputs) {

return Array.from(inputs).map(input => ({

name: input.name,

type: input.dataset.validation,

value: input.value

}));

}

function validateItem(validation, validationRules) {

if (!validationRules.has(validation.type)) {

return false;

}

return validationRules.get(validation.type).test(validation.value);

}

function validateForm(form) {

const result = {

get isValid() {

return this.errors.length === 0;

},

errors: []

};

for (let validation of createValidationQueries(form.querySelectorAll('input'))) {

let isValid = validateItem(validation, validationRules);

if (!isValid) {

result.errors.push(

new Error(`${validation.value} is not a valid ${validation.name} value`)

);

}

}

return result;

}

// Test Setup

mocha.setup('bdd');

const { expect } = chai;

describe('the form validator', function () {

let form;

beforeEach(function () {

form = document.querySelector('.test-form').cloneNode(true);

});

describe('the validateForm function', function () {

it('should validate a form with all of the possible validation types', function () {

const name = form.querySelector('input[name="first-name"]');

const age = form.querySelector('input[name="age"]');

name.value = 'Bob';

age.value = '42';

const result = validateForm(form);

expect(result.isValid).to.be.true;

expect(result.errors.length).to.equal(0);

});

it('should return an error when a name is invalid', function () {

const name = form.querySelector('input[name="first-name"]');

const age = form.querySelector('input[name="age"]');

name.value = '!!!';

age.value = '42';

const result = validateForm(form);

expect(result.isValid).to.be.false;

expect(result.errors[0]).to.be.instanceof(Error);

expect(result.errors[0].message).to.equal('!!! is not a valid first-name value');

});

it('should return an error when an age is invalid', function () {

const name = form.querySelector('input[name="first-name"]');

const age = form.querySelector('input[name="age"]');

name.value = 'Greg';

age.value = 'a';

const result = validateForm(form);

expect(result.isValid).to.be.false;

expect(result.errors[0]).to.be.instanceof(Error);

expect(result.errors[0].message).to.equal('a is not a valid age value');

});

it('should return multiple errors if more than one field is invalid', function () {

const name = form.querySelector('input[name="first-name"]');

const age = form.querySelector('input[name="age"]');

name.value = '!!!';

age.value = 'a';

const result = validateForm(form);

expect(result.isValid).to.be.false;

expect(result.errors[0]).to.be.instanceof(Error);

expect(result.errors[0].message).to.equal('!!! is not a valid first-name value');

expect(result.errors[1]).to.be.instanceof(Error);

expect(result.errors[1].message).to.equal('a is not a valid age value');

});

});

describe('the createValidationQueries function', function () {

it(

'should map input elements with a data-validation attribute to an array of validation objects',

function () {

const name = form.querySelector('input[name="first-name"]');

const age = form.querySelector('input[name="age"]');

name.value = 'Bob';

age.value = '42';

const validations = createValidationQueries([name, age]);

expect(validations.length).to.equal(2);

expect(validations[0].name).to.equal('first-name');

expect(validations[0].type).to.equal('alphabetical');

expect(validations[0].value).to.equal('Bob');

expect(validations[1].name).to.equal('age');

expect(validations[1].type).to.equal('numeric');

expect(validations[1].value).to.equal('42');

}

);

});

describe('the validateItem function', function () {

const validationRules = new Map([

['alphabetical', /^[a-z]+$/i]

]);

it(

'should return true when the passed item is deemed valid against the supplied validation rules',

function () {

const validation = {

type: 'alphabetical',

value: 'Bob'

};

const isValid = validateItem(validation, validationRules);

expect(isValid).to.be.true;

}

);

it(

'should return false when the passed item is deemed invalid',

function () {

const validation = {

type: 'alphabetical',

value: '42'

};

const isValid = validateItem(validation, validationRules);

expect(isValid).to.be.false;

}

);

it(

'should return false when the specified validation type is not found',

function () {

const validation = {

type: 'foo',

value: '42'

};

const isValid = validateItem(validation, validationRules);

expect(isValid).to.be.false;

}

);

});

});

mocha.run();

}());