interface Product {

id: number; // not shown to the user, but needs to be preserved

productCode: string; // expected format: 003-5000-ABCD

productName: string; // required, trim whitespace

vendorId: number; // user needs a pick list of vendor names that map to the Guid

price: number;

activationDate: Date; // date we started selling. Always <= today

}

Walk through

Suppose that you have the following model and will be building an HTML based screen as shown below. In this example, we are doing most of the work in the browser. There is an API to get and save the data with the server, but we are responsible for the rest in the client. We’ll use jQuery syntax to simplify manipulation of HTML.

interface Product {

id: number; // not shown to the user, but needs to be preserved

productCode: string; // expected format: 003-5000-ABCD

productName: string; // required, trim whitespace

vendorId: number; // user needs a pick list of vendor names that map to the Guid

price: number;

activationDate: Date; // date we started selling. Always <= today

}

A screenshot of a computer

Description automatically generated

Let’s suppose that we’ve decided each property’s editor row has this HTML structure:

<div><label for="use the id of the editor">label</label></div>

<div>the appropriate editor</div>

<div class="errorMessageArea" id="Error + use the id of the editor "></div>

Let’s also gather validators needed for each property:

Property name required Data Type Check Additional

productCode yes "^\d{3}\-\d{4}\-[A-Z]{4}$"

productName yes

vendor yes

price yes Must convert to a number Greater or equal to 0

activated no Must convert to a date Less than or equal to today’s date

What are the actions needed to get this to work?

1. Create all of the HTML, using <input> and <select> tags depending on the field type.
2. Give each <input> and <select> tag a name attribute that we associate with the model property. For example: <input id="activationDate" type="date" name="activationDate" />
3. Write code that converts a model property value to a string for use with the HTML tag’s value attribute. For example, the activationDate is a Date object and needs to be a localized string representing a date that is compatible with <input type="date">.

let adValue = Intl.DateTimeFormat("en-US", {

year: 'numeric',

month: 'numeric',

day: 'numeric'

}).format(myProduct.activationDate);

1. Assign the string to the value attribute.

$("#activationDate").value = adValue;

1. Setup style sheet classes that are used for when there are errors.

.inputError {

border-color: red;

}

. errorMessageArea

{

display: none;

}

.errorMessageArea.validationError

{

display: inline;

color: red;

padding: 2px;

}

1. Create showError and clearError functions.

function showError(idToEditor, errorMessage)

{

$("#" + idToEditor).addClass("inputError");

$("#Error" + idToEditor).addClass("validationError").append(

$("<span class="error">).text(errorMessage));

}

function clearError(idToEditor)

{

$("#" + idToEditor).removeClass("inputError");

$("#Error" + idToEditor).removeClass("validationError").clear();

}

1. Monitor the onChange event of the <input> and <select> elements. The function does a lot of work:
   1. Call clearError()
   2. Get the value still as a string:

adValue = adElement.value;

* 1. If the field is required, trim the string value and call showError() if the length is 0. Then stop.
  2. Confirm the input is compatible with the Model property. This is called a Data Type Check.