UT07.2: Validación con Frameworks

Contenido

1. Frameworks

Hasta ahora hemos creado toda la funcionalidad basándonos en APIs nativas. Como hemos visto, programando podemos implementar cualquier función que deseemos, aunque el trabajo es bastante tedioso, teniendo que realizar una codificación a medida de cada función. En el mercado existen multitud de frameworks, plugins y herramientas que nos facilitan el tratamiento de los formularios al realizar toda la función de una forma declarativa. Veamos algunos ejemplos.

2. jQuery Mask Plugin

Existe una enorme biblioteca de plugins construidos sobre jQuery que nos pueden facilitar mucho implementar determinadas funcionalidades. La lista la podemos consultar aquí https://plugins.jquery.com/ .

Un ejemplo es jQuery Mask Plugin, el cual nos facilita muchísimo la vida a la hora de crear máscaras para formularios, ya que nos evita la codificación personalizada de cada máscara. Se encuentra disponible en https://igorescobar.github.io/jQuery-Mask-Plugin/

2.1. Uso del plugin

Para instanciarlo tan solo necesitamos importar su librería, lo cual haremos utilizando un CDN.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.mask/1.14.16/jquery.mask.js"
    integrity="sha256-yE5LLp5HSQ/z+hJeCqkz9hdjNkk1jaiGG0tDCraumnA=" crossorigin="anonymous">
    </script>
```

El plugin trabaja en base a expresiones regulares predefinidas. Los caracteres que podemos utilizar en las expresiones regulares son:

- 0: Representa un dígito.
- 9: Representa un dígito opcional.
- #: Representa un dígito que se repetirá recursivamente.
- A: Representa una carácter minúscula, mayúscula o dígito.
- S: Representa un carácter minúscula o mayúscula.

Vamos a replicar las máscaras para DNI, teléfono y cuenta corriente que hemos creado en el documento anterior. Primero vamos a definir el formulario sobre el que vamos a trabajar.

```
const form = document.forms.fValidation;
form.setAttribute('novalidate', '');

$(form).submit(function (event){
  event.preventDefault();
  event.stopPropagation();
});
```

El formulario solamente consta de los tres elementos que incluirán la máscara, y hemos deshabilitado el evento submit.

Para crear una máscara basta con invocar el método `mask()` sobre un elemento del formulario. Para el DNI sería de la siguiente forma.

```
$(form.vfDni).mask('00000000S');
```

Podemos crear patrones para nuevos caracteres, por ejemplo, el carácter + del teléfono. Lo hacemos pasando un objeto literal con la propiedad *translation* con los nuevos patrones.

```
$(form.vfPhone).mask('(P00) 000 00 00 00', {'translation': {
  P: {pattern: /(\+)/}
}});
```

Para la cuenta corriente, el objeto literal indica que si el dato no cumple con el patrón se elimine, además de indicar un placeholder.

```
$(form.vfCreditCard).mask('0000 0000 0000 0000', {
  clearIfNotMatch: true,
  placeholder: '0000 0000 0000 0000'
});
```

2.2. Máscaras dinámicas

Podemos cambiar el tipo de máscara dinámicamente en función de cómo introduzcamos los caracteres en el elemento. La siguiente función recibe el valor actual del elemento, el evento que se ha disparado y el elemento sobre el cual está actuando la máscara. En función del valor, la función devuelve un tipo de máscara diferente para ajustarse a qué prefijo estamos utilizando en el teléfono.

```
function PhoneMaskBehavior (val, e, elem) {
  let mask = '(I00) 000 00 00 00';
  if (/^\(\+/.test(val)){
    mask = '(P00) 000 00 00 00';
  } else if (/^\(0/.test(val)){
    mask = '(ZZ00) 000 00 00 00';
  } else if (/^(\()?[69]/.test(val)){
    mask = 'N00 00 00 00';
  }
  return mask;
}
```

Modificamos la máscara del teléfono para crearla dinámicamente. Al método le pasamos la función como argumento, y en el objeto literal configuramos los caracteres que vamos a utilizar en la máscara. En el objeto podemos añadirle eventos, en este caso el evento `onKeyPress` invocará la función para cambiar de máscara cada vez que se pulse una tecla. El evento `onComplete` se ejecutará cuando la máscara haya quedado satisfecha. En este segundo evento debemos validar el resultado final.

```
$(form.vfPhone).mask(PhoneMaskBehavior, {'translation': {
  'I': {pattern: /[\+069]/},
  'P': {pattern: /[\+]/},
  'Z': {pattern: /0/},
  'N': {pattern: /[69]/},
}, onKeyPress: function(val, e, elem, options) {
  if (elem.val().length < 3) elem.mask(PhoneMaskBehavior.apply({}, argume
nts), options);
}, onComplete: function(val, e, elem, options) {
  let mask = '(P00) 000 00 00 00';
  if (/^\(\+/.test(val)){
```

```
    mask = '(P00) 000 00 00 00';
  } else if (/^\(0/.test(val)){
    elem.val(elem.val().replace(/^\(00/,'(+'));
  } else if (/^(\()?[69]/.test(val)){
    elem.val('(+34)' + elem.val());
  }
  elem.mask(mask, options);
  maskCheckElement(e, elem);
}});
```

La función `maskCheckElement()` y `showFeedBack()` las utilizamos para validar el valor y mostrar la retroalimentación.

```
function maskCheckElement (event, elem) {
  elem.val(elem.val().trim());
  if (!elem.get(0).checkValidity()) {
    showFeedBack(elem, false);
  } else {
    showFeedBack(elem, true);
  }
}

function showFeedBack(input, valid, message) {
  let validClass = (valid) ? 'is-valid' : 'is-invalid';
  let div = (valid) ? input.nextAll("div.valid-
feedback") : input.nextAll("div.invalid-feedback");
  input.nextAll('div').removeClass('d-block');
  div.removeClass('d-none').addClass('d-block');
  input.removeClass('is-valid is-invalid').addClass(validClass);
  if (message) {
    div.empty();
    div.append(message);
  }
}
```

3. Validación con Boostrap

Si estamos utilizando un framework como Bootstrap, podemos crear una validación prácticamente de forma declarativa. Necesitamos añadir el atributo `novalidate` al formulario, y una clase `needs-validation`. Para mostrar la retroalimentación, debemos añadir en el evento submit la clase `was-validated`, esto lo hacemos en el manejador del evento. Como vemos, este manejador utiliza el método `checkValidity()` del formulario para comprobar que todos los valores están conforme a la declaración, en caso contrario cancela el envío del formulario.

```
let form = document.forms.fBootstrapValidation;
form.setAttribute('novalidate', '');
form.classList.add('needs-validation');

form.addEventListener('submit', (event) => {
  if (!form.checkValidity()) {
    event.preventDefault();
    event.stopPropagation();
  }
  form.classList.add('was-validated');
}, false);
```