UT07.3: MVC

Contenido

1. Introducción

Para seguir evolucionando nuestro proyecto, vamos a crear formularios desde los cuales podamos actualizar el *Modelo*.

2. Administración

Vamos a crear un menú en la cabecera denominado *administración* con los enlaces que nos den acceso a los formularios de creación y eliminación de objetos del *Modelo*.

Creamos el método en la *Vista* para generar el menú con las posibles opciones.

```
showAdminMenu() {
  const menuOption = document.createElement('li');
  menuOption.classList.add('nav-item');
  menuOption.classList.add('dropdown');
  menuOption.insertAdjacentHTML(
    'afterbegin',
    '<a class="nav-link dropdown-toggle" href="#" id="adminMenu"
role="button" data-bs-toggle="dropdown" aria-expanded="false">
Adminitración</a>',
  );
  const suboptions = document.createElement('ul');
  suboptions.classList.add('dropdown-menu');
  suboptions.insertAdjacentHTML('beforeend', '<li><a id="lnewCategory"
class="dropdown-item" href="#new-category">Crear categoría</a></li>');
  suboptions.insertAdjacentHTML('beforeend', '<li><a id="ldelCategory"
class="dropdown-item" href="#del-category">Eliminar categoría</a></li>');
  suboptions.insertAdjacentHTML('beforeend', '<li><a id="lnewProduct"
class="dropdown-item" href="#new-product">Crear producto</a></li>');
  suboptions.insertAdjacentHTML('beforeend', '<li><a id="ldelProduct"
class="dropdown-item" href="#del-product">Eliminar producto</a></li>');
  menuOption.append(suboptions);
  this.menu.append(menuOption);
}
```

El menú debe generarse al cargar la página, por lo que invocamos el método en el evento `onLoad()` del *Controlador*, mostrándose en la página el nuevo menú.

```
this[VIEW].showAdminMenu();
```

3. Categorías

Vamos a añadir dos opciones para crear y eliminar categorías del *Manager*.

3.1. Crear una categoría

Implementamos por fases esta funcionalidad.

3.1.1. Generación del formulario

Creamos un método para la generación del formulario. El método no necesita argumentos porque no muestra información alguna del *Modelo*.

```
showNewCategoryForm() {
  this.main.replaceChildren();
  if (this.categories.children.length > 1)
this.categories.children[1].remove();

  const container = document.createElement('div');
  container.classList.add('container');
  container.classList.add('my-3');
  container.id = 'new-category';

  container.insertAdjacentHTML(
    'afterbegin',
    '<h1 class="display-5">Nueva categoría</h1>',
  );
  container.insertAdjacentHTML(
    'beforeend',
    `<form name="fNewCategory" role="form" class="row g-3" novalidate>
    <div class="col-md-6 mb-3">
      <label class="form-label" for="ncTitle">Título *</label>
      <div class="input-group">
        <span class="input-group-text"><i class="bi bi-type"></i></span>
        <input type="text" class="form-control" id="ncTitle"
name="ncTitle"
          placeholder="Título de categoría" value="" required>
        <div class="invalid-feedback">El título es obligatorio.</div>
        <div class="valid-feedback">Correcto.</div>
      </div>
    </div>
    <div class="col-md-6 mb-3">
      <label class="form-label" for="ncUrl">URL de la imagen *</label>
      <div class="input-group">
        <span class="input-group-text"><i class="bi bi-file-
image"></i></span>
        <input type="url" class="form-control" id="ncUrl" name="ncUrl"
placeholder="URL de la imagen"
          value="" required>
        <div class="invalid-feedback">La URL no es válida.</div>
        <div class="valid-feedback">Correcto.</div>
      </div>
    </div>
    <div class="col-md-12 mb-3">
      <label class="form-label" for="ncDescription">Descripción</label>
      <div class="input-group">
        <span class="input-group-text"><i class="bi bi-body-
text"></i></span>
        <input type="text" class="form-control" id="ncDescription"
name="ncDescription" value="">
```

```
        <div class="invalid-feedback"></div>
        <div class="valid-feedback">Correcto.</div>
      </div>
    </div>
    <div class="mb-12">
      <button class="btn btn-primary" type="submit">Enviar</button>
      <button class="btn btn-primary" type="reset">Cancelar</button>
    </div>
  </form>`,
  );
  this.main.append(container);
}
```

Tenemos que vincular el enlace del menú con el manejador que genera el formulario mediante un método *bind*. El método recibe el manejador como argumento.

```
bindAdminMenu(hNewCategory) {
  const newCategoryLink = document.getElementById('lnewCategory');
  newCategoryLink.addEventListener('click', (event) => {
    this[EXCECUTE_HANDLER](hNewCategory, [], '#new-category', { action:
'newCategory' }, '#', event);
  });
}
```

El manejador invoca el método de la vista para crear el formulario.

```
handleNewCategoryForm = () => {
  this[VIEW].showNewCategoryForm();
};
```

Por último, invocamos al método *bind* justo después de invocar el método de generación del menú en el evento `onLoad()` del *Controlador* para que se pueda mostrar el formulario.

```
this[VIEW].bindAdminMenu(this.handleNewCategoryForm);
```

3.1.2. Submit y validación del formulario

Vamos a crear un modal para indicarle al usuario que la categoría se ha creado correctamente. En primer lugar, vamos a crear un script denominado **validation.js** y lo integramos como un módulo más en el manager. De momento tendremos dos funciones auxiliares para mostrar la retroalimentación y la validación por defecto de los elementos de un formulario, tal y como hicimos en los ejemplos de validación.

```
function showFeedBack(input, valid, message) {
  const validClass = (valid) ? 'is-valid' : 'is-invalid';
  const messageDiv = (valid) ?
input.parentElement.querySelector('div.valid-feedback') :
input.parentElement.querySelector('div.invalid-feedback');
  for (const div of input.parentElement.getElementsByTagName('div')) {
    div.classList.remove('d-block');
  }
  messageDiv.classList.remove('d-none');
  messageDiv.classList.add('d-block');
  input.classList.remove('is-valid');
  input.classList.remove('is-invalid');
  input.classList.add(validClass);
  if (message) {
    messageDiv.innerHTML = message;
  }
}

function defaultCheckElement(event) {
  this.value = this.value.trim();
  if (!this.checkValidity()) {
    showFeedBack(this, false);
  } else {
    showFeedBack(this, true);
  }
}
```

Tenemos que enlazar el botón de submit del formulario con el manejador que permita añadir la categoría al *Manager*. Este método recibe el manejador y se lo va a pasar a una función externa donde implementamos toda la validación. En este caso la funcionalidad no implica guardarse en el histórico.

```
bindNewCategoryForm(handler){
  newCategoryValidation(handler);
}
```

Este método debe ser invocado justo después de generar el formulario.

```
this[VIEW].bindNewCategoryForm(this.handleCreateCategory);
```

En la función externa tenemos toda la validación del formulario. Recibe como argumento el manejador que será invocado si tenemos éxito en la validación. El submit del formulario hace la validación de los elementos, pero su comportamiento por defecto es cancelado, aunque sea correcta ya que no debemos perder la página actual. En caso de tener éxito en la validación, invocaremos el manejador pasándole todos los datos recogidos del formulario que nos permite crear un objeto de tipo `Category`.

También tratamos el evento `reset` del formulario para restaurar valores y eliminar la retroalimentación, además tenemos en cuenta la validación en línea.

```
function newCategoryValidation(handler) {
  const form = document.forms.fNewCategory;
  form.setAttribute('novalidate', true);
  form.addEventListener('submit', function (event) {
    let isValid = true;
    let firstInvalidElement = null;
    this.ncDescription.value = this.ncDescription.value.trim();
    showFeedBack(this.ncDescription, true);
    if (!this.ncUrl.checkValidity()) {
      isValid = false;
      showFeedBack(this.ncUrl, false);
      firstInvalidElement = this.ncUrl;
    } else {
      showFeedBack(this.ncUrl, true);
    }
    if (!this.ncTitle.checkValidity()) {
      isValid = false;
      showFeedBack(this.ncTitle, false);
      firstInvalidElement = this.ncTitle;
    } else {
      showFeedBack(this.ncTitle, true);
    }
    if (!isValid) {
      firstInvalidElement.focus();
    } else {
      handler(this.ncTitle.value, this.ncUrl.value,
this.ncDescription.value);
    }
    event.preventDefault();
    event.stopPropagation();
  });

  form.addEventListener('reset', (function (event) {
    for (const div of this.querySelectorAll('div.valid-feedback,
div.invalid-feedback')) {
      div.classList.remove('d-block');
      div.classList.add('d-none');
    }
    for (const input of this.querySelectorAll('input')) {
      input.classList.remove('is-valid');
      input.classList.remove('is-invalid');
    }
    this.ncTitle.focus();
  }));
  form.ncTitle.addEventListener('change', defaultCheckElement);
  form.ncUrl.addEventListener('change', defaultCheckElement);
}
```

Exportamos la función en el módulo de *validación*.

```
export { newCategoryValidation };
```

Y la importamos en la vista del **manager**.

```
import {newCategoryValidation} from './validation.js';
```

El manejador en el *Controlador* recibe los datos del formulario para crear un objeto `Category` y añadirlo al *Manager*. En una variable booleana recoge si tiene éxito, e invocará a la *Vista* para que el usuario sea informado. El manejador invoca el método de la *Vista* pasándole el booleano, la categoría creada y la posible excepción con el error.

```
handleCreateCategory = (title, url, desc) => {
  const cat = this[MODEL].getCategory(title, url);
  cat.description = desc;

  let done; let
    error;
  try {
    this[MODEL].addCategory(cat);
    done = true;
  } catch (exception) {
    done = false;
    error = exception;
  }
  this[VIEW].showNewCategoryModal(done, cat, error);
};
```

El método en la *Vista* generará el modal en caso de tener éxito, o una capa indicando el error dentro del formulario si la categoría ya existe. Añadimos un modal en HTML en final del *index.html*. Este modal permanecerá oculto.

```
<div class="modal fade" id="messageModal" data-bs-backdrop="static" data-
bs-keyboard="false" tabindex="-1"
    aria-labelledby="messageModalTitle" aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered modal-dialog-
scrollable">
        <div class="modal-content">
            <div class="modal-header">
                <h1 class="modal-title fs-5" id="messageModalTitle">Modal
title</h1>
                <button type="button" class="btn-close" data-bs-
dismiss="modal" aria-label="Close"></button>
            </div>
            <div class="modal-body">...</div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Cerrar</button>
            </div>
        </div>
    </div>
</div>
```

Generamos el método en la vista que cumplimenta el modal. Utilizamos los objetos de Bootstrap, instanciamos un modal y lo mostramos, también añadimos un manejador sobre el evento de Bootstrap `hidden.bs.modal` para que gestione el formulario al ocultarse el modal. Este manejador solo debe ejecutarse una única vez.

```
showNewCategoryModal(done, cat, error) {
  const messageModalContainer = document.getElementById('messageModal');
  const messageModal = new bootstrap.Modal('#messageModal');

  const title = document.getElementById('messageModalTitle');
  title.innerHTML = 'Nueva Categoría';
  const body = messageModalContainer.querySelector('.modal-body');
  body.replaceChildren();
  if (done) {
    body.insertAdjacentHTML('afterbegin', `<div class="p-3">La categoría
<strong>${cat.title}</strong> ha sido creada correctamente.</div>`);
  } else {
    body.insertAdjacentHTML(
      'afterbegin',
      `<div class="error text-danger p-3"><i class="bi bi-exclamation-
triangle"></i> La categoría <strong>${cat.title}</strong> ya está
creada.</div>`,
    );
  }
  messageModal.show();
  const listener = (event) => {
    if (done) {
      document.fNewCategory.reset();
    }
    document.fNewCategory.ncTitle.focus();
  };
  messageModalContainer.addEventListener('hidden.bs.modal', listener, {
once: true });
}
```

3.1.3. Actualización del menú de categorías

Al crear una categoría debemos actualizar el menú de categorías. De momento lo haremos desde el manejador `handleCreateCategory()` invocando el evento `onAddCategory()` en el caso de crearse la categoría.

```
this.onAddCategory();
```

Tenemos que hacer unos cambios sobre la versión anterior para acometer esta funcionalidad. Modificamos el index.html para añadir una nueva opción de menú para las categorías con la estructura del desplegable sin las subopciones, en lugar de reconstruirlo en la vista.

```
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle show" href="#" id="navCats"
role="button"
        data-bs-toggle="dropdown" aria-expanded="true">Categorías</a>
    <ul class="dropdown-menu" data-bs-popper="static">
    </ul>
</li>
```

Actualizamos `showCategoriesInMenu()` para que solamente genere las subopciones y que se pueda regenerar en base al evento.

```
showCategoriesInMenu(categories) {
  const navCats = document.getElementById('navCats');
  const container = navCats.nextElementSibling;
  container.replaceChildren();
  for (const category of categories) {
    container.insertAdjacentHTML('beforeend', `<li><a data-
category="${category.title}" class="dropdown-item" href="#product-
list">${category.title}</a></li>`);
  }
}
```

Tenemos una errata en `bindProductsCategoryListInMenu()`, debemos utilizar la propiedad `nextElementSibling` al tener nodos de texto en el código HTML de la página.

```
const links = navCats.nextElementSibling.querySelectorAll('a');
```

3.2. Eliminar una categoría

Los pasos son similares a la creación de una categoría.

3.2.1. Generación del formulario

Creamos un método para la generación del formulario. El método debe recibir el iterador con las categorías para poder elegir cuál queremos eliminar mediante un botón. La información de la categoría a borrar la tomaremos de los atributos personalizados del botón.

```
showRemoveCategoryForm(categories) {
  this.main.replaceChildren();
  if (this.categories.children.length > 1)
this.categories.children[1].remove();
  const container = document.createElement('div');
  container.classList.add('container');
  container.classList.add('my-3');
  container.id = 'remove-category';
  container.insertAdjacentHTML(
    'afterbegin',
    '<h1 class="display-5">Eliminar una categoría</h1>',
  );
  const row = document.createElement('div');
  row.classList.add('row');
  for (const category of categories) {
    row.insertAdjacentHTML('beforeend', `<div class="col-lg-3 col-md-6">
      <div class="cat-list-image"><img alt="${category.title}"
src="${category.url}" />
      </div>
      <div class="cat-list-text">
        <a data-category="${category.title}" href="#category-
list"><h3>${category.title}</h3></a>
        <div>${category.description}</div>
      </div>
      <div><button class="btn btn-primary" data-
category="${category.title}" type='button'>Eliminar</button></div>
  </div>`);
  }
  container.append(row);
  this.main.append(container);
}
```

En el método `bindAdminMenu()` debemos vincular el enlace del menú con el manejador que invoque la vista del formulario de eliminación. Añadimos el parámetro `hRemoveCategory` que será el manejador. Vinculamos el enlace con el manejador.

```
bindAdminMenu(hNewCategory, hRemoveCategory) {
  const newCategoryLink = document.getElementById('lnewCategory');
  newCategoryLink.addEventListener('click', (event) => {
    this[EXCECUTE_HANDLER](hNewCategory, [], '#new-category', { action:
'newCategory' }, '#', event);
  });
  const delCategoryLink = document.getElementById('ldelCategory');
  delCategoryLink.addEventListener('click', (event) => {
    this[EXCECUTE_HANDLER](hRemoveCategory, [], '#remove-category', {
action: 'removeCategory' }, '#', event);
  });
}
```

En la invocación del *bind* en el evento *onLoad* tenemos que pasar como argumento el manejador para generar la vista.

```
this[VIEW].bindAdminMenu(this.handleNewCategoryForm,
this.handleRemoveCategoryForm);
```

Creamos el manejador pasando como argumento el listado de categorías para saber cuál tenemos que eliminar.

```
handleRemoveCategoryForm = () => {
  this[VIEW].showRemoveCategoryForm(this[MODEL].categories);
};
```

3.2.2. Eliminación de la categoría

Ya disponemos de un modal para mostrar los mensajes al eliminar la categoría. La retroalimentación con el usuario es fundamental. Creamos una vista para cumplimentar el modal con la información de borrado.

```
showRemoveCategoryModal(done, cat, error) {
  const messageModalContainer = document.getElementById('messageModal');
  const messageModal = new bootstrap.Modal('#messageModal');

  const title = document.getElementById('messageModalTitle');
  title.innerHTML = 'Borrado de categoría';
  const body = messageModalContainer.querySelector('.modal-body');
  body.replaceChildren();
  if (done) {
    body.insertAdjacentHTML('afterbegin', `<div class="p-3">La categoría
<strong>${cat.title}</strong> ha sido eliminada correctamente.</div>`);
  } else {
    body.insertAdjacentHTML(
      'afterbegin',
      `<div class="error text-danger p-3"><i class="bi bi-exclamation-
triangle"></i> La categoría <strong>${cat.title}</strong> no se ha podido
borrar.</div>`,
    );
  }
  messageModal.show();
  const listener = (event) => {
    if (done) {
      const removeCategory = document.getElementById('remove-category');
      const button = removeCategory.querySelector(`button.btn[data-
category="${cat.title}"]`);
      button.parentElement.parentElement.remove();
    }
  };
  messageModalContainer.addEventListener('hidden.bs.modal', listener, {
once: true });
}
```

Como podemos ver, estamos capturando el `hidden.bs.modal` de Bootstrap que se dispara cuando cerramos en el modal. Para evitar recargar la vista, en el manejador de eventos borramos la capa que contiene la categoría que estamos eliminando, a partir del atributo personalizado con el título de la categoría.

Tenemos que vincular a cada botón del enlace de categorías el manejador en el *Controlador* que borra la categoría. Al manejador le pasamos el título de la categoría recogida del `dataset` añadido al botón. Además, vamos a dotar de navegabilidad al enlace con el título de la categoría para mostrar sus productos.

```
bindRemoveCategoryForm(delHandler, getCategoryHandler) {
  const removeContainer = document.getElementById('remove-category');
  const buttons = removeContainer.getElementsByTagName('button');
  for (const button of buttons) {
    button.addEventListener('click', function (event) {
      delHandler(this.dataset.category);
    });
  }
  const categoryLinks = removeContainer.querySelectorAll('a[data-
category]');
  for (const link of categoryLinks) {
    link.addEventListener('click', (event) => {
      this[EXCECUTE_HANDLER](
        getCategoryHandler,
        [link.dataset.category],
        '#product-list',
        { action: 'productsCategoryList', category: link.dataset.category
},
        '#category-list',
        event,
      );
    });
  }
}
```

En el manejador `handleRemoveCategoryForm()` invocamos la vinculación una vez generada la vista del formulario.

```
this[VIEW].bindRemoveCategoryForm(this.handleRemoveCategory,
this.handleProductsCategoryList);
```

Pasamos como argumento dos manejadores. El primero el encargado de realizar el borrado. El segundo el que nos permite implementar la navegación, para ello podemos reutilizar `handleProductsCategoryList()` que permite obtener los productos a partir del título de la categoría.

Definimos el manejador que realiza el borrado.

```
handleRemoveCategory = (title) => {
  let done; let error; let
    cat;
  try {
    cat = this[MODEL].getCategory(title);
    this[MODEL].removeCategory(cat);
    done = true;
    this.onAddCategory();
  } catch (exception) {
    done = false;
    error = exception;
  }
  this[VIEW].showRemoveCategoryModal(done, cat, error);
};
```

3.3. Rehacer el historial de navegación

Para contemplar la navegación por los formularios añadimos dos propiedades al objeto `historyActions` en **mitienda.js**.

```
newCategory: () =>  ManagerApp.handleNewCategoryForm(),
removeCategory: () => ManagerApp.handleRemoveCategoryForm(),
```