UT06.2: Ejemplo Manager

Contenido

1. Manipulación de `history`

Para manipular el objeto `history` necesitamos realizar dos acciones:

- Apilar entradas en el objeto cada vez que cliquemos en un enlace.
- Recuperar el estado al cliquear en los botones de atrás y adelante.

1.1. Apilar entradas

En la *Vista* creamos un **método privado** encargado de ejecutar el manejador asociado al evento. Sus objetivos son **situar el scroll en el elemento contenedor de la vista** de la función que estamos ejecutando porque tenemos que **cancelar la acción por defecto** que realiza el enlace. Por último, el método **apila la nuestra entrada** en el objeto `history`, por lo que le debemos pasar un **objeto literal que nos permita restaurar el estado** y la **url** que vamos a asignar al enlace.

Los argumentos que recibe son la función manejadora del *Controlador*, los argumentos que recibirá el manejador empaquetados en un array, por lo que utilizamos un operador *spread* para invocar el manejador, la cadena que nos permite seleccionar el objeto contenedor de la *Vista*, el objeto con los datos de restauración del estado, la url y el objeto de evento para cancelar la acción por defecto.

Definimos un `Symbol` en las vistas del *Manager* y *ShoppingCart* para implementar el método.

```
const EXCECUTE_HANDLER = Symbol('excecuteHandler');
```

Añadimos el método en las vistas de *Manager y en el ShoppingCart*.

```
[EXCECUTE_HANDLER](handler, handlerArguments, scrollElement, data, url,
event) {
  handler(...handlerArguments);
  const scroll = document.querySelector(scrollElement);
  console.log(scroll);
  if (scroll) scroll.scrollIntoView();
  //$(scrollElement).get(0).scrollIntoView();
  history.pushState(data, null, url);
  event.preventDefault();
}
```

En cada evento *bind* debemos invocar el método `[EXECUTE_HANDLER]()`. Un detalle importante es que para poder acceder a este método, el evento debe tener asociada una función arrow como función manejadora para que el contexto de ejecución sea el de la *Vista*.

Para los **enlaces de inicio** tenemos los siguientes eventos. Ejecutamos el método interno pasando el manejador del *Controlador* recibido, este manejador no necesita argumentos por lo que pasamos un array vacío, para el scroll restauramos el comienzo del documento con el `body` y el objeto evento para cancelar la acción por defecto del enlace.

Como entrada en `history` el objeto literal contiene una propiedad `action` que nos permite diferenciar qué acción estamos realizando, en este caso el inicio, y la url que será la de inicio también, esta url no tiene por qué existir. Modificamos por tanto el método *bindInit()* en *Manager* y *ShoppingCart*.

```
bindInit(handler) {
  document.getElementById('init').addEventListener('click', (event) => {
    this[EXCECUTE_HANDLER](handler, [], 'body', { action: 'init' }, '#',
event);
  });
  document.getElementById('logo').addEventListener('click', (event) => {
    this[EXCECUTE_HANDLER](handler, [], 'body', { action: 'init' }, '#',
event);
  });
}
```

1.2. Recuperar el estado

Tenemos que asociar el manejador al evento `popstate` que recupere el objeto con el estado, y lo haremos en el módulo *mitienda* para que la recuperación esté centralizada en un único punto. Tanto en *managerApp.js* como en *shoppingCartApp.js* tenemos exportado el objeto de la aplicación para que pueda ser utilizado desde *mitienda*.

Definimos el evento `popstate` para restaurar el estado de la página en función del tipo de acción apilada. El manejador del evento acceder al objeto literal guardado en la propiedad `state` y en base a la propiedad `action` invocar las operaciones en `ShoppingCartController` y `ManagerController` en base a la acción elegida.

```
window.addEventListener('popstate', (event) => {
  if (event.state) {
    historyActions[event.state.action](event);
  }
});
```

Para evitar tener una estructura `switch` con cada acción que tiene un mal rendimiento, lo refactorizamos en un objeto literal para invocarlo en base al nombre de la acción. En el caso de la operación de inicio debemos lanzar el método `handleInit()` de ambos controladores.

```
const historyActions = {
  init: () => {
    ManagerApp.handleInit();
    ShoppingCartApp.handleInit();
  },
};
```

1.3. Apilación del evento inicial

Debemos apilar el primer evento reemplazando la petición de inicio de la aplicación con la acción de inicio en el fichero de `mitienda.js`.

```
history.replaceState({action: 'init'}, null);
```

En este punto estamos en condiciones de testear la acción de inicio es recuperada del historial de navegación al volver atrás.

1.4. Implementación de métodos bind

Debemos redefinir cada uno de los métodos *bind* de ambas vistas para que registren las acciones en el historial a través del método `[EXECUTE_HANDLER]()`.

En *ShoppingCart*, para el enlace de acceso al carrito seguimos sin argumentos en el manejador del *Controlador*, indicamos la cadena de selección del objeto contenedor de la vista, por último, tenemos la información para restaurar el estado que indica la acción ejecutada y una url para marcar la diferencia.

```
bindShowShoppingCart(handler) {
  this.linkShoppingcart.addEventListener('click', (event) => {
    this[EXCECUTE_HANDLER](
      handler,
      [],
      '#shoppingcart-table',
      { action: 'showShoppingCart' },
      '#shoppingcart',
      event,
    );
  });
}
```

Añadimos la acción al objeto `historyActions`. En este caso mostramos la tabla resumen del carrito.

```
showShoppingCart: () => ShoppingCartApp.handleShowShoppingCart(),
```

En el *Manger,* los enlaces de los listados de las categorías los vamos a cambiar en los métodos `bindProductsCategoryList()` y `bindProductsCategoryListInMenu()` modificando el manejador de eventos para cada uno de los enlaces en el bucle. El estado que estamos guardando incluye la acción en la propiedad `action` y la categoría que hemos mostrado en la propiedad `category`.

```
link.addEventListener('click', (event) => {
  const { category } = event.currentTarget.dataset;
  this[EXCECUTE_HANDLER](
    handler,
    [category],
    '#product-list',
    { action: 'productsCategoryList', category },
    '#category-list',
    event,
  );
});
```

Nuevamente añadimos la acción al objeto `historyActions`. Como podemos ver recuperamos la categoría a listar del estado en la propiedad `category`.

```
productsCategoryList: (event) =>
ManagerApp.handleProductsCategoryList(event.state.category),
```

Para los enlaces de los tipos de productos utilizamos el mismo procedimiento, recuperamos el tipo de producto del atributo personalizado y lo pasamos como argumento para el manejador y lo guardamos en el estado en la propiedad `type`.

```
link.addEventListener('click', (event) => {
  const { type } = event.currentTarget.dataset;
  this[EXCECUTE_HANDLER](
    handler,
    [type],
    '#product-list',
    { action: 'productsTypeList', type },
    '#type-list',
    event,
  );
});
```

Registramos el tipo de acción en el objeto `historyActions`.

```
productsTypeList: (event)
=>  ManagerApp.handleProductsTypeList(event.state.type),
```

Por último, tenemos los enlaces de acceso a los productos individuales en los listados, que deben recuperar el atributo personalizado con el número de serie, lo hacemos en `bindShowProduct()`. Tenemos el manejador para los enlaces.

```
link.addEventListener('click', (event) => {
  const { serial } = event.currentTarget.dataset;
  this[EXCECUTE_HANDLER](
    handler,
    [serial],
    '#single-product',
    { action: 'showProduct', serial },
    '#single-product',
    event,
  );
});
```

Y el manejador para las imágenes.

```
image.addEventListener('click', (event) => {
  const { serial } = event.currentTarget.dataset;
  this[EXCECUTE_HANDLER](
    handler,
    [serial],
    '#single-product',
    { action: 'showProduct', serial },
    '#single-product',
    event,
  );
});
```

No debemos olvidar registrar la acción en el objeto `historyActions`.

```
showProduct: (event) => ManagerApp.handleShowProduct(event.state.serial),
```

2. Abrir producto en una ventana nueva

Vamos a añadir un botón a cada producto individual para que se abra en una ventana nueva. Tenemos que realizar modificaciones tanto en la *Vista* como en el *Controlador*.

2.1. Modificación de métodos

Debemos modificar el constructor de la *Vista* añadiendo una propiedad destinada a albergar la referencia a la nueva ventana.

```
this.productWindow = null;
```

El método `showProduct()` que genera la vista para un producto individual, añadimos un botón para abrir la ventana junto al botón de comprar. Generamos un identificador para cada botón para que sea fácilmente seleccionables.

```
<div class="cart mt-4 align-items-center">
  <button id="b-buy" data-serial="${product.serial}" class="btn btn-
primary text-uppercase mr-2 px-4">Comprar</button>
  <button id="b-open" data-serial="${product.serial}" class="btn btn-
primary text-uppercase mr-2 px-4">Abrir en nueva ventana</button>
</div>
```

2.2. Esqueleto de página de producto

En la raíz creamos un fichero **product.html** para cargar los productos individuales. El contenido lo cargaremos desde la página principal.

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Mi tienda. Web de ejemplo JS</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min
.css" rel="stylesheet"
        integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.1/font/bootstrap-icons.css">
    <!-- Google Fonts -->
    <link
href="https://fonts.googleapis.com/css?family=Open+Sans:300,400,600,700"
rel="stylesheet" />
    <!-- Animanate CSS -->
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.mi
n.css" />
```

```
    <!-- Estilos locales -->
    <link href="css/main.css" rel="stylesheet" />
</head>

<body>

    <!-- Header -->
    <header>
        <nav class="navbar navbar-expand-lg navbar-dark bg-dark bg-
gradient fixed-top"></nav>
    </header>

    <main>
        <!-- VISTAS -->
    </main>

    <footer class="bg-dark bg-gradient text-white" id="footer">
        <div class="footer-top container">
            <div class="copyright">
                © Copyright <strong><span>Mi Tienda</span></strong>.
            </div>
            <div class="credits">
            </div>
        </div>
    </footer>

    <a id="toparrow" href="#"><i class="bi bi-arrow-up-circle-
fill"></i></a>

    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.m
in.js"
        integrity="sha384-
I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+hVqc2pM8ODewa9r"
        crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.min.j
s"
        integrity="sha384-
BBtl+eGJRgqQAUMxJ7pMwbEyER4l1g+O15P+16Ep7Q9Q+zqX6gSbd85u4mG4QzX+"
        crossorigin="anonymous"></script>
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"
        integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>

</body>
```

```
</html>
```

2.3. Vista en la ventana nueva

Diseñamos una vista para el producto en la nueva página. Tenemos que tener en cuenta que este método debe modificar el documento de la nueva página, por lo que haremos referencia a él utilizando la referencia `this.productWindow`.

```
showProductInNewWindow(product, message) {
  const main = this.productWindow.document.querySelector('main');
  const header = this.productWindow.document.querySelector('header nav');
  main.replaceChildren();
  header.replaceChildren();
  let container;
  if (product) {
    this.productWindow.document.title = `${product.brand} -
${product.model}`;
    header.insertAdjacentHTML('beforeend', `<h1 data-
serial="${product.serial}" class="display-5">${product.brand} -
${product.model}</h1>`);
    container = document.createElement('div');
    container.id = 'single-product';
    container.classList.add(`${product.constructor.name}-style`);
    container.classList.add('container');
    container.classList.add('mt-5');
    container.classList.add('mb-5');
    container.insertAdjacentHTML('beforeend', `<div class="row d-flex
justify-content-center">
        <div class="col-md-10">
          <div class="card">
            <div class="row">
              <div class="col-md-12">
                <div class="images p-3">
                  <div class="text-center p-4"> <img id="main-image"
src="${product.url}"/> </div>
                </div>
              </div>
              <div class="col-md-12">
                <div class="product p-4">
                  <div class="mt-4 mb-3"> <span class="text-uppercase
text-muted brand">${product.brand}</span>
                    <h5 class="text-uppercase">${product.model}</h5>
                    <div class="price d-flex flex-row align-items-
center">
                      <span class="act-
price">${product.price.toLocaleString('es-ES', { style: 'currency',
currency: 'EUR' })}</span>
                    </div>
                  </div>
```

```
                <p class="about">${product.description}</p>
                <div class="sizes mt-5">
                  <h6 class="text-uppercase">Características</h6>
                </div>
                <div class="cart mt-4 align-items-center"> <button
data-serial="${product.serial}" class="btn btn-primary text-uppercase mr-
2 px-4">Comprar</button> </div>
              </div>
            </div>
          </div>
        </div>
      </div>`);
    container.insertAdjacentHTML('beforeend', '<button class="btn btn-
primary text-uppercase m-2 px-4"
onClick="window.close()">Cerrar</button>');
    container.querySelector('h6').insertAdjacentHTML('afterend',
this.instance[product.constructor.name](product));
    main.append(container);
  } else {
    container = document.createElement('div');
    container.classList.add('container');
    container.classList.add('mt-5');
    container.classList.add('mb-5');
    container.insertAdjacentHTML('beforeend', `<div class="row d-flex
justify-content-center">${message}</div>`);
  }
  main.append(container);
  this.productWindow.document.body.scrollIntoView();
}
```

2.4. Manejador del *Controlador*

Necesitamos un manejador en el *Controlador* para que obtenga el objeto producto del *Modelo*, e invoque el nuevo método de la *Vista*.

```
handleShowProductInNewWindow = (serial) => {
  try {
    const product = this[MODEL].getProduct(serial);
    this[VIEW].showProductInNewWindow(product);
  } catch (error) {
    this[VIEW].showProductInNewWindow(null, 'No existe este producto en
la página.');
  }
};
```

2.5. Enlazar manejador con la vista

En la *Vista* tenemos que definir el método *bind* para vincular los eventos del usuario con los manejadores del *Controlador*. Seleccionamos el botón de apertura de ventana, y en su evento *click* comprobamos si no está abierta aún para asociarla con la propiedad del constructor e invocar el manejador. No podemos añadir contenido en la nueva ventana hasta que su árbol DOM se haya cargado, por lo ejecutamos el manejado en respuesta al evento `DOMContentLoaded`.

Si la ventana ya está abierta, cambiamos el contenido y le pasamos el foco.

```
bindShowProductInNewWindow(handler) {
  const bOpen = document.getElementById('b-open');
  bOpen.addEventListener('click', (event) => {
    if (!this.productWindow || this.productWindow.closed) {
      this.productWindow = window.open('product.html', 'ProductWindow',
'width=800, height=600, top=250, left=250, titlebar=yes, toolbar=no,
menubar=no, location=no');
      this.productWindow.addEventListener('DOMContentLoaded', () => {
        handler(event.target.dataset.serial);
      });
    } else {
      handler(event.target.dataset.serial);
      this.productWindow.focus();
    }
  });
}
```

Por último, invocamos al método *bind* en el manejador `handleShowProduct()`, una vez generada la vista del producto seleccionado.

```
this[VIEW].bindShowProductInNewWindow(
  this.handleShowProductInNewWindow,
);
```