CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

ACTIVIDAD FINAL
SEGUNDA EVALUACIÓN

Desarrollo Web en entorno cliente
CFGS DAW

Álvaro Maceda Arranz

alvaro.maceda@ceedcv.es

2021/2022

Versión:220209.1132

Licencia

ACTIVIDAD FINAL - 2ª EVALUACIÓN

Revisa los criterios de evaluación para saber que es lo que debes tener en cuenta a la hora de realizar el ejercicio: no es suficiente que el programa cumpla la función, debe cumplir además otros criterios para ser considerado un buen código JavaScript.

Recuerda leer bien lo que pide la práctica: no se trata de que lo hagas como pienses que debe funcionar sino como se especifica en la misma.

1. ENUNCIADO

Debes programar una aplicación con tres pantallas. Debes utilizar el HTML y CSS proporcionado en el repositorio de la práctica:

https://github.com/CEED-2021/trabajo-final-segunda-evaluacion.git

Puedes realizar **modificaciones dinámicas** al HTML para conseguir lo solicitado en la práctica pero **no puedes realizar ninguna otra modificación al HTML, CSS o imágenes**.

1.1 Pantalla de login

Cuando arranque la aplicación por primera vez debes mostrar la pantalla de login:

Al pulsar el botón de “Sign in”:

- Si el formulario es válido, debes enviar una petición `POST` al API en formato `JSON` con los campos `username` y `password`:

```
POST /login

Petición:
{
  "username": "<uername>",
  "password": "<password>"
}
```

Los campos del formulario tienen validación HTML. Para comprobar si el formulario es válido puedes usar la función `checkValidity` del formulario.

Para marcar el formulario como validado y mostrar los errores debes añadirle la clase CSS `was-validated`

- Mientras se ejecuta la petición de login debes mostrar un loader:

Para mostrar el loader debes añadirle la clase `active`, y eliminarla para ocultarlo.

- Si la petición es incorrecta o se ha producido un error debes mostrar el mensaje bajo el formulario con el contenido del error devuelto por el servidor, o con el error de `fetch` si se ha producido un error.

Para mostrar el error deberás añadir la clase `show` al div contenedor del error

- Si la petición es correcta, el servidor devolverá un mensaje como este:

```
{
  "access_token": "eyJhbGZKQ01Y...oondIU",
  "token_type": "Bearer",
  "expires_in": 300,
  "player_id": 1
}
```

  El `access_token` deberá ser enviado en todas las peticiones subsiguientes en la cabecera, dentro del campo `authorization`, con el siguiente formato:

```
"authorization": "Bearer <TOKEN>"
```

- La autentificación tiene un límite de tiempo. Cuando el límite de tiempo haya pasado, el servidor devolverá un error 401 con una descripción como esta:

```
{
  "error": "Invalid authentication token",
  "description": "jwt expired"
}
```

  Cuando se produzca ese error (o cualquier otro error 401) debes volver a la pantalla de login para realizar una nueva autentificación.

- Al pulsar el botón “Sign in” debes mostrar:
  - En caso de que sea la primera vez en la sesión que se hace login, la lista de partidas
  - En caso de estar haciendo login de nuevo porque había fallado la autentificación al realizar alguna petición al servidor **debe volver a la pantalla donde estaba antes** (desde el inicio de esa pantalla, no hace falta que recrees el estado en el que estaba la pantalla antes de que fallara la petición)

1.2 Lista de partidas

La URL de esta ventana debe ser `/list.html?player=<id del jugador>`. La ventana mostrará los datos del `id` indicado en la URL.

Al abrir esta pantalla debes lanzar una petición para obtener el nombre del jugador:

```
GET player/:id

Respuesta:
{
  "id": 3,
  "username": "player3",
  "name": "Player 3"
}
```

A la vez que obtienes los datos del jugador debes obtener los datos de las partidas de ese jugador mediante otra petición:

```
GET /player/:id/games

Respuesta:
[ 1,3,4,... ]
```

Para cada una de las partidas obtenidas debes enviar una nueva petición para obtener la fecha y el resultado final de la misma.

```
GET /game/:id

Respuesta:
{
  "player": 1,
  "id": 1,
  "date": "2021-04-23T18:25:43.511Z",
  "result": [
    [1,0,2],
    [1,2,1],
    [2,2,1]
  ]
},
```

Un “1” dentro del resultado significa que ha jugado “x” en esa posición. Un “2” significa que ha jugado “o”. Un 0 significa que no se ha jugado en esa posición.

Debes cargar los datos necesarios de la forma más rápida posible. Mientras se resuelven las peticiones debes mostrar un loader, que desaparecerá una vez tengan todos los datos:

Una vez tengas **tanto el nombre del jugador como las partidas**, debes mostrar los datos en la ventana:

Si un jugador no tiene partidas, debes mostrar el texto “No games” en lugar de las partidas.

Al pulsar el botón “View Game” se navegará a la pantalla de movimientos de esa partida.

No hace falta que controles errores de carga en esta ventana: si se produce un error sólo debes mostrarlo en la consola del navegador.

1.3 Movimientos de la partida

La URL de esta ventana debe ser `/game.html?game=<id del juego>&movement=<codigo para obtener el primer movimiento>`. La ventana mostrará los movimientos de la partida con `id` indicado en la URL. En `movement` se le debe pasar el código para obtener el primer movimiento de la partida.

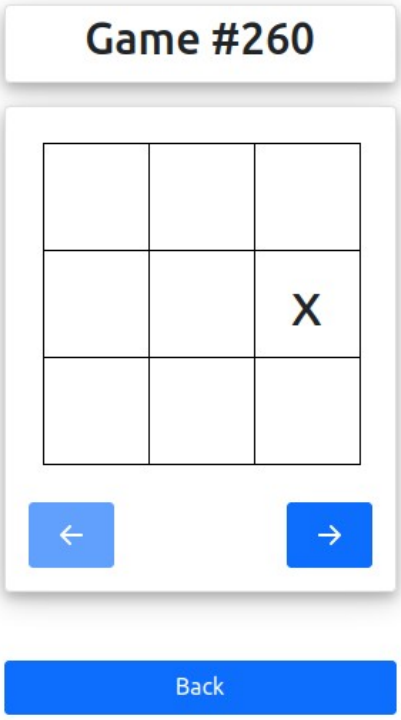
Para obtener un movimiento se utiliza la siguiente petición:

```
GET /game/3/movements/c4...

Respuesta:
{
  "movement": [0,0],
  "next": "0877533...aa"
}
```

`movement` indica la fila y la columna donde se ha jugado. El primer movimiento siempre es de “x”. Para obtener los siguientes movimientos debes lanzar la misma petición con el valor de `next` recibido.

Debes obtener todos los movimientos de la partida nada mas cargar la ventana. Debes mostrar un loader hasta que se haya cargado el primero movimiento y mostrarlo en el tablero nada más se haya cargado:



La flecha derecha se habilitará si hay más movimientos disponibles posteriores al actual (si el movimiento posterior no se ha cargado, se habilitará cuando esté cargado) La flecha izquierda se habilitará si no estamos en el primer movimiento:

Al pulsar la flecha derecha se mostrará el juego tras el siguiente movimiento. Al pulsar la flecha izquierda, se mostrará el juego en el momento del movimiento anterior.

Al pulsar “Back” se volverá a la lista de juegos del jugador.

No hace falta que controles errores de carga en esta ventana: si se produce un error sólo debes mostrarlo en la consola del navegador.

1.4 API

Puedes probar el servidor en esta dirección: https://dwec-tres-en-raya.herokuapp.com

También tienes el código del servidor en esta dirección:

https://github.com/CEED-2021/tres-en-raya-server

Puedes clonar el repositorio y lanzarlo con `yarn start` después de haber instalado las dependencias.

En el servidor hay tres usuarios para pruebas: `player1/pass1`, `player2/pass2` y `player3` (debes adivinar la contraseña de este último)

El código del servidor es limitado y solamente devuelve un conjunto limitado de datos. Sin embargo, debes programar la aplicación como si pudieras recibir un número indeterminado de partidas y movimientos. Para esta práctica no hace falta que implementes ningún tipo de paginación.

2. Entorno de desarrollo

Para que la aplicación se vea correctamente deberás instalar las dependencias de Bootstrap en el proyecto:

```
"dependencies": {
  "@popperjs/core": "^2.11.2",
  "bootstrap": "^5.1.3"
}
```

Además, deberás importar el código y el CSS:

```
import 'bootstrap'
import 'bootstrap/dist/css/bootstrap.min.css'
```

Todo el código HTML, CSS y js, así como las imágenes, estarán en el directorio `src`.

Todas las pruebas estarán en el directorio `tests`

Además de programar el código, deberás montar un entorno de desarrollo con Webpack. Debe admitir cuatro scripts:

- `yarn test` ejecutará las pruebas de la aplicación con Jest
- `yarn start` lanzara `webpack-dev-serve` con autoreloading
- `yarn build` generará los ficheros distribuibles de la aplicación en el directorio `dist`

- `yarn eslint` ejecutará el linter sobre los directorios `src` y `test`

Se debe transformar el código usando el preset `preset-env` de Babel.

Se debe poder lanzar la aplicación en desarrollo con `yarn start`

Se debe poder lanzar la aplicación y funcionar correctamente con el código generado en `dist`. No incluyas ese código en tu entrega, debe poder generarse automáticamente con `yarn build`

El código generado debe tener tres páginas distintas en el mismo directorio: `index.html` (para el login), `list.html` (para la lista de partidas) y `game.html` (para los movimientos del juego)

El código final del ejercicio deberá lanzar las peticiones al servidor de Internet (https://dwec-tres-en-raya.herokuapp.com)

3. Pruebas

Debes completar las pruebas que hay en el fichero `api.test.js`. Puedes añadir más tests pero no serán evaluados.

4. Linter

Para que el ejercicio se corrija el linter no debe devolver ningún error. En caso de que el linter devuelva un error, la máxima nota del trabajo será de 1.

Las reglas del linter son las que están especificadas en `.eslintrc.json`. No puede modificarse este fichero. Asimismo no se admitirá deshabilitar ninguna de las reglas de eslint por ningún medio: en ese caso se procederá como si el programa hubiese fallado el linter.

5. Entrega

Para la entrega debes eliminar los directorios `node_modules` y `dist` y comprimir el directorio de tu proyecto en un único fichero `.zip` o `.gz`.

Debes entregar el ejercicio como un único fichero con el siguiente nombre: `NOMBRE_APELLIDO1.[zip|gz]` (sustituye *NOMBRE* y *APELLIDO1* por tu nombre y tu primer apellido) El fichero se entregará en la tarea del curso habilitada a tal efecto.

No se admitirán entregas pasada la fecha límite.

6. Criterios de evaluación

- El programa es correcto, realiza la función que se solicita en el enunciado
- Se han utilizado estructuras del lenguaje adecuadas: bucles, condicionales, operadores, etc.
- Se ha estructurado correctamente el código utilizando módulos
- Se han utilizado variables y constantes de forma adecuada

- Se utilizan correctamente y cuando corresponda los tipos de datos y objetos predefinidos del lenguaje (Arrays, objetos planos, Map, Set, etc.)
- Se han utilizado funciones para estructurar el código, definiendo y utilizando parámetros y valores de respuesta de forma adecuada
- El programa es lo más sencillo posible para realizar su función.
- No existe código repetido: se han extraído los comportamientos comunes a funciones y se ha intentado hacer el código genérico.
- Se han creado las pruebas requeridas para validar la funcionalidad del código utilizando las validaciones adecuadas.
- Se han estructurado las pruebas de forma correcta.