Desarrollo web en entorno cliente.
Práctica 1.

Diseña una web para desplazar cartas de una baraja de un div a un conjunto de div (uno por cada palo).

1. Diseña una estructura adecuada usando módulos y clases separando todo lo posible tu código de la capa de presentación.
2. Diseña tu programa para que una carta sólo pueda ser soltada en un div marcado a través de un dataset.
3. Usa fetch sobre una API-Rest para guardar la estructura actual, de manera que cada vez que un cliente se conecte recupere el estado actual.

Evaluación:

1. Diseño y Estructura del Código

- **Excelente (4)**: El código está perfectamente estructurado, utilizando módulos y clases bien definidas. Cada parte del código está separada de manera adecuada, siguiendo principios sólidos de separación de responsabilidades (como la separación entre lógica y presentación). El código es fácil de entender y seguir.
- **Bueno (3)**: El código está mayormente estructurado en módulos y clases, pero algunas partes están ligeramente mezcladas o no son completamente claras. Aún sigue principios de separación de responsabilidades, pero con áreas que podrían mejorarse.
- **Aceptable (2)**: El código tiene una estructura básica, pero no sigue un enfoque modular claro. Algunas áreas del código están combinadas o poco organizadas, lo que dificulta la comprensión del flujo.
- **Insuficiente (1)**: El código no tiene una estructura modular clara y se encuentra desorganizado. La separación entre la lógica y la presentación es confusa y poco mantenible.

2. Interactividad y Funcionalidad de las Cartas

- **Excelente (4)**: Las cartas se pueden mover y soltar correctamente en los divs de destino, y solo se pueden soltar en divs específicos mediante el uso de `dataset`. La funcionalidad está completamente implementada y funciona sin errores.
- **Bueno (3)**: Las cartas se pueden mover y soltar en los divs de destino, pero hay pequeños detalles que no funcionan de manera perfecta (por ejemplo, podrían haber problemas visuales al soltar o algún pequeño error con el `dataset`).
- **Aceptable (2)**: Las cartas se pueden mover, pero no se pueden soltar correctamente o no se utiliza `dataset`adecuadamente para restringir los destinos. La funcionalidad no está completamente implementada.
- **Insuficiente (1)**: Las cartas no se pueden mover o soltar, o el uso de `dataset` no está implementado, lo que impide que el comportamiento de la aplicación sea el esperado.

3. Uso de Fetch para Recuperar/Guardar Estado

- **Excelente (4)**: Se utiliza `fetch` de manera correcta para interactuar con la API REST. El estado de la baraja se guarda correctamente y se recupera al cargar la página. La funcionalidad de persistencia está completamente implementada y funciona entre sesiones.
- **Bueno (3)**: Se utiliza `fetch` para interactuar con la API REST, pero algunos detalles de la persistencia del estado o la recuperación de los datos no siempre funcionan de manera consistente.
- **Aceptable (2)**: Se utiliza `fetch`, pero la integración con la API REST tiene fallos importantes, como errores al recuperar el estado o problemas con la persistencia de datos.
- **Insuficiente (1)**: No se utiliza `fetch` para interactuar con la API REST o la persistencia de datos no funciona correctamente.

4. Separación de Lógica y Presentación

- **Excelente (4)**: El código JavaScript está completamente separado de la capa de presentación (HTML y CSS). Las manipulaciones del DOM se realizan de manera eficiente sin mezclar lógica de negocio con la visualización de la interfaz.
- **Bueno (3)**: El código JavaScript está mayormente separado, pero puede haber algunos puntos donde la lógica y la presentación estén ligeramente mezclados (por ejemplo, algunos cambios de estilo directamente en el código JS).
- **Aceptable (2)**: El código JavaScript y la presentación están parcialmente separados, pero hay áreas donde la lógica afecta directamente a la presentación, lo que hace que el código sea menos modular.
- **Insuficiente (1)**: El código JavaScript y la presentación están completamente entrelazados, lo que hace que el código sea difícil de entender y mantener.

5. Usabilidad e Interfaz de Usuario (UI)

- **Excelente (4)**: La interfaz es clara, intuitiva y fácil de usar. El diseño es atractivo y todos los elementos interactivos tienen un comportamiento predecible y comprensible. El usuario puede interactuar con las cartas de manera fluida.
- **Bueno (3)**: La interfaz es funcional y clara, pero puede haber algunos detalles menores que no son del todo intuitivos o algunos problemas menores de diseño. La interacción es mayormente fácil de entender.
- **Aceptable (2)**: La interfaz tiene algunos problemas de usabilidad y el diseño no es completamente claro. El usuario puede experimentar dificultades para interactuar con los elementos o entender cómo se debe usar la página.
- **Insuficiente (1)**: La interfaz es difícil de usar, poco clara o el diseño es confuso. La interacción con las cartas no es intuitiva y la navegación es difícil.

6. Validación y Manejo de Errores

- **Excelente (4)**: El programa maneja todos los posibles errores de manera efectiva, como la falta de respuesta de la API, errores en la carga de datos, o problemas con

el movimiento de cartas. El usuario es informado adecuadamente cuando ocurre un error.
- **Bueno (3)**: El programa maneja la mayoría de los errores, pero algunos fallos menores no son gestionados correctamente, o el usuario no es informado de manera adecuada.
- **Aceptable (2)**: El programa maneja algunos errores, pero no de manera consistente. Podrían producirse fallos sin que se les dé una solución adecuada al usuario.
- **Insuficiente (1)**: El programa no maneja adecuadamente los errores, lo que puede generar fallos o comportamientos inesperados sin ninguna forma de retroalimentación al usuario.

7. Comentarios y Documentación

- **Excelente (4)**: El código está completamente documentado, con comentarios claros y útiles que explican la lógica y las decisiones de desarrollo. La estructura del proyecto está bien documentada, facilitando su comprensión.
- **Bueno (3)**: El código está mayormente documentado, pero algunos detalles complejos pueden no estar suficientemente explicados. La estructura del proyecto está documentada en su mayoría.
- **Aceptable (2)**: El código tiene pocos comentarios, lo que dificulta la comprensión de su funcionamiento. No está claro por qué se toman algunas decisiones de desarrollo.
- **Insuficiente (1)**: El código no tiene comentarios ni documentación, lo que hace que sea difícil de entender y mantener.

Puntuación Total

- **Excelente (26-28 puntos)**: El proyecto está bien estructurado, funciona perfectamente y sigue buenas prácticas en cuanto a separación de responsabilidades y manejo del estado.
- **Bueno (20-25 puntos)**: El proyecto está bien desarrollado, pero hay algunos detalles menores a mejorar. Puede haber algunos errores, pero la funcionalidad principal está implementada correctamente.
- **Aceptable (15-19 puntos)**: El proyecto cumple con la funcionalidad básica, pero hay varias áreas que necesitan mejorar, como la organización del código, la usabilidad o la integración con la API.
- **Insuficiente (menos de 15 puntos)**: El proyecto no cumple con varios aspectos importantes, ya sea por fallos en la funcionalidad, la estructura del código o la implementación de la API.

Versión 2. Modifica tu proyecto y usa en tu proyecto websockets. (+1)