# DAW Cliente – MVC

El patrón Modelo-Vista-Controlador (MVC) es un enfoque de diseño de software que divide una aplicación en tres componentes principales: el Modelo, la Vista y el Controlador. Este patrón se puede aplicar tanto en el lado del cliente como en el lado del servidor. Aquí te proporcionaré una breve descripción de cómo puedes implementar el patrón MVC en el lado del cliente, principalmente utilizando JavaScript para la lógica del cliente y HTML/CSS para la presentación.

1. **Modelo (Model):** El modelo representa los datos y la lógica de la aplicación. Puede incluir funciones para acceder y modificar los datos.
2. **Vista (View):** La vista es la interfaz de usuario (UI). Representa la presentación de los datos y responde a las interacciones del usuario.
3. **Controlador (Controller):** El controlador maneja las interacciones del usuario y actualiza tanto el modelo como la vista en consecuencia.

Ejemplo sencillo de una aplicación web básica usando MVC:

### index.html

```
<!-- index.html (Vista) -->
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Mi Aplicación MVC</title>
  <style>
    /* Estilos CSS para la presentación */
    body {
      font-family: Arial, sans-serif;
      text-align: center;
    }
  </style>
</head>
<body>
  <h1>Mi Aplicación MVC</h1>
  <button id="b1">Cargar datos</button>
  <p id="output">Datos del Modelo: <span id="data"></span></p>
  <script type="module" src="controller.js"></script>
</body>
</html>
```

### model.js

```
// model.js (Modelo)
const model = {
    data: null,

    fetchData: async function() {
      return new Promise(resolve => {
        setTimeout(() => {
          this.data = 'Datos cargados desde el modelo';
          resolve();
        }, 1000);
      });
    }
  };

  // Exportar el modelo para que pueda ser importado por otros archivos
  export default model;
```

### view.js

```
// view.js (Vista)
const view = {
    render: function(data) {
      document.getElementById('data').textContent = data;
    }
  };

// Exportar la vista para que pueda ser importada por otros archivos
export default view;
```

### controller.js

```
// controller.js (Controlador)
import model from './model.js'; // Importar el modelo
import view from './view.js'; // Importar la vista

const controller = {
    loadData: async function() {
        await model.fetchData();
        view.render(model.data);
    }
};

document.getElementById('b1').addEventListener('click', controller.loadData)

// Exportar el controlador para que pueda ser importado por otros archivos
export default controller;

// Iniciar la aplicación cargando el controlador
```

```
//controller.loadData();
```

Este es un ejemplo básico que simula la carga de datos desde el modelo (usando un temporizador en lugar de una solicitud a una API real). Cada componente (Modelo, Vista y Controlador) tiene su propio archivo, pero en aplicaciones más grandes, puedes estructurarlos de manera diferente, como en carpetas separadas.

Al hacer clic en el botón "Cargar Datos", el controlador solicita al modelo cargar los datos, y luego actualiza la vista con los datos obtenidos del modelo. Este enfoque facilita la escalabilidad y el mantenimiento de tu código al separar las responsabilidades y permitir que cada componente realice su función específica.

En este ejemplo:

- En `model.js`, he utilizado `export default` para exportar el objeto `model`.
- En `view.js`, he utilizado `export default` para exportar el objeto `view`.
- En `controller.js`, he utilizado `import` para importar `model` y `view`.

Estas declaraciones permiten que los archivos se comuniquen entre sí y utilicen las funciones y objetos definidos en otros archivos. Esto sigue las convenciones de importación/exportación de módulos en JavaScript y es compatible tanto con navegadores modernos como con entornos Node.js.