Etapa 6

Rumbo al Proyecto final Full Stack Python con objetos y API REST

Sexta etapa:

Implementar un frontend

En esta etapa veremos una serie de códigos mínimos que nos ayuden a desarrollar un frontend que utilize las funciones de la API desarrollada.
Haremos incapié en los temas relacionados con la API en sí, y no tanto en el apartado CSS o HTML.
Usaremos Vue3 para simplificar el código.

Menú principal

El archivo de entrada al frontend es simplemente un menú con enlaces que llevan a cada una de las secciónes implementadas:

index.html:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo de uso de la API</title>
    <link rel="stylesheet" href="estilos.css">
</head>

<body>
    <div>
        <h1>Ejemplos de uso de la API</h1>
        <h3>Codo a Codo 2023</h3>
        <table>
            <tr>
                <td class="contenedor-centrado"><a href="altas.html">Alta de productos</a></td>
            </tr>

            <tr>
                <td class="contenedor-centrado "><a href="listado.html">Listado de productos</a></td>
            </tr>
```

```
            <tr>
                <td class="contenedor-centrado "><a href="modificaciones.html">Modificar datos de productos</a></td>
            </tr>

            <tr>
                <td class="contenedor-centrado "><a href="listadoEliminar.html">Eliminar productos</a></td>
            </tr>

            <tr>
                <td class="contenedor-centrado "><a href="carrito1.html">Carrito de compras</a></td>
            </tr>
        </table>
</body>

</html>
```

Alta de productos

altas.html:

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Agregar producto</title>
    <link rel="stylesheet" href="estilos.css">
</head>

<body>
    <h1>Agregar Productos al Inventario</h1>
    <h3>Codo a Codo 2023</h3>
    <form id="formulario">
        <label for="codigo">Código:</label>
        <input type="text" id="codigo" name="codigo" required><br>

        <label for="descripcion">Descripción:</label>
        <input type="text" id="descripcion" name="descripcion" required><br>

        <label for="cantidad">Cantidad:</label>
        <input type="number" id="cantidad" name="cantidad" required><br>
```

```
        <label for="precio">Precio:</label>
        <input type="number" step="0.01" id="precio" name="precio" required><br>

        <button type="submit">Agregar Producto</button>
        <a href="index.html">Menu principal</a>
    </form>

    <script>
        //const URL = "http://127.0.0.1:5000/"
        const URL = "https://xxxxx.pythonanywhere.com/"
        // Capturamos el evento de envío del formulario
        document.getElementById('formulario').addEventListener('submit', function (event) {
            event.preventDefault(); // Evitamos que se recargue la página

            // Obtenemos los valores del formulario
            var codigo = document.getElementById('codigo').value;
            var descripcion = document.getElementById('descripcion').value;
            var cantidad = document.getElementById('cantidad').value;
            var precio = document.getElementById('precio').value;

            // Creamos un objeto con los datos del producto
            var producto = {
                codigo: codigo,
                descripcion: descripcion,
                cantidad: cantidad,
                precio: precio
            };
            console.log(producto)
            // Realizamos la solicitud POST al servidor
            fetch(URL + 'productos', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(producto)
            })
                .then(function (response) {
                    // Código para manejar la respuesta
                    if (response.ok) {
                        return response.json(); // Parseamos la respuesta JSON
                    } else {
                        // Si hubo un error, lanzar explícitamente una excepción
                        // para ser "catcheada" más adelante
                        throw new Error('Error al agregar el producto.');
                    }
                })
                .then(function (data) {
                    alert('Producto agregado correctamente.');
                    //Limpiamos el formulario.
                    document.getElementById('codigo').value = "";
                    document.getElementById('descripcion').value = "";
                    document.getElementById('cantidad').value = "";
                    document.getElementById('precio').value = "";
                })
                .catch(function (error) {
                    // Código para manejar errores
                    alert('Error al agregar el producto.');
                });
        })
    </script>
</body>

</html>
```

Descripción del script utilizado:

Se comienza declarando una constante:

```
const URL = "https://xxxxx.pythonanywhere.com/";
```

Esta constante almacena la URL del servidor al que se enviarán las solicitudes POST. Puede ser una dirección local (como "http://127.0.0.1:5000/") o una dirección remota (como en este caso). Se debe cambiar la URL dependiendo de las necesidades del proyecto. A continuación, se utiliza el método **addEventListener** para agregar un evento de escucha al formulario con el identificador "formulario" cuando se envíe:

```
document.getElementById('formulario').addEventListener('submit', function (event) {
        event.preventDefault(); // Evitamos que se envie el form por ahora
        // Código del evento
});
```

Esto permite capturar el evento de envío del formulario y ejecutar el código correspondiente cuando se envía el formulario. **event.preventDefault();** se utiliza para evitar que la página se recargue cuando se envía el formulario. Esto se logra llamando al método **preventDefault()** en el objeto event pasado como parámetro en la función del evento. A continuación, se obtienen los valores de los campos de entrada de texto y se almacenan en variables:

```
// Obtenemos los valores del formulario
var codigo = document.getElementById('codigo').value;
var descripcion = document.getElementById('descripcion').value;
var cantidad = document.getElementById('cantidad').value;
var precio = document.getElementById('precio').value;
```

Esto se logra utilizando el método **getElementById()** para obtener los elementos del DOM correspondientes a los campos de entrada y accediendo a su propiedad value para obtener el valor ingresado por el usuario.Luego, se crea un objeto producto con los datos obtenidos del formulario:

```
// Creamos un objeto con los datos del producto
var producto = {
        codigo: codigo,
        descripcion: descripcion,
        cantidad: cantidad,
        precio: precio
};
```

Este objeto producto contiene las propiedades codigo, descripcion, cantidad y precio, con sus respectivos valores obtenidos del formulario. Se realiza una solicitud POST al servidor utilizando el método fetch():

```
fetch(URL + 'productos', {
        method: 'POST',
        headers: {
                'Content-Type': 'application/json'
        },
        body: JSON.stringify(producto)
})
        .then(function (response) {
                // Código para manejar la respuesta
        })
        .catch(function (error) {
                // Código para manejar errores
        });
```

El URL utilizado en la solicitud POST se construye concatenando la constante URL con la ruta específica, en este caso, 'productos'. Se especifica que el método de la solicitud es POST, se establece el encabezado 'Content-Type': 'application/json' para indicar que se está enviando datos en formato JSON, y se convierte el objeto producto a JSON utilizando JSON.stringify() antes de enviarlo como cuerpo de la solicitud.

Después de realizar la solicitud POST, se utiliza el método then() para manejar la respuesta del servidor:

```
.then(function (response) {
        if (response.ok) {
                return response.json(); // Parseamos la respuesta JSON
        } else {
                throw new Error('Error al agregar el producto.');
        }
})
.then(function (data) {
        // Código para manejar los datos de la respuesta
})
.catch(function (error) {
        // Código para manejar errores
});
```

En este caso, se verifica si la respuesta del servidor es exitosa (response.ok). Si es así, se llama al método json() en la respuesta para obtener los datos de la respuesta en formato JSON. Si la respuesta no es exitosa, se lanza un error. Finalmente, se utiliza otro then() para manejar los datos de la respuesta:

```
.then(function (data) {
    alert('Producto agregado correctamente.');
    //Limpiamos el formulario.
    document.getElementById('codigo').value = "";
    document.getElementById('descripcion').value = "";
    document.getElementById('cantidad').value = "";
    document.getElementById('precio').value = "";
})
.catch(function (error) {
        // Código para manejar errores
```

```
        alert('Error al agregar el producto.');
});
```

En este caso, se muestra una alerta indicando que el producto se agregó correctamente y se restablecen los valores de los campos del formulario a través de la manipulación del DOM. En caso de producirse algún error durante el proceso, se captura en el bloque catch() y se muestra una alerta de error. Este script maneja el evento de envío del formulario, obtiene los valores ingresados por el usuario, realiza una solicitud POST al servidor con los datos del producto y maneja la respuesta del servidor.

Listado de productos en el inventario (sin Vue3)

El siguiente documento HTML utiliza la API implementada para generar una tabla con el contenido de la tabla que contiene el inventario completo.
Se utilizan técnicas de manipulación del DOM "puras", sin Vue, para que el alumno pueda ver en acción lo aprendido antes:

listado.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Listado de Productos</title>
    <link rel="stylesheet" href="estilos.css">
</head>

<body>
    <h1>Listado de Productos</h1>
    <h3>Codo a Codo 2023</h3>
    <table>
        <thead>
            <tr>
                <th>Código</th>
                <th>Descripción</th>
                <th>Cantidad</th>
                <th align="right">Precio</th>
            </tr>
        </thead>
        <tbody id="tablaProductos">
        </tbody>
    </table>

    <div class="contenedor-centrado">
        <a href="index.html">Menu principal</a>
    </div>

    <script>
        //const URL = "http://127.0.0.1:5000/"
        const URL = "https://xxxxx.pythonanywhere.com/"

        // Realizamos la solicitud GET al servidor para obtener todos los productos
        fetch(URL + 'productos')
            .then(function (response) {
                if (response.ok) {
                    return response.json(); // Parseamos la respuesta JSON
                } else {
```

```
                    // Si hubo un error, lanzar explícitamente una excepción
                    // para ser "catcheada" más adelante
                    throw new Error('Error al obtener los productos.');
                }
            })
            .then(function (data) {
                var tablaProductos = document.getElementById('tablaProductos');

                // Iteramos sobre los productos y agregamos filas a la tabla
                data.forEach(function (producto) {
                    var fila = document.createElement('tr');
                    fila.innerHTML = '<td>' + producto.codigo + '</td>' +
                        '<td>' + producto.descripcion + '</td>' +
                        '<td align="right">' + producto.cantidad + '</td>' +
                        '<td align="right">      ' + producto.precio + '</td>';
                    tablaProductos.appendChild(fila);
                });
            })
            .catch(function (error) {
                // Código para manejar errores
                alert('Error al obtener los productos.');
            });
    </script>
</body>
</html>
```

El código JavaScript se encuentra dentro de la etiqueta <script> y se ejecuta cuando la página se carga en el navegador. Su objetivo principal es obtener los datos de los productos desde la API y mostrarlos en la tabla dentro del documento HTML.

Obtención de datos de la API:

- Mediante la función fetch, se realiza una solicitud GET al servidor para obtener todos los productos.
- La URL base de la API se define como una constante llamada URL.
- La función fetch devuelve una promesa que representa la respuesta del servidor.
- Se encadenan las funciones then para manejar la respuesta recibida.

Parseo de la respuesta JSON:

- En el primer then, se verifica si la respuesta del servidor es exitosa mediante la propiedad ok de la respuesta.
- Si la respuesta es exitosa, se llama a la función response.json() para parsear la respuesta como JSON.
- La función response.json() también devuelve una promesa que representa los datos parseados.

Manipulación del DOM:

- Una vez que se obtienen los datos parseados en el segundo then, se accede al elemento HTML con el id "tablaProductos" utilizando document.getElementById('tablaProductos').
- La tabla se representa mediante el elemento <tbody> dentro del documento HTML.
- Se utiliza un bucle forEach para iterar sobre los productos obtenidos en data.

Creación de filas de la tabla:

- Para cada producto, se crea una nueva fila de tabla utilizando document.createElement('tr').
- Se accede a las propiedades del producto, como codigo, descripcion, cantidad y precio.
- Se utiliza la propiedad innerHTML para asignar el contenido HTML de la fila.
- El contenido HTML se crea concatenando las propiedades del producto en las celdas correspondientes de la fila.

Agregado de filas a la tabla:

- Una vez que se crea la fila con el contenido del producto, se agrega a la tabla utilizando el método appendChild del elemento tablaProductos.
- Esto agrega la fila como un hijo del elemento <tbody>, lo que hace que aparezca visualmente en la tabla.

Manejo de errores:

- Si se produce algún error durante la obtención de los productos (por ejemplo, si la respuesta del servidor no es exitosa), se ejecuta el bloque catch.
- Dentro del bloque catch, se muestra una alerta con el mensaje "Error al obtener los productos".

En resumen, el código JavaScript se encarga de obtener los productos desde la API, parsear los datos recibidos, y luego crear y agregar filas a la tabla en el documento HTML para mostrar los productos al usuario. La manipulación del DOM se realiza utilizando métodos y propiedades proporcionados por el navegador, como createElement, innerHTML y appendChild.

Eliminar productos del inventario (con Vue3)

El código siguiente muestra como generar un listado similar al anterior, pero empleando Vue3. Y se agrega un botón "Eliminar" en cada fila de la tabla, que utiliza la API desarrollada para quitar del inventario el producto seleccionado.

listadoEliminat.html:

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Listado de Productos</title>
    <link rel="stylesheet" href="estilos.css">
</head>

<body>
    <h1>Baja de Productos</h1>
    <h3>Codo a Codo 2023</h3>
    <table>
        <thead>
            <tr>
                <th>Código</th>
                <th>Descripción</th>
                <th>Cantidad</th>
                <th align="right">Precio</th>
                <th>Acciones</th>
            </tr>
        </thead>
        <tbody>
            <tr v-for="producto in productos">
                <td>{{ producto.codigo }}</td>
                <td>{{ producto.descripcion }}</td>
                <td>{{ producto.cantidad }}</td>
                <td align="right">{{ producto.precio }}</td>
                <td><button @click="eliminarProducto(producto.codigo)">Eliminar</button></td>
            </tr>
        </tbody>
    </table>

    <div class="contenedor-centrado">
        <a href="index.html">Menu principal</a>
    </div>

    <script src="https://unpkg.com/vue@next"></script>
    <script>
        //const URL = "http://127.0.0.1:5000/"
        const URL = "https://arielcodo.pythonanywhere.com/";

        const app = Vue.createApp({
            data() {
                return {
                    productos: []
                }
            },
            methods: {
                obtenerProductos() {
                    // Obtenemos el contenido del inventario
```

```
                    fetch(URL + 'productos')
                        .then(response => {
                            if (response.ok) {
                                return response.json(); // Parseamos la respuesta JSON
                            } else {
                                // Si hubo un error, lanzar explícitamente una excepción
                                // para ser "catcheada" más adelante
                                throw new Error('Error al obtener los productos.');
                            }
                        })
                        .then(data => {
                            // El código Vue itera este elemento para generar la tabla
                            this.productos = data;
                        })
                        .catch(error => {
                            console.log('Error:', error);
                            alert('Error al obtener los productos.');
                        });
                },
                eliminarProducto(codigo) {
                    // Eliminamos el producto de la fila seleccionada
                    fetch(URL + `productos/${codigo}`, { method: 'DELETE' })
                        .then(response => {
                            if (response.ok) {
                                // Eliminar el producto de la lista después de eliminarlo en el servidor
                                this.productos = this.productos.filter(producto => producto.codigo !== codigo);
                                console.log('Producto eliminado correctamente.');
                            } else {
                                // Si hubo un error, lanzar explícitamente una excepción
                                // para ser "catcheada" más adelante
                                throw new Error('Error al eliminar el producto.');
                            }
                        })
                        .catch(error => {
                            // Código para manejar errores
                            alert('Error al eliminar el producto.');
                        });
                }
            },
            mounted() {
                //Al cargar la página, obtenemos la lista de productos
                this.obtenerProductos();
            }
        });

        app.mount('body');
    </script>
</body>
</html>
```

Este código utiliza Vue.js 3 para realizar solicitudes a la API y manipular la lista de productos en la interfaz. Al cargar la página, se obtiene la lista de productos desde la API y se muestra en la tabla mediante la directiva v-for. Al hacer clic en el botón "Eliminar" de una fila, se elimina el producto tanto en la API como en la lista local, actualizando la interfaz en tiempo real. Funciona de la siguiente manera:

1) Se incluye la biblioteca Vue.js 3 mediante la etiqueta <script src="" title="https://unpkg.com/vue@next>" class="https">https://unpkg.com/vue@next"></script>.

2) Se crea una instancia de Vue mediante const app = Vue.createApp({...}). Aquí se define la lógica de Vue para la página.

3) En la sección data(), se declara un objeto productos que se utilizará para almacenar los datos de los productos obtenidos de la API.

4) En la sección methods, se definen dos métodos:

- **obtenerProductos()** realiza una solicitud GET a la API para obtener la lista de productos. Si la respuesta es exitosa, se asigna la respuesta JSON a this.productos, que actualiza los datos en la interfaz.
- **eliminarProducto(codigo)** se activa cuando se hace clic en el botón "Eliminar" de una fila de producto. Realiza una solicitud DELETE a la API para eliminar el producto con el código especificado. Si la respuesta es exitosa, se actualiza la lista local productos filtrando el producto eliminado y se reflejan los cambios en la interfaz.

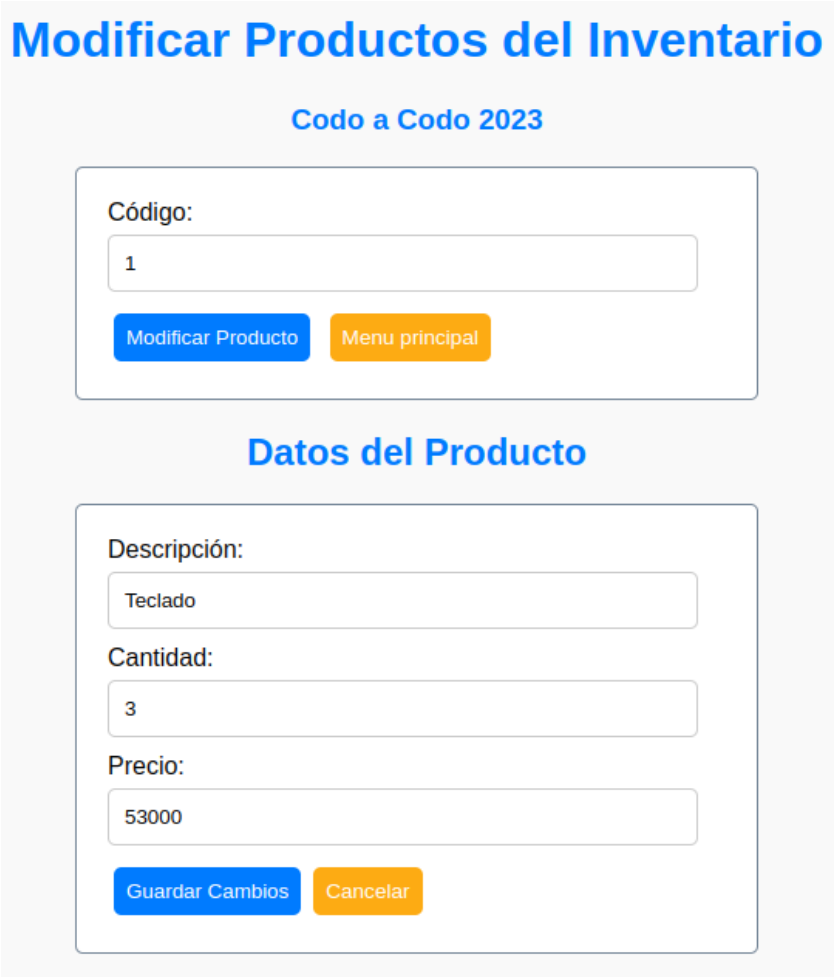
5) En la sección mounted(), se llama al método obtenerProductos() cuando la página se carga. Esto asegura que se obtenga la lista de productos y se muestren en la tabla.

6) En el HTML, se utiliza la directiva v-for="producto in productos" para iterar sobre la lista de productos y generar una fila en la tabla para cada producto. La directiva :key="producto.codigo" se utiliza para proporcionar una clave única a cada fila y mejorar el rendimiento de Vue al actualizar la lista.

7) En cada fila de la tabla, se utilizan las interpolaciones { { producto.codigo } }, { { producto.descripcion } } , { { producto.cantidad } } y { { producto.precio } } para mostrar los datos de cada producto.

8) El botón "Eliminar" tiene un manejador de eventos @click="eliminarProducto(producto.codigo)" que llama al método eliminarProducto pasando el código del producto como parámetro.

Modificar productos en el inventario (con Vue3)



El ejemplo siguiente nos muestra una manera de utilizar la API para modificar los datos de un producto del inventario. Se utiliza Vue3.

modificaciones.html:

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Modificar Producto</title>
    <link rel="stylesheet" href="estilos.css">
</head>

<body>
    <h1>Modificar Productos del Inventario</h1>
    <h3>Codo a Codo 2023</h3>
    <div id="app">
        <form @submit.prevent="obtenerProducto">
            <label for="codigo">Código:</label>
            <input type="text" v-model="codigo" required><br>
            <button type="submit">Modificar Producto</button> <a href="index.html">Menu principal</a>
        </form>

        <div v-if="mostrarDatosProducto">
            <h2>Datos del Producto</h2>
            <form @submit.prevent="guardarCambios">
                <label for="descripcionModificar">Descripción:</label>
```

```
                <input type="text" id="descripcionModificar" v-model="descripcion" required><br>

                <label for="cantidadModificar">Cantidad:</label>
                <input type="number" id="cantidadModificar" v-model="cantidad" required><br>

                <label for="precioModificar">Precio:</label>
                <input type="number" step="0.01" id="precioModificar" v-model="precio" required><br>

                <button type="submit">Guardar Cambios</button>
                <a href="modificaciones.html">Cancelar</a>
            </form>
        </div>
    </div>

    <script src="https://unpkg.com/vue@next"></script>
    <script>
        //const URL = "http://127.0.0.1:5000/"
        const URL = "https://xxxxx.pythonanywhere.com/";

        const app = Vue.createApp({
            data() {
                return {
                    codigo: '',
                    mostrarDatosProducto: false,
                    descripcion: '',
                    cantidad: '',
                    precio: ''
                }
            },
            methods: {
                obtenerProducto() {
                    fetch(URL + 'productos/' + this.codigo)
                        .then(response => {
                            if (response.ok) {
                                return response.json();
                            } else {
                                throw new Error('Error al obtener los datos del producto.');
                            }
                        })
                        .then(data => {
                            this.descripcion = data.descripcion;
                            this.cantidad = data.cantidad;
                            this.precio = data.precio;
                            this.mostrarDatosProducto = true;
                        })
                        .catch(error => {
                            alert('Error al obtener los datos del producto.');
                        });
                },
                guardarCambios() {
                    const producto = {
                        codigo: this.codigo,
                        descripcion: this.descripcion,
                        cantidad: this.cantidad,
                        precio: this.precio
                    };

                    fetch(URL + 'productos/' + this.codigo, {
                        method: 'PUT',
                        headers: {
                            'Content-Type': 'application/json'
                        },
                        body: JSON.stringify(producto)
                    })
                        .then(response => {
                            if (response.ok) {
                                return response.json();
                            } else {
                                throw new Error('Error al guardar los cambios del producto.');
                            }
                        })
                        .then(data => {
                            alert('Cambios guardados correctamente.');
                            location.reload();
                        })
                        .catch(error => {
                            alert('Error al guardar los cambios del producto.');
                        });
                }
            }
        });

        app.mount('#app');
    </script>
</body>

</html>
```

En el HTML, hemos incluido el CDN de Vue 3 y hemos creado un contenedor con el id "app" donde se montará nuestra aplicación Vue. Dentro del contenedor, tenemos un formulario de consulta y un formulario de modificación, que inicialmente se muestra oculto.

En el script, comenzamos por definir la constante URL, que representa la URL de la API utilizada para obtener y modificar los datos del producto.

Luego, utilizamos Vue.createApp para crear nuestra aplicación Vue. Dentro de data(), definimos las propiedades que serán utilizadas en nuestra aplicación: codigo para almacenar el código del producto, mostrarDatosProducto para controlar la visibilidad del formulario de modificación, y descripcion, cantidad y precio para almacenar los datos del producto seleccionado.

En la sección de methods, definimos dos métodos: **obtenerProducto** y **guardarCambios**. El método obtenerProducto se ejecuta cuando se envía el formulario de consulta. En este método, utilizamos fetch para realizar una solicitud GET a la API y obtener los datos del producto correspondiente al código ingresado.

obtenerProducto():

```
obtenerProducto() {
    fetch(URL + 'productos/' + this.codigo)
        .then(response => {
            if (response.ok) {
                return response.json();
            } else {
                throw new Error('Error al obtener los datos del producto.');
            }
        })
        .then(data => {
            this.descripcion = data.descripcion;
            this.cantidad = data.cantidad;
            this.precio = data.precio;
            this.mostrarDatosProducto = true;
        })
        .catch(error => {
            alert('Error al obtener los datos del producto.');
        });
},
```

Dentro del primer then, verificamos si la respuesta de la solicitud es exitosa (código de respuesta 200-299). Si es así, utilizamos response.json() para parsear la respuesta en formato JSON. Si la respuesta es un error, lanzamos una excepción para ser "catcheada" más adelante en el catch.

Dentro del segundo then, recibimos el objeto data que contiene los datos del producto en formato JSON. Luego, actualizamos los campos de los formularios de modificación con los valores obtenidos de data. También hacemos los ajustes necesarios para mostrar el formulario de modificación y ocultar el formulario de consulta.

En caso de que ocurra algún error durante la solicitud GET, el catch se ejecutará y se mostrará una alerta de error.

El método guardarCambios se ejecuta cuando se envía el formulario de modificación. En este método, creamos un objeto producto con los datos actualizados del formulario y utilizamos fetch para realizar una solicitud PUT a la API y guardar los cambios.

guardarCambios():

```
guardarCambios() {
    const producto = {
        codigo: this.codigo,
        descripcion: this.descripcion,
        cantidad: this.cantidad,
        precio: this.precio
    };

    fetch(URL + 'productos/' + this.codigo, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(producto)
    })
        .then(response => {
            if (response.ok) {
                return response.json();
            } else {
                throw new Error('Error al guardar los cambios del producto.');
            }
        })
        .then(data => {
            alert('Cambios guardados correctamente.');
            location.reload();
        })
        .catch(error => {
            alert('Error al guardar los cambios del producto.');
```

```
        });
}
```

Al igual que antes, verificamos si la respuesta de la solicitud PUT es exitosa. Si es así, utilizamos response.json() para parsear la respuesta en formato JSON. Si la respuesta es un error, lanzamos una excepción.

Dentro del segundo then, mostramos una alerta de que los cambios se han guardado correctamente y recargamos la página para volver al formulario de consulta.

En caso de que ocurra algún error durante la solicitud PUT, el catch se ejecutará y se mostrará una alerta de error.

```
app.mount('#app');
```

Finalmente, utilizamos app.mount('#app') para montar nuestra aplicación Vue en el contenedor con el id "app" en el HTML.

PD: Se comparte tambien el archivo **modificaciones_sinVue3.html** que hace exactamente lo mismo, pero sin usar Vue3.

Gestion del carrito de compras

Esta es la sección que tiene como tarea mostrar el "catálogo" de productos que existen en la base de datos, agregando botones para sumar o quitar un elemento del carrito de compras.

Recuerda que es una ejemplo de como usar la API. Un fronted real debería implementarse con botones que permitan eliminar productos directamente desde la vista del carrito, sumar o quitar mas de una unidad a la vez, etcétera. Pero hemos intentado mantener el ejemplo lo más simple posible.

carrito.html:

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inventario</title>
  <script src="https://unpkg.com/vue@next"></script>
  <link rel="stylesheet" href="estilos.css">
</head>

<body>
  <div id="app">
    <h1>Carrito de compras</h1>
    <h3>Codo a Codo 2023</h3>
    <table>
```

```
    <thead>
      <tr>
        <th>Código</th>
        <th>Descripción</th>
        <th>Cantidad</th>
        <th>Precio</th>
        <th>Carrito</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="producto in productos" :key="producto.codigo">
        <td>{{ producto.codigo }}</td>
        <td>{{ producto.descripcion }}</td>
        <td align="right">{{ producto.cantidad }}</td>
        <td align="right">    {{ producto.precio }}</td>
        <td>
          <button @click="agregarAlCarrito(producto)">  <b>+</b>  </button>
          <button @click="restarDelCarrito(producto)">  <b>-</b>  </button>
        </td>
      </tr>
    </tbody>
  </table>

  <div v-if="mostrarCarrito">
    <h3>Productos en el carrito:</h3>
    <table>
      <thead>
        <tr>
          <th>Código</th>
          <th>Descripción</th>
          <th>Cantidad</th>
          <th>Precio</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="item in carrito" :key="item.codigo">
          <td>{{ item.codigo }}</td>
          <td>{{ item.descripcion }}</td>
          <td align="right">{{ item.cantidad }}</td>
          <td align="right">    {{ item.precio }}</td>
        </tr>
      </tbody>
    </table>
  </div>

  <div v-if="!mostrarCarrito" class="contenedor-centrado">
    <button @click="obtenerCarrito">Mostrar carrito</button>
  </div>

  <div class="contenedor-centrado">
    <a href="index.html">Menu principal</a>
  </div>


</div>

<script>
  //const URL = "http://127.0.0.1:5000/"
  const URL = "https://xxxxx.pythonanywhere.com/"

  const app = Vue.createApp({
    data() {
      return {
        productos: [],
        mostrarCarrito: false,
        carrito: [],
      };
    },
    created() {
      this.obtenerProductos();
    },
    methods: {
      obtenerProductos() {
        fetch(URL + 'productos')
          .then(response => response.json())
          .then(data => {
            this.productos = data;
          })
          .catch(error => {
            console.error(URL + 'productos', error);
            alert('Error al obtener los productos.');
          });
      },
      agregarAlCarrito(producto) {
        fetch(URL + 'carrito', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({
```

```
              codigo: producto.codigo,
              cantidad: 1, // Agregamos una unidad al carrito
            }),
          })
            .then(response => response.json())
            .then(data => {
              alert(data.message);
            })
            .catch(error => {
              console.error('Error al agregar el producto al carrito:', error);
              alert('Error al agregar el producto al carrito.');
            });
        },
        restarDelCarrito(producto) {
          fetch(URL + 'carrito', {
            method: 'DELETE',
            headers: {
              'Content-Type': 'application/json',
            },
            body: JSON.stringify({
              codigo: producto.codigo,
              cantidad: 1, // Restamos una unidad del carrito
            }),
          })
            .then(response => response.json())
            .then(data => {
              alert(data.message);
            })
            .catch(error => {
              console.error('Error al restar el producto del carrito:', error);
              alert('Error al restar el producto del carrito.');
            });
        },
        obtenerCarrito() {

          fetch(URL + 'carrito')
            .then(response => response.json())
            .then(data => {
              this.carrito = data;
              this.mostrarCarrito = true;
            })
            .catch(error => {
              console.error('Error al obtener el carrito:', error);
              alert('Error al obtener el carrito.');
            });
        },
      },
    });

    app.mount('#app');
  </script>
</body>

</html>
```

En el HTML, hemos incluido el CDN de Vue 3 y hemos creado un contenedor con el id "app" donde se montará nuestra aplicación Vue. Dentro del contenedor, tenemos una tabla que muestra los productos disponibles en el inventario, y un botón para mostrar el carrito de compras.

En el script, comenzamos por definir la constante URL, que representa la URL de la API utilizada para obtener y modificar los datos de los productos y el carrito.

Luego, utilizamos Vue.createApp para crear nuestra aplicación Vue. Dentro de data(), definimos las propiedades que serán utilizadas en nuestra aplicación: productos para almacenar la lista de productos del inventario, mostrarCarrito para controlar la visibilidad del carrito de compras, y carrito para almacenar los productos seleccionados en el carrito.

En la sección created(), definimos el método obtenerProductos que se ejecutará cuando la aplicación Vue sea creada. En este método, utilizamos fetch para realizar una solicitud GET a la API y obtener la lista de productos del inventario.

obtenerProductos():

```
obtenerProductos() {
    fetch(URL + 'productos')
        .then(response => response.json())
        .then(data => {
            this.productos = data;
        })
        .catch(error => {
            console.error(URL + 'productos', error);
            alert('Error al obtener los productos.');
        });
},
```

Dentro de la cadena de promesas del fetch, utilizamos el primer then para parsear la respuesta en formato JSON y asignarla a la propiedad productos de nuestra aplicación Vue.

En caso de que ocurra algún error durante la solicitud GET, el catch se ejecutará y se mostrará una alerta de error.

Luego, en la sección methods, definimos los métodos agregarAlCarrito, restarDelCarrito y obtenerCarrito. El método agregarAlCarrito se ejecuta cuando se hace clic en el botón "+" de un producto. En este método, utilizamos fetch para realizar una solicitud POST a la API y agregar el producto al carrito.

agregarAlCarrito(producto);

```
agregarAlCarrito(producto) {
    fetch(URL + 'carrito', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            codigo: producto.codigo,
            cantidad: 1, // Agregamos una unidad al carrito
        }),
    })
        .then(response => response.json())
        .then(data => {
            alert(data.message);
        })
        .catch(error => {
            console.error('Error al agregar el producto al carrito:', error);
            alert('Error al agregar el producto al carrito.');
        });
},
```

Dentro de la cadena de promesas del fetch, utilizamos el primer then para parsear la respuesta en formato JSON y mostrar una alerta con el mensaje devuelto por la API.

En caso de que ocurra algún error durante la solicitud POST, el catch se ejecutará y se mostrará una alerta de error.

El método restarDelCarrito es similar al método agregarAlCarrito, pero realiza una solicitud DELETE a la API para restar una unidad del producto en el carrito.

El método obtenerCarrito se ejecuta cuando se hace clic en el botón "Mostrar carrito". En este método, utilizamos fetch para realizar una solicitud GET a la API y obtener los productos del carrito.

obtenerCarrito():

```
obtenerCarrito() {
    fetch(URL + 'carrito')
        .then(response => response.json())
        .then(data => {
            this.carrito = data;
            this.mostrarCarrito = true;
        })
        .catch(error => {
            console.error('Error al obtener el carrito:', error);
            alert('Error al obtener el carrito.');
        });
},
```

Dentro de la cadena de promesas del fetch, utilizamos el primer then para parsear la respuesta en formato JSON y asignarla a la propiedad carrito de nuestra aplicación Vue. Además, establecemos la propiedad mostrarCarrito en true para mostrar el carrito en el HTML.

En caso de que ocurra algún error durante la solicitud GET, el catch se ejecutará y se mostrará una alerta de error.

Finalmente, utilizamos app.mount('#app') para montar nuestra aplicación Vue en el contenedor con el id "app" en el HTML.

Hoja de estilos:

Todos los archivos HTML mostrados utilizan el mismo código CSS:

estilos.css:

```
/* Estilos para todo el proyecto */
body {
    font-family: Arial, sans-serif;
    background-color: #f9f9f9;
}

.contenedor-centrado {
    display:flex;
    width: 100%;
    justify-content: center;
}

p {
    font-family: 'Times New Roman', Times, serif;
    background-color: #f9f9f9;
    max-width: 400px;
    margin: 0 auto;
    padding: 20px;
}

h1, h2, h3, h4, h5, h6 {
    text-align: center;
    color: #007bff;
}

form {
    max-width: 400px;
    margin: 0 auto;
    padding: 20px;
    background-color: white;
    border: 1px solid lightslategray;
    border-radius: 5px;
}

table {
    max-width: 90%;
    margin: 0 auto;
```

```
    padding: 20px;
    background-color: white;
    border: 1px solid lightslategray;
    border-radius: 5px;
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"],
input[type="number"],
textarea {
    width: 90%;
    padding: 10px;
    margin-bottom: 10px;
    border: 1px solid #cccccc;
    border-radius: 5px;
}

input[type="submit"] {
    padding: 10px;
    background-color: #007bff;
    color: #ffffff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

input[type="submit"]:hover {
    background-color: #0056b3;
}

button {
    padding: 8px;
    margin:4px;
    background-color: #007bff;
    color: #ffffff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

a {
    padding: 8px;
    margin:4px;
    background-color: #fdab13;
    color: #ffffff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    text-decoration: none;
    font-size: 85%;
}

a:hover {
    background-color: #cd7b00;
}
```

Breve descripción de cada estilo implementado:

- **body:** Aplica estilos al elemento <body> de la página. Establece la fuente del texto como Arial o una fuente genérica sin serifa (sans-serif) y establece el color de fondo como #f9f9f9.
- **.contenedor-centrado:** Aplica estilos a un contenedor específico utilizando la clase .contenedor-centrado. Hace uso de flexbox para centrar horizontalmente su contenido y establece su ancho como el 100% del contenedor padre.
- **p:** Aplica estilos a los elementos <p> (párrafos). Establece la fuente del texto como 'Times New Roman', Times, serif, establece el color de fondo como #f9f9f9, limita el ancho máximo a 400px y centra el elemento horizontalmente utilizando los márgenes automáticos. Además, se añade un relleno interno de 20px.
- **h1, h2, h3, h4, h5, h6:** Aplica estilos a los elementos de encabezado del nivel 1 al 6 (<h1> a <h6>). Los alinea al centro y establece el color del texto como #007bff (un tono de azul).
- **form:** Aplica estilos a los elementos <form> (formularios). Establece el ancho máximo a 400px, centra el formulario horizontalmente utilizando los márgenes automáticos y añade un relleno interno de 20px. Además, establece el color de fondo como blanco, agrega un borde de 1px sólido en lightslategray y un borde redondeado de 5px.
- **table:** Aplica estilos a los elementos <table> (tablas). Establece el ancho máximo al 90%, centra la tabla horizontalmente utilizando los márgenes automáticos y añade un relleno interno de 20px. Además, establece el color de fondo como blanco, agrega un borde de 1px

sólido en lightslategray y un borde redondeado de 5px.

- **label:** Aplica estilos a los elementos <label> (etiquetas). Los muestra como bloques y agrega un margen inferior de 5px.
- **input[type="text"], input[type="number"], textarea:** Aplica estilos a los elementos de entrada de tipo texto, número y área de texto (<input type="text">, <input type="number"> y <textarea>). Establece el ancho al 90%, agrega un relleno de 10px, un margen inferior de 10px, un borde de 1px sólido en #cccccc y un borde redondeado de 5px.
- **input[type="submit"]:** Aplica estilos al botón de envío (<input type="submit">). Agrega un relleno de 10px, establece el color de fondo como #007bff, el color del texto como blanco, elimina el borde y añade un borde redondeado de 5px. Además, cambia el cursor del mouse a un puntero cuando se pasa sobre el botón.
- **input[type="submit"]:hover:** Aplica estilos al botón de envío cuando se pasa el cursor sobre él. Cambia el color de fondo a #0056b3.
- **button:** Aplica estilos a los elementos <button> (botones). Agrega un relleno de 8px, un margen de 4px, establece el color de fondo como #007bff, el color del texto como blanco, elimina el borde y añade un borde redondeado de 5px. Además, cambia el cursor del mouse a un puntero cuando se pasa sobre el botón.
- **button:hover:** Aplica estilos a los botones cuando se pasa el cursor sobre ellos. Cambia el color de fondo a #0056b3.
- **a:** Aplica estilos a los elementos <a> (enlaces). Agrega un relleno de 8px, un margen de 4px, establece el color de fondo como #fdab13, el color del texto como blanco, elimina el borde y añade un borde redondeado de 5px. Además, cambia el cursor del mouse a un puntero cuando se pasa sobre el enlace y establece el tamaño de fuente al 85%.
- **a:hover:** Aplica estilos a los enlaces cuando se pasa el cursor sobre ellos. Cambia el color de fondo a #cd7b00.