Enviar archivos en JSON codificados con Base64

Si nuestro servidor requiere que le pasemos todos los datos en formato JSON, los archivos deben ser convertidos a string. Para ello los codificaremos utilizando el formato Base64.

```
window.addEventListener("load", (e) => {
   ...

   // Cuando se seleccione un archivo, lo procesamos
   document.getElementById("photo").addEventListener('change',  () => {
      var file   = document.getElementById("photo").files[0];
      var reader = new FileReader();

      reader.addEventListener("load", () => { //Evento de conversión a Base64 completa (asíncrono)
         imagePreview.src = reader.result; // Mostramos la imagen cargada en un elemento <img>
      }, false);

      if (file) { // Si se ha seleccionado un archivo válido (convertir a Base64)
         reader.readAsDataURL(file);
      }
   });
});
```

De esta manera, podemos incluir el archivo dentro de un objeto JSON como un dato más. En el servidor se decodificará y se guardará otra vez como archivo:

```
let prod = {
    name: document.getElementById("name").value,
    description: document.getElementById("description").value,
    photo: imagePreview.src
};

fetch(`${SERVER}/products`, {
   method: 'POST',
   body: JSON.stringify(prod),
   headers: {
      'Content-Type': 'application/json'
   }
});
```

Ejemplos

1. Ejemplo de una petición **GET**. Recibiremos una respuesta JSON con un array de productos, que recorreremos y crearemos la estructura HTML de cada producto para añadir al DOM.

```
function getProducts() {
   fetch(`${SERVER}/products`).then(resp => {
      if(!resp.ok) throw new Error(resp.statusText);
      return resp.json(); // El método json() devuelve una promesa
   }).then(respJSON => {
      respJSON.products.forEach(p => appendProduct(p));
   }).catch(error => console.error(error));
}

function appendProduct(product) {
   let tbody = document.querySelector("tbody");
```

```
    let tr = document.createElement("tr");
    // Imagen
    let imgTD = document.createElement("td");
    let img = document.createElement("img");
    img.src = `${SERVER}/img/${product.photo}`;
    imgTD.appendChild(img);

    // Nombre
    let nameTD = document.createElement("td");
    nameTD.textContent = product.name;

    // Descripción
    let descTD = document.createElement("td");
    descTD.textContent = product.description;

    tr.appendChild(imgTD);
    tr.appendChild(nameTD);
    tr.appendChild(descTD);
    tbody.appendChild(tr);
}
```

2. Ejemplo de llamada **POST**. Los datos se envían al servidor en formato JSON. Recibiremos a su vez otra respuesta JSON con el producto insertado. Si todo va bien, el producto recibido se insertará en el DOM.

```
function addProduct() {
    let prod = {
        name: document.getElementById("name").value,
        description: document.getElementById("description").value,
        photo: imagePreview.src
    };

    fetch(`${SERVER}/products`, {
        method: 'POST',
        body: JSON.stringify(prod),
        headers: {
            'Content-Type': 'application/json'
        }
    }).then(resp => {
        if(!resp.ok) throw new Error(resp.statusText);
        return resp.json(); // promise
    }).then(respJSON => {
        appendProduct(respJSON.product);
    }).catch(error => console.error(error));
}
```

Encapsular llamadas AJAX con clases

Para facilitar un poco más las llamadas AJAX al servidor, podemos crearnos una clase con unos cuantos métodos estáticos.

```
"use strict";

class Http {
    ajax(method, url, headers = {}, body = null) {
        return fetch(url, { method, headers, body})
            .then(resp => {
```

```
            if(!resp.ok) throw new Error(resp.statusText);
            return resp.json();
        });
    }

    get(url) {
        return Http.ajax('GET', url);
    }

    post(url, data) {
        return Http.ajax('POST', url, {
            'Content-Type': 'application/json'
        }, JSON.stringify(data));
    }

    put(url, data) {
        return Http.ajax('PUT', url, {
            'Content-Type': 'application/json'
        }, JSON.stringify(data));
    }

    delete(url) {
        return Http.ajax('DELETE', url);
    }
}
```

También es buena idea crear clases intermediarias para gestionar las peticiones al servidor. Por ejemplo, vamos a crear una clase que se encargue de gestionar lo relacionado con productos:

```
class ProductService {
    constructor() {
        this.http = new Http();
    }

    getProducts() {
        return this.http.get(`${SERVER}/products`).then((response) => {
            return response.products.map(prod => new Product(prod));
        });
    }

    add(product) {
        return this.http.post(`${SERVER}/products`, product)
            .then((response) => {
                return response.product;
            });
    }

    update(product) {
        return Http.put(`${SERVER}/products/${product.id}`, product)
            .then((response) => {
                return new response.product;
            });
    }

    delete(id) {
        return Http.delete(`${SERVER}/products/${id}`);
    }
}
```

En el archivo JavaScript principal el código ahora queda más limpio:

```
let productService = new ProductService();

function getProducts() { // Obtener productos y añadirlos al DOM
   productService.getProducts().then(prods => {
      products = prods;
      let tbody = document.querySelector("tbody");
      products.forEach(p => {
         appendProduct(p);
      });
   }).catch(error => alert(error.toString()));
}

function addProduct() { // Añadir un producto al servidor e insertarlo en el DOM (tabla)
   let prod = {
      name: document.getElementById("name").value,
      description: document.getElementById("description").value,
      photo: imagePreview.src
   };

   productService.add(prod).then(pResp => { // Recibimos el producto añadido a la base de datos
      let tbody = document.querySelector("tbody");
      appendProduct(pResp);
   }).catch(error => alert(error.toString()));
}
```

Enviar formularios con archivos usando FormData

Si necesitamos enviar un formulario que contiene archivos por AJAX, podemos utilizar la clase FormData. El servidor recibirá los archivos por separado y el resto de la información en formato **urlencoded**.

```
<form id="addProduct">
   <p><input type="text" name="name" id="name" placeholder="Product's name" required></p>
   <p><input type="text" name="description" id="description" placeholder="Description" required></p>
   <p>Photo: <input type="file" name="photo" id="photo" required></p>
   <button type="submit">Add</button>
</form>
```

Podemos instanciar el objeto FormData y añadirle los valores manualmente:

```
let formData = new FormData();
formData.append("name", document.getElementById("name").value);
formData.append("description", document.getElementById("description").value);
formData.append("photo", document.getElementById("photo").files[0]);

fetch(`${SERVER}/products`, {
   method: 'POST',
   body: formData
})
```

O podemos añadirle el formulario entero con todo lo que contiene (después podríamos seguir añadiendo más datos si queremos):

```
fetch(`${SERVER}/products`, {
   method: 'POST',
   body: new FormData(document.getElementById("addProduct")),
   headers: {
      'Content-Type': 'application/json'
   }
})
```