Subida de archivos al servidor

Datos

{desafío} latam_

Implementar la funcionalidad de subida de archivos a un servidor utilizando Express de acuerdo al entorno Node.js.

- Unidad 1:
  API REST
- Unidad 2:
  Subida de archivos al servidor
- Unidad 3:
  JWT

Te encuentras aquí

{desafío}
latam_

¿Qué aprenderás en esta sesión?

- *Utiliza archivos variados subidos por Upload File en un proyecto Express.*

¿Qué acciones podríamos realizar con los datos obtenidos desde una API REST?

/* Filtros */

Filtros

Siguiendo con el pensamiento industrial, en donde aplicamos todas estas herramientas en escenarios donde se tienen cientos o miles de datos, filtrar las consultas a la API REST resulta una práctica casi indispensable en cualquier servicio WEB.

Para ofrecer filtros en nuestra API tenemos que procesar los datos que disponemos para devolverle al usuario lo que nos pide en una consulta donde especifica la información que necesita.

Filtros

Ejemplo de esto es el filtro que aplica la pokeapi con los tipos de pokemones, en donde puedes hacer una consulta con la siguiente URL https://pokeapi.co/api/v2/type/fire y esto te devolverá una respuesta filtrando todos los pokemones y devolviendo solo los de tipo fuego

Filtros


```
288      "name": "fire",
289    ▸ "names": [↔],
340    ▾ "pokemon": [
341    ▾   {
342    ▾     "pokemon": {
343            "name": "charmander",
344            "url": "https://pokeapi.co/api/v2/pokemon/4/"
345          },
346          "slot": 1
347        },
348    ▾   {
349    ▾     "pokemon": {
350            "name": "charmeleon",
351            "url": "https://pokeapi.co/api/v2/pokemon/5/"
352          },
353          "slot": 1
354        },
```

Ejercicio guiado:

Guitarras Filtradas

Ejercicio guiado

Guitarras filtradas

Agregar una ruta **GET /api/v2/body/:cuerpo** que reciba como parámetro el tipo de cuerpo de las guitarras (campo “body”) y devuelva solo las guitarras que cumplan con esa especificación.
Sigue los pasos para realizar este ejercicio:

- **Paso 1**: Crear una función de nombre “filtroByBody” que reciba un parámetro “body”.

- **Paso 2**: Utilizar el método “filter” para retornar solo las guitarras que tengan como cuerpo el mismo declarado como parámetro en la función.

Ejercicio guiado

Guitarras filtradas

- **Paso 3**: Crear una ruta **GET /api/v2/body/:cuerpo**
- **Paso 4**: Guardar en una constante el parámetro “cuerpo”
- **Paso 5**: Devolver al cliente un objeto con las propiedades “cant” y “guitarras”. Utiliza la función “filtroByBody” para darle valor a las propiedades.

Ejercicio guiado

Guitarras filtradas

```
// Paso 1
const filtroByBody = (body) => {
  // Paso 2
  return guitarras.filter((g) => g.body === body);
};

// Paso 3
app.get("/api/v2/body/:cuerpo", (req, res) => {
  // Paso 4
  const cuerpo = req.params.cuerpo;
  // Paso 5
  res.send({
    cant: filtroByBody(cuerpo).length,
    guitarras: filtroByBody(cuerpo),
  });
});
```

Ejercicio guiado

Guitarras filtradas

Ahora consulta la ruta http://localhost:3000/api/v2/body/Telecaster para probar el filtro que hemos creado, deberás obtener algo como la imagen que te muestro a continuación:


```
{
  "cant": 2,
  "guitarras": [
    {
      "id": 1,
      "name": "Dean 350f",
      "brand": "DEAN",
      "model": "350f",
      "body": "Stratocaster",
      "color": "Dark blue",
      "pickups": "Sigle Coil",
      "strings": 6,
      "value": 350,
      "stock": 2
    },
    {
      "id": 2,
      "name": "Ibanez RG8570Z",
      "brand": "IBANEZ",
      "model": "RG8570Z",
      "body": "Stratocaster",
      "color": "Green",
      "pickups": "Sigle Coil",
      "strings": 7,
      "value": 1200,
      "stock": 6
    }
  ]
}
```

/* Ordenación de los datos */

Ordenación de los datos

La ordenación de los datos se trata como su nombre lo índica, de reorganizar la data solicitada por el cliente de la manera en la que es especificado en la misma consulta. Lo más común de ver es el ordenamiento en algún campo numérico de forma ascendente o descendente.

Aunque es posible un caso de uso en el que se solicite un ordenamiento por campos textuales; lo que sería un ordenamiento alfabético.

De cualquier forma la manera en la que el usuario especifica esta organización es por medio de las Query Strings, es decir, en la propia URL especificará con parámetros el orden en el que recibir los datos.

Ejercicio guiado:

Guitarras ordenadas

Ejercicio guiado

Guitarras ordenadas

Crear una función que reciba un parámetro “order” y permita ordenar el arreglo de guitarras por su propiedad “value” de forma descendiente o ascendiente.

El objetivo será ofrecer las siguientes rutas:

- http://localhost:3000/api/v2/guitarras?values=desc
- http://localhost:3000/api/v2/guitarras?values=asc

Ejercicio guiado

Guitarras ordenadas

El primer endpoint devolvería las guitarras en un orden descendente según su valor y el segundo de forma ascendente.

Incluir en la ruta **/api/v2/guitarras** la extracción de un parámetro “values”, obtenido por Query Strings y utilizarlo en condiciones para devolver el arreglo de guitarras correspondiente a “asc” o “desc”.

Ejercicio guiado

Guitarras ordenadas

- **Paso 1**: Crear una función de nombre “orderValues” que reciba un parámetro “order”.
- **Paso 2**: Retornar en la función un operador ternario que evalúe si el valor del parámetro es “asc” o “desc” y devuelva el arreglo de guitarras ordenado según el valor recibido. Utiliza el método sort para esto.
- **Paso 3**: Extraer de la Query String el parámetro “values” en la ruta **/api/v2/guitarras**.
- **Paso 4**: Utilizar condicionales “if” para devolver la ejecución de la función “orderValues” en caso de que el valor del parámetro “values” sea “asc” o “desc”.

Ejercicio guiado

Guitarras ordenadas

```
// Paso 1
const orderValues = (order) => {
  // Paso 2
  return order == "asc"
    ? guitarras.sort((a, b) => (a.value > b.value ? 1 : -1))
    : order == "desc"
    ? guitarras.sort((a, b) => (a.value < b.value ? 1 : -1))
    : false;
};

app.get("/api/v2/guitarras", (req, res) => {
  // Paso 3
  const { values } = req.query;
  // Paso 4
  if (values == "asc") return res.send(orderValues("asc"));
  if (values == "desc") return res.send(orderValues("desc"));
  res.send({
    guitarras: HATEOASV2(),
  });
});
```

Ejercicio guiado

Guitarras ordenadas

Para probar este ordenamiento, consulta la siguiente dirección:

http://localhost:3000/api/v2/guitarras?values=asc

Y deberás recibir lo que te muestro en la siguiente imagen:


```
[
  {
    "id": 3,
    "name": "Epiphone Les Paul LP-100",
    "brand": "EPIPHONE",
    "model": "LP-100",
    "body": "LES PAUL",
    "color": "Gray",
    "pickups": "hambucker",
    "strings": 6,
    "value": 309,
    "stock": 8
  },
  {↔},
  {↔},
  {
    "id": 4,
    "name": "Fender American Performer",
    "brand": "FENDER",
    "model": "American Performen",
    "body": "Telecaster",
    "color": "White",
    "pickups": "Sigle Coil",
    "strings": 6,
    "value": 1329,
    "stock": 4
  }
]
```

/* Selección de campos */

Selección de campos

La selección de campos se refiere al filtrado de las propiedades pertenecientes al recurso que se consulta en una API REST, es decir, podemos tener un objeto con 5 propiedades distintas pero el usuario podría solo necesitar 2.

Para lograr esto necesitaremos nuevamente las Query Strings, porque es en la URL en donde se especificarán cuáles son los campos requeridos por el cliente.

Ejercicio guiado:

Seleccionando campos

Ejercicio guiado

Seleccionando campos

Crear una función que filtre las propiedades de la guitarra deseada, dejando solo las que sean especificadas en la URL. Esto lo sabremos al obtener un parámetro “fields” dentro de la Query String, el objetivo es poder ofrecer la siguiente ruta:

- http://localhost:3000/api/V2/guitarra/1?fields=id,name,brand,color

Como puedes observar los campos los separaremos por comas (,). Al recibir esta consulta en el servidor se deberá devolver la guitarra de id 1, pero solamente los campos id, name, brand y color.

Ejercicio guiado

Seleccionando campos

- **Paso 1**: Crear una función de nombre “fieldsSelect” que reciba dos parámetros “guitarra” y “fields”. El objetivo de esta función será devolver la misma guitarra recibida pero solo con los campos recibidos como segundo parámetro.

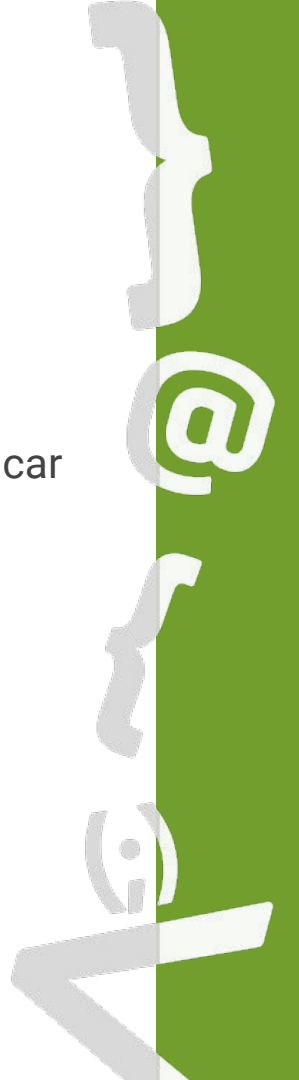
- **Paso 2**: Utilizar un “for in” para recorrer la guitarra recibida y dentro un condicional “if” que evalúe con el método “includes” si alguna propiedad no coincide con los campos recibidos como segundo parámetro, en caso de ser así, deberás eliminar esa propiedad. El objetivo de esto, es recortar el objeto y dejar solo las propiedades declaradas por el cliente.

Ejercicio guiado

Seleccionando campos

- **Paso 3**: Crear una ruta GET /api/v2/guitarra/:id.

- **Paso 4**: Extraer el parámetro “id” de la ruta, el cual utilizaremos para especificar la guitarra que devolveremos con los atributos procesados.



Ejercicio guiado

Seleccionando campos

- **Paso 5**: Utilizar un condicional “if” que verifique si se está recibiendo en la Query String el parámetro “fields” y retorne un objeto con una propiedad “guitarra”, cuyo valor debe ser la ejecución de la función “fieldSelect”, sabiendo que esta función espera recibir como parámetros la instancia de una guitarra y los campos que seleccionará, envía como primer argumento la ejecución de la función “guitarra”, y como segundo argumento el parámetro “fields” recibido en Query Strings dividido(split) por el carácter coma “,”.

Ejercicio guiado

Seleccionando campos

```
// Paso 1
const fieldsSelect = (guitarra, fields) => {
  // Paso 2
  for (propiedad in guitarra) {
    if (!fields.includes(propiedad)) delete guitarra[propiedad];
  }
  return guitarra;
};
// Paso 3
app.get("/api/v2/guitarra/:id", (req, res) => {
  // Paso 4
  const { id } = req.params;
  // Paso 5
  const { fields } = req.query;
if (fields) return res.send({ guitarra: fieldsSelect(guitarra(id), fields.split(",") ) });
  res.send({
    guitarra: guitarra(id),
  });
});
```

Ejercicio guiado

Seleccionando campos

Ahora intenta consultar la siguiente ruta :

http://localhost:3000/api/V2/guitarra/1?fields=id,name,brand,color

Y deberás recibir la siguiente imagen:


```
{
  "guitarra": {
    "id": 1,
    "name": "Dean 350f",
    "brand": "DEAN",
    "color": "Dark blue"
  }
}
```

/* Paginación de datos */

Paginación de datos

La paginación de datos en una API REST se refiere a la devolución particionada de los recursos que están alojados en el lado del servidor. La Pokeapi una vez más nos sirve como ejemplo, en ella también aplica esta buena práctica, ya que al consultar el endpoint https://pokeapi.co/api/v2/pokemon/ recibimos 2 propiedades “next” y “previous” claves de la paginación.


```
{
  "count": 964,
  "next": "https://pokeapi.co/api/v2/pokemon/?offset=40&limit=20",
  "previous": "https://pokeapi.co/api/v2/pokemon/?offset=0&limit=20",
  "results": [↔]
}
```

Ejercicio guiado:

Guitarras por página

Ejercicio guiado

Guitarras por página

Agregar dentro de la ruta **api/V2/guitarras** un condicional que evalúa si existe el parámetro “page”, de ser así, almacenar el valor en una variable y utilizar el método slice para recortar el arreglo de guitarras devolviendo siempre solo 2 por página.

El objetivo será poder disponibilizar el siguiente endpoint:

- http://localhost:3000/api/V2/guitarras?page=2

Ejercicio guiado

Guitarras por página

Sigue los pasos para realizar este ejercicio modificando la lógica de la ruta **api/V2/guitarras**:

- **Paso 1**: Agregar un condicional “if” que evalúe si existe el parámetro “page” dentro de la Query Strings.

- **Paso 2**: Guardar en una constante el parámetro “page”.

Ejercicio guiado

Guitarras por página

- **Paso 3**: Responder al cliente un objeto que contenga la ejecución de la función **HATEOASV2** con el método slice concatenado para devolver solo 2 guitarras según la página declarada por el cliente. Ocupa la siguiente fórmula para definir el inicio y final del método slice:

    - Inicio: (número de página) * (cantidad de recursos) - (cantidad de recursos)
        - En nuestro ejercicio sería page * 2 - 2
    - Final: (número de página) * (cantidad de recursos)
        - En nuestro ejercicio sería page * 2

Ejercicio guiado

Guitarras por página

```
app.get("/api/v2/guitarras", (req, res) => {
 const { values } = req.query;
 if (values == "asc") return res.send(orderValues("asc"));
 if (values == "desc") return res.send(orderValues("desc"));
 // Paso 1
 if (req.query.page) {
   // Paso 2
   const { page } = req.query;
   // Paso 3
   return res.send({ guitarras: HATEOASV2().slice(page * 2 - 2, page * 2) });
 }
 res.send({
   guitarras: HATEOASV2(),
 });
});
```

Ejercicio guiado

Guitarras por página

Ahora consulta la siguiente dirección para obtener las guitarras correspondientes a la página 2 http://localhost:3000/api/V2/guitarras?page=2

Deberás recibir lo que te muestro en la siguiente imagen:


```
{
  "guitarras": [
    {
      "guitar": "Epiphone Les Paul LP-100",
      "src": "http://localhost:3000/guitarra/3"
    },
    {
      "guitar": "Fender American Performer",
      "src": "http://localhost:3000/guitarra/4"
    }
  ]
}
```

/* Uso del payload para el manejo de errores con códigos de estado HTTP */

Uso del payload para el manejo de errores con códigos de estado HTTP

Los códigos de estado HTTP normalmente son generados por defecto sin tener que programarlos. El problema con esto, es que no controlamos el estado de la respuesta del servidor, por lo que el usuario podría recibir un código ambiguo y sin una descripción detallada sobre qué sucedió e incluso cómo podría resolver el problema.

Uso del payload para el manejo de errores con códigos de estado HTTP

Un ejemplo de esto, es lo que hace la API https://mindicador.cl/ cuando recibe una consulta de un indicador basado en una fecha pero ésta no existe o es incorrecta. En la siguiente imagen te muestro el payload que recibimos de esta API al consultar este endpoint https://mindicador.cl/api/dolar/003-01-2020.


```
{
  "error": "500 Internal Server Error",
  "message": "Fecha incorrecta",
  "description": null
}
```

Uso del payload para el manejo de errores con códigos de estado HTTP

Devolver errores personalizados amerita estudiar una gran cantidad de casos de uso, poniéndose en los zapatos del cliente y preguntándose ¿Qué mensaje sería el ideal para dar a entender de la mejor manera qué sucedió?.

En Express contamos con el método “status” del objeto response para definir nosotros mismos el error de estado HTTP, por ejemplo si quisiéramos responder que el recurso solicitado no existe, podríamos ocupar la siguiente instrucción como respuesta de la ruta:

```
res.status(404).send(
   {
      error: "404 Not Found",
      message: "No existe el recurso que
solicita"
   }
);
```

Ejercicio guiado:

¡Error! Guitarra no encontrada

Ejercicio guiado

¡Error! Guitarra no encontrada

Modificar la ruta **api/v2/guitarra/:id** agregando un operador ternario que evalúa si existe la guitarra según el id recibido en la ruta, de no ser así devolver el siguiente objeto:

```
{
  error: "404 Not Found",
  message: "No existe una guitarra con ese ID",
};
```

En el cual tenemos una propiedad “error” con el texto “404 Not Found” y en una propiedad “message” con el texto “No existe una guitarra con ese ID”

Ejercicio guiado

¡Error! Guitarra no encontrada

- **Paso 1**: Crear un operador ternario que evalúe si la guitarra con el id recibido en la ruta existe.
- **Paso 2**: En el caso exitoso, devuelve la guitarra encontrada.
- **Paso 3**: En el caso de fracaso, devuelve el objeto indicando el estado 404 y el mensaje “No existe una guitarra con ese ID”.

Ejercicio guiado

¡Error! Guitarra no encontrada

```
app.get("/api/v2/guitarra/:id", (req, res) => {
  const { id } = req.params;
  const { fields } = req.query;
  if (fields) return res.send({ guitarra: fieldsSelect(guitarra(id), fields.split(",") ) });
  // Paso 1
  guitarra(id)
  // Paso 2
    ? res.send({
        guitarra: guitarra(id),
      })
    : // Paso 2
      res.status(404).send({
        error: "404 Not Found",
        message: "No existe una guitarra con ese ID",
      });
});
```

Ejercicio guiado

¡Error! Guitarra no encontrada

Ahora consulta la ruta para intentar obtener la guitarra de id 10:
http://localhost:3000/api/v2/guitarra/10

Deberás recibir lo que te muestro en la siguiente imagen:


```
{
  "error": "404 Not Found",
  "message": "No existe una guitarra con ese ID"
}
```

Resumen

- Para ofrecer filtros en nuestra API tenemos que procesar los datos que disponemos para devolverle al usuario lo que nos pide en una consulta donde especifica la información que necesita.

- La ordenación de los datos se trata de reorganizar la data solicitada por el cliente de la manera en la que es especificado en la misma consulta.

- La selección de campos se refiere al filtrado de las propiedades, es decir, podemos tener un objeto con 5 propiedades distintas pero el usuario podría solo necesitar 2.

¿Porqué es importante filtrar los datos?

{desafío} latam_

¿Existe algún concepto que no se haya comprendido?

{desafío} latam_

Próxima sesión...

- *Desafío guiado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales