



Semana 2

Carlos López Lorenzo



Índice

Semana 2	1
1.Conectar la base de datos	3
2.Probar el endpoint con datos reales	4



1. Conectar la base de datos

-Sustituir los datos mockeados por datos en la base de datos:

Para esto comenzaremos creando un schéma de mongo en nuestro código con la estructura que deseemos que tengan los datos que vamos a almacenar

```
JS SchemaPersonajes > ...
1 import mongoose from "mongoose";
2
3 const personajesSchema = new mongoose.Schema({
4   name: {
5     type: String,
6     required: true,
7   },
8   Rareza: {
9     type: String,
10    required: true,
11  },
12  Costo: {
13    type: String,
14    required: true,
15  }
16 });
17
18 const Personaje = mongoose.model("Personaje", personajesSchema);
19
20 export default Personaje;
```

Este es un ejemplo que he creado en mi backend en un archivo .js que incluiré en mi código principal, con el schema creado la base de datos ya detectara o creará una tabla para este schema, luego para introducir los datos mockeados, ya que son pocos podemos introducirlos desde la base de datos manualmente, creando un método post o con el uso de plugins en caso de que sean muchos datos podemos pasarlos con el uso de código.

En mi caso he creado un método post para introducir los datos manualmente en la base de datos, con el siguiente código:

```
//Endpoint para introducir datos en la base de datos para hacer pruebas
app.post("/Personajes", async (req, res) => {
  try {
    const nuevoPersonaje = new Personaje(req.body);
    await nuevoPersonaje.save();
    res.status(201).json(nuevoPersonaje);
  } catch (error) {
    res.status(500).send("Error al agregar el Personaje");
  }
});
```



-Implementar consultas básicas como listar productos:

Para hacer la consulta usaremos el método get creado anteriormente que nos listara todos los personajes en formato json

```
app.get("/Personajes", async (req, res) => {  
  try {  
    const personajes = await Personaje.find();  
    res.json(personajes);  
  } catch (error) {  
    console.error("Error al obtener los personajes:", error);  
    res.status(500).json({ error: "Error al obtener los personajes" });  
  }  
});
```

Con este podremos hacer una consulta en postman para obtener todos los personajes

2.Probar el endpoint con datos reales

Ahora añadiremos algunos datos reales con el método post y los listamos con el método get, tal que así:

The screenshot shows the Postman interface. At the top, a POST request is configured to `http://localhost:3000/personajes`. The request body is set to JSON and contains the following data:

```
{  
  "name": "Leñador",  
  "Rareza": "Épica",  
  "Costo": 8  
}
```

Below the request, the response is displayed in the 'Body' tab, showing a successful JSON response:

```
{  
  "name": "P.E.K.K.A",  
  "Rareza": "Épica",  
  "Costo": "8",  
  "_id": "67a4709fcc8200809fcbdfef1",  
  "__v": 0  
}
```



The screenshot displays a Postman REST client interface. At the top, the URL bar shows `http://localhost:3000/personajes`. The request method is set to `GET`. The `Body` tab is selected, showing a JSON payload: `{ "name": "Leñador", "Rareza": "Legendaria", "Costo": 4 }`. The status bar at the bottom indicates a successful response: `Status: 200 OK`, `Time: 117 ms`, and `Size: 552 B`. The response body is shown in the `Body` tab, displaying a JSON array of two objects: `[{ "_id": "67a4709fcc8200009fcbdfef1", "name": "P.E.K.K.A", "Rareza": "Épica", "Costo": "0", "__v": 0 }, { "_id": "67a470cbcc8200009fcbdfef3", "name": "Leñador", "Rareza": "Legendaria", "Costo": 4 }]`.

Con estos métodos en postman podemos añadir y listar nuestros personajes.