Desarrollo de API con Node y Express

Explicando el código

```
const express = require('express')
const http = require('http')
const mongoose = require('mongoose')
const app = express()

require('./database')

const PORT = 3000

app.use(express.json())
```

- En el proyecto, hemos instalado express y mongoose para poder utilizarlos en el proyecto. Una vez hecho eso, inicializamos las constantes que vamos a requerir para poder realizar el código.
- Inicializamos la app como express y la inicializamos con app.use(express.json())
- Nos traemos el fichero database que vamos a necesitar y contiene el código:

```
const mongoose = require('mongoose')

mongoose.connect('mongodb://127.0.0.1:27017/users')
.then(() => console.log('Connected to MongoDB'))
.catch((err) => console.log('Error connecting to MongoDB', err))
```

- Este código nos sirve para podernos conectar a la BBDD inicializada en MongoDB.
- También establecemos la constante del puerto para que nos facilite el cambio de puerto si es necesario.

```
app.get('/users/ph', (req, res) => {
    const options = {
        hostname: 'jsonplaceholder.typicode.com',
        path: '/users',
        method: 'GET',
    }

    const request = http.request(options, (response) => {
        let data = ''

        response.on('data', (chunk) => {
            data += chunk
        })

        response.on('end', () => {
            const users = JSON.parse(data);
            const filteredUsers = users.map((user) => {
                const { id, name, username, email, website } = user;
                return { id, name, username, email, website };
            });
            res.json(filteredUsers);
        })
    })

    request.on('error', (error) => {
        console.error(error)
        res.status(500).json({ error: 'Error al obtener los usuarios de JSONPlaceholder' })
    })

    request.end()
})
```

- Para establecer el GET de la API tenemos que especificar la ruta en la que queremos que se pueda realizar, en este caso /users/ph.
- Le decimos las opciones que queremos, como el path y el tipo de accion que queremos realizar y lanzamos la request para que nos saque los datos.
- En este caso, he optado por hacer un filtro que solo nos saque el id, name, username, email y website ya que tienen más datos que no queremos utilizar y lo establecemos como respuesta después de aplicar el filtro.

ANTES DE SEGUIR HAGO UN COMENTARIO

A partir de este momento, el POST, DELETE y PUT no son funcionales. No porque estén mal planteados sino porque la API que se está usando no permite que se realicen esas acciones.

Aun así, al intentar realizarlas en Postman, se hace la acción como si se hubiera hecho, pero al mirar los datos no cambia nada.

```
app.post('/users/ph', (req, res) => {

    const options = {
        hostname: 'jsonplaceholder.typicode.com',
        path: '/users',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
    }

    const request = http.request(options, (response) => {
        let data = ''

        response.on('data', (chunk) => {
            data += chunk;
        })

        response.on('end', () => {
            const user = JSON.parse(data)
            res.status(201).json(user)
        })
    })

    request.on('error', (error) => {
        console.error(error)
        res.status(500).json({ error: 'Error al crear el usuario en JSONPlaceholder' })
    })

    const userData = JSON.stringify(req.body)
        request.write(userData)
        request.end()
    })
```

- Para el POST es como en el GET pero esta vez incluimos los headers en las opciones.
- Al finalizar, le decimos que escriba los datos del usuario como un usuario nuevo dentro de la API.

```
app.put('/users/ph/:id', (req, res) => {

    const options = {
        hostname: 'jsonplaceholder.typicode.com',
        path: `/users/${req.params.id}`,
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json',
        },
    }

    const request = http.request(options, (response) => {
        let data = ''

        response.on('data', (chunk) => {
            data += chunk
        })

        response.on('end', () => {
            const user = JSON.parse(data)
            res.json(user)
        })
    })

    request.on('error', (error) => {
        console.error(error)
        res.status(500).json({ error: 'Error al actualizar el usuario en JSONPlaceholder' })
    })

    const userData = JSON.stringify(req.body)
    request.write(userData)
    request.end()
})
```

- Para el PUT es muy parecido al POST, solo que en la URL le indicamos que debe ser con un id y que los datos que se escriben tienen que hacerlo sobre el path del user del que corresponda el id con el indicado en la URL.

```
app.delete('/users/ph/:id', (req, res) => {

    const options = {
        hostname: 'jsonplaceholder.typicode.com',
        path: `/users/${req.params.id}`,
        method: 'DELETE',
    }

    const request = http.request(options, (response) => {
        response.on('end', () => {
            res.sendStatus(204)
        })
    })

    request.on('error', (error) => {
        console.error(error);
        res.status(500).json({ error: 'Error al borrar el usuario en JSONPlaceholder' })
    })

    request.end()
})
```

- Para el DELETE es algo más distinto, ya que no hay que hacer una request tan compleja llamando a todos los datos. Simplemente en el user en el que el id corresponde al escrito en la URL, se envía una respuesta 204, la cual no tiene contenido.

```
app.listen(PORT, () => {
    console.log(`Server listening on port ${PORT}`)
})
```

- Por último, hacemos que nuestra app escuche en la variable PORT que inicializamos al principio del código.

Conclusiones y observaciones

Este ejercicio me ha resultado algo complicado, sobre todo en la parte de implementar una API externa, ya que es algo que no hemos visto en clase. Aun así, he conseguido implementar una la API Jsonplaceholder aunque no es la más adecuada para esta tarea ya que no permite realizar ni POST ni PUT ni DELETE.

Es bastante probable que si hubiera conseguido utilizar una API como la de Spotify hubiera podido implementar y probar todos los métodos, aun así, he realizado los métodos también fuera de la API para poder demostrar que soy capaz de hacerlos.

```
const userSchema = new mongoose.Schema({
    name: { type: String, required: true },
    username: { type: String, required: true },
    email: String,
    website: { type: String, default: 'unknown'}
})

const User = mongoose.model('User', userSchema)
```

- Primero, ceamos la constante userSchema, que es la que va a contener la estructura de los usuarios que va a contener la API.
- Indicamos que la constante User es la que va a tener ese modelo.

```
app.post('/users', async (req, res) => {
    try {
        const user = new User(req.body)
        await user.save()
        res.status(201).send(user)
    } catch (error) {
        res.status(400).send({error: error.message})
    }
})

app.get('/users', async (req, res) => {
    try {
        const users = await User.find()
        res.send(users)
    } catch(error) {
        res.status(500).send({error: error.message})
    }
})
```

- Establecemos la ruta /users para poder hacer estas peticiones, y con esto creamos el user que se estableció en el body para guardarlo en la lista de los ya existentes en el POST.
- En el GET, buscamos todos los users disponibles y los enviamos como respuesta.

```
app.delete('/users/:id', async (req, res) => {
    try {
        const user = await User.findByIdAndDelete(req.params.id)
        if (!user) {
            return res.status(400).send({error: 'User no encontrado'})
        }
        res.send(user)
    } catch(error) {
        res.status(500).send({error: error.message})
    }
})

app.put('/users/:id', async (req, res) => {
    try {
        const user = await User.findByIdAndUpdate(req.params.id, req.body)
        if (!user) {
            return res.status(400).send({error: 'Tarea no encontrada'})
        }
        res.send(user)
    } catch(error) {
        res.status(500).send({error: error.message})
    }
})
```

- En el DELETE establecemos la ruta con el id del user y buscamos y borramos el user según el id introducido en la ruta.
- Para el PUT buscamos el user por el id y lo updateamos según el id introducido.

Soy consciente de que el ejercicio no está perfecto ya que no se pueden realizar las acciones, pero creo que soy capaz de demostrar que se hacerlos.

Tampoco he sido capaz de implementar las cookies, ya que no tenía sentido con esta API. Quizás utilizando la de Spotify podría haber hecho un filtrado según el artista o el género musical.

Sería algo así:

```
app.get('/cancion_por_genero', async (req, res) => {
    try {
        const filtroGenero = req.query.genre || req.cookies.ultimoGenero || null
        const filtro = filtroGenero ? {status: filtroGenero} : {}
        if (filtroGenero) {
            res.cookie('ultimoGenero', filtroGenero)
        }
        const canciones = await Cancion.find(filtro)
        res.send(canciones)
    } catch {
        res.status(500).send({error: error.message})
    }
})
```

Se podría implementar de manera parecida a esta, al menos haciéndolo sin una API externa. Quizás se haría de manera diferente con una API externa, pero al no poder probarlo no lo sé a ciencia cierta.

Mi objetivo habría sido implementar la API de Spotify y haber sido capaz de realizar todas las consultas e implementar las cookies, pero no he sido capaz de entender cómo podría implementarse esa API, aunque si la de placeholder, quizás porque es más complicado obtener la ruta que se necesita.

Como dije antes, soy consciente de que el ejercicio no está perfecto, pero creo que he sido capaz de defender mis conocimientos lo mejor posible, y me gustaría aprender a hacerlo como debe ser.