Desarrollo Backend con NodeJS

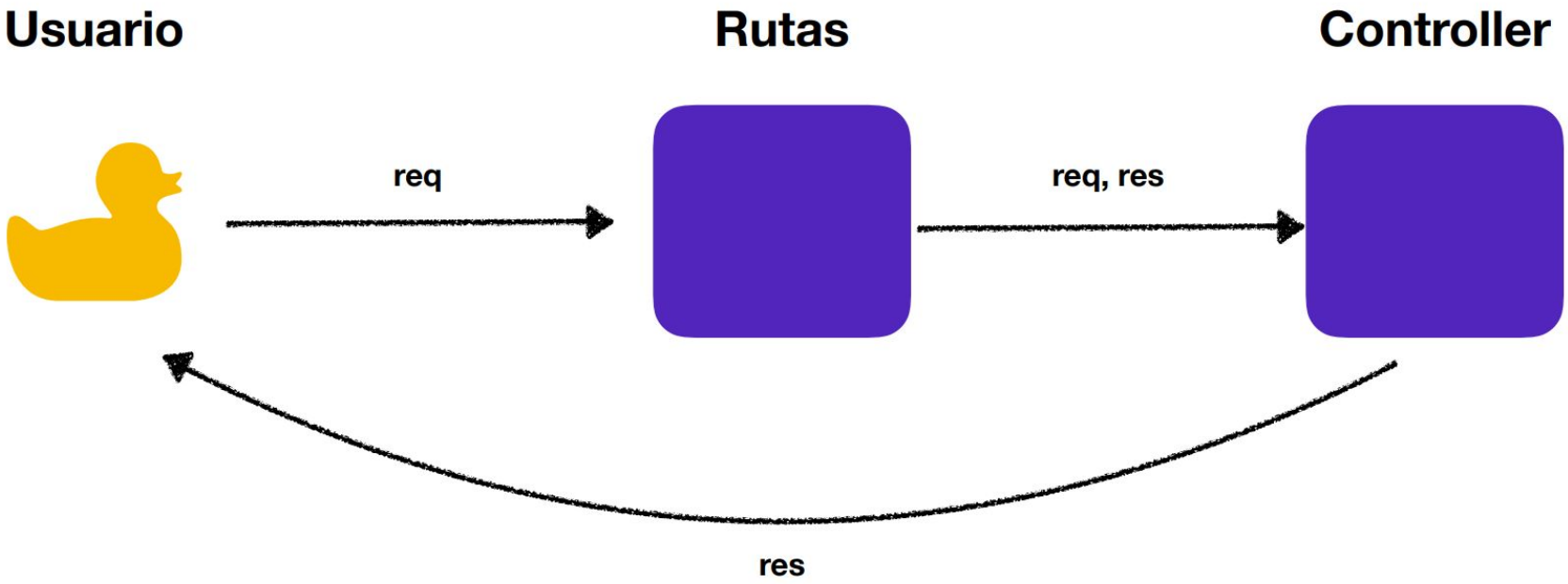
Async / await / Axios y spread operator

course[It]

Haciendo requests desde nuestra API como CLIENTE

Las APIs pueden comunicarse entre sí y consumirse una a la otra sin problemas. **Para ello utilizan el mismo protocolo de comunicación HTTP**.

La diferencia es que en vez de **definir endpoints para ser consumidos**, vamos a estar consumiendo **endpoints externos**



Axios

Axios es una librería para hacer requests HTTP

- https://www.npmjs.com/package/axios
- npm install axios

Importante

Axios no es ni por cerca la única manera de hacer requests como cliente. De hecho Javascript tiene su propia herramienta nativa: **fetch**

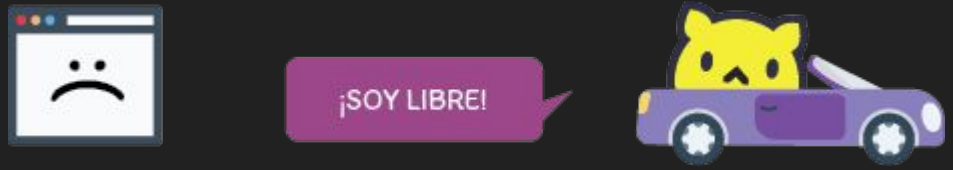
https://developer.mozilla.org/es/docs/Web/API/Fetch_API

Utilizando Axios

```
const axios = require("axios");

class UserController {
  getUser(req, res) {
    const data = axios.get("https://api.github.com/users/doomling");
    console.log(data);

    // adivinen que va a tirar este console.log()

    res.send('mando cualquier cosa para no colgar la request')
  }
}

module.exports = UserController;
```

¿Qué es Node?

- ➔ Node.js es un entorno de ejecución de Javascript
- ➔ Nos permite correr Javascript en el servidor
- ➔ Utiliza el mismo motor que Google Chrome
- ➔ Single-thread
- ➔ Asíncrono / no bloqueante
- ➔ Nos regala algunas funcionalidades nativas como
  - ◆ Acceso al sistema de archivos
  - ◆ Requests HTTP



Código asíncrono en Javascript

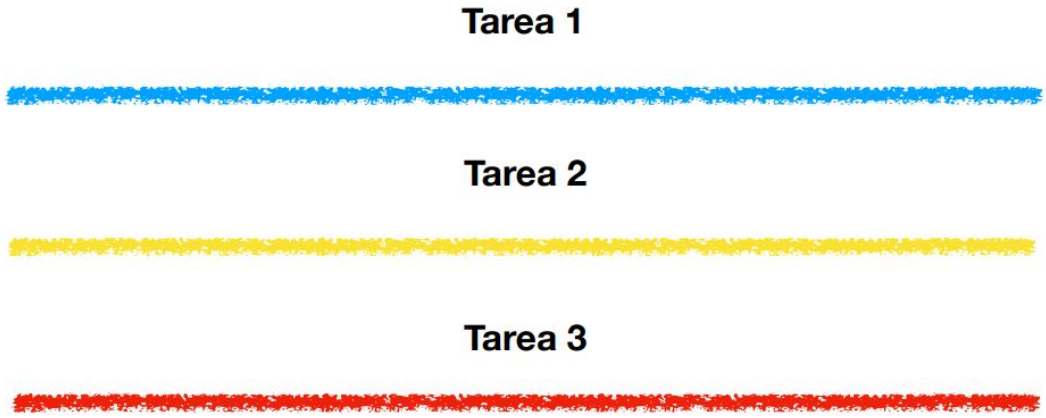
Promesas
Async / await

Async / Await

Async/await es una forma de manejar código asincrónico dentro de Javascipt y fue introducida junto con muchos cambios en ES6

Código sincronico

Código asincrónico



Utilizando Async / Await

```
const axios = require("axios");

class UserController {

  async getUser (req, res) {
    const data = await axios.get("https://api.github.com/users/doomling");
    console.log(data);

    res.json(data.data)
  }
}

module.exports = UserController;
```

Spread operator ...

El spread operator o spread syntax es una funcionalidad que se introdujo a Javascript y permite expandir estructuras de datos iterables

Hablemos de inmutabilidad en Javascript

```
const a = [1, 2, 3];
const b = a;

b.push(5);

console.log(a, "es a");
console.log(b, "es b");
```

Utilizando el spread operator

```
const a = [1, 2, 3];
const b = [...a, 5];

console.log(a, "es a");
console.log(b, "es b");
```

Agregando información utilizando el spread operator ...

```
const axios = require("axios");

class UserController {
  async getUser (req, res) {
    const data = await axios.get("https://api.github.com/users/doomling");
    console.log(data);

    const newData = {...data.data, mascotas: ['Kerrigan', 'Alyx']}

    res.json(newData)
  }
}

module.exports = UserController;
```

Modelando datos

Modelar es simplemente la acción de crear un objeto a mano ‘mapeando’ la información según nuestras necesidades. De esta manera podemos filtrar solo lo que efectivamente queremos devolver.

Modelando datos

```
const axios = require("axios");

class UserController {
  async getUser(req, res) {
    const data = await axios.get("https://api.github.com/users/doomling");
    console.log(data);

    const modeledData = {
      nombre: data.data.name,
      empresa: data.data.company,
    };

    res.json(modeledData);
  }
}

module.exports = UserController;
```