PROYECTO 9: USO DE PROMESAS

TIEMPO ESTIMADO: 150 MINUTOS

DESCRIPCIÓN

- *Realizar consultas a servicios API realizando un conversor en tiempo real de divisas*:

OBJETIVOS

- Entender el concepto de asincronismo
- Saber usar promesas y sus ventajas frente a las funciones de callback
- Uso de Async / Await

PRIMEROS PASOS

- Fork del repositorio en GitHub
- Clonar tu repositorio
- Inicializar proyecto con npm init
- Instalar y configurar eslint extendiendo de standard:

```
npm i -D eslint@5.4.0
node_modules/.bin/eslint --init
```

- Personalizar eslint utilizando el fichero *.eslintrc.json*

EJEMPLO BÁSICO

SUMA ASÍNCRONA DE NÚMEROS

- Modifica script sumaFiles.js para que:
  - Haga lo mismo pero de forma asíncrona
  - Utiliza funciones de callback

```
const fs = require('fs')
const numero1 = fs.readFileSync('./numero1', 'utf-8')
const numero2 = fs.readFileSync('./numero2', 'utf-8')
console.log(`El resultado de la suma es  ${parseInt(numero1)+p
```

SUMA ASÍNCRONA DE NÚMEROS CON FUNCIONES DE CALLBACK

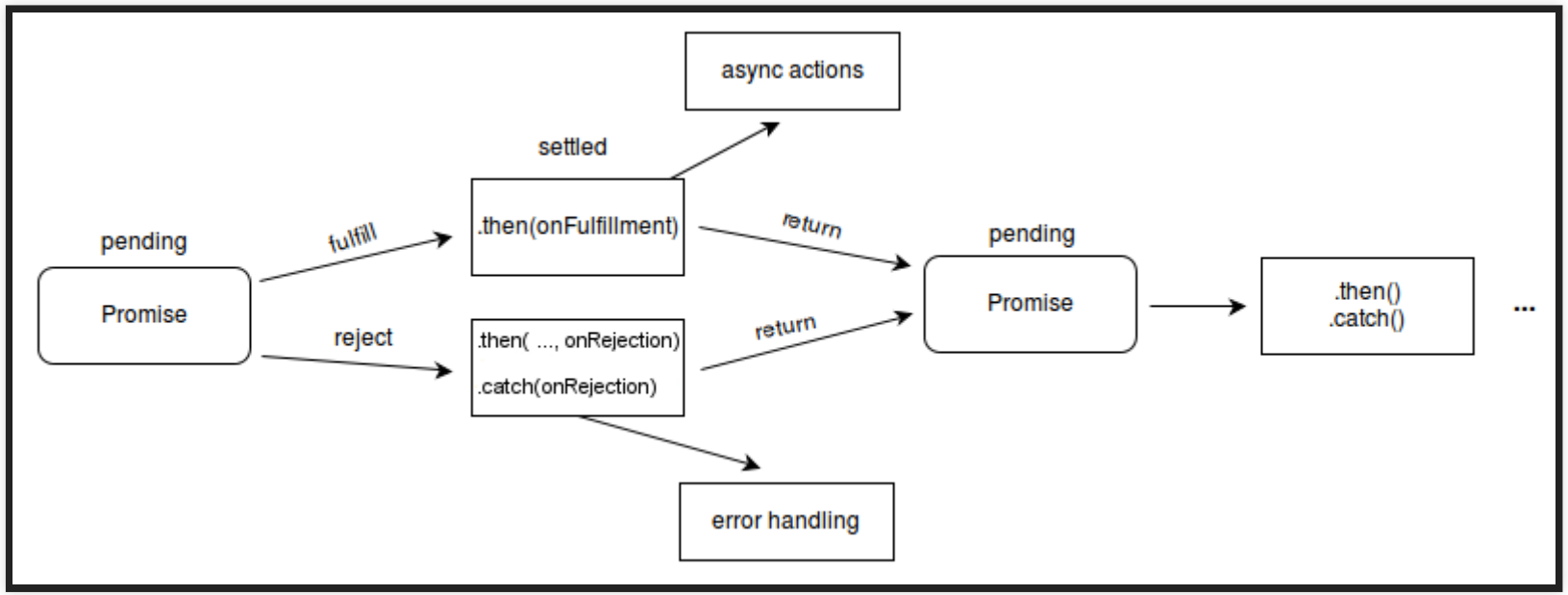
```
const fs = require('fs')
fs.readFile('./numero1', 'utf-8', (err, numero1) => {
  if (err) throw err
  fs.readFile('./numero2', 'utf-8', (err, numero2) => {
    if (err) throw err
    console.log(`El resultado de la suma es  ${parseInt(numero
  })
})
```

DESVENTAJAS FUNCIONES DE CALLBACK

- ¿Y si hubiera que leer 5 ficheros?
  - Excesiva anidación (**callback hell**)
  - Mayor dificultad de desarrollo (peor legibilidad)
    - Lo **ideal** sería utilizar **código secuencial y asíncrono**
  - Se trata de escribir código asíncrono con un estilo síncrono.
  - Podemos tener **muchas llamadas asíncronas** (accesos bbdd, llamadas api....)

- Mayor thoughput **si se leen** los dos ficheros a la vez
- Podríamos incluso hacer otra cosa mientras se leen

PROMESAS AL RESCATE



```
function getData(fileName, type) {
  return new Promise(function(resolve, reject){
    fs.readFile(fileName, type, (err, data) => {
        err ? reject(err) : resolve(data);
    })
  })
}
```

SUMA ASÍNCRONA CON PROMESAS

```
const fs = require('fs')

var numero1

const getData = (fileName, type) =>
  new Promise((resolve, reject) => {
    fs.readFile(fileName, type, (err, data) => {
      err ? reject(err) : resolve(parsetInt(data))
    })
  })

getData('numero1', 'utf-8')
  .then(fileContent => {
    numero1 = fileContent
    return getData('numero2')
```

INTENTO FALLIDO DE SUMA ASÍNCRONA CON PROMESAS

```
const fs = require('fs')
var numero1
var numero2
fs.readFile('./numero1', 'utf-8', (err, numero) => {
  (err) ? console.log(err) : numero1 = numero
})
fs.readFile('./numero2', 'utf-8', (err, numero) => {
  err ? console.log(err) : (numero2 = numero)
})
while (!numero1 && !numero2){}
console.log(`El resultado de la suma es  ${numero1 + numero2}`

// las promesas no retornan nunca: Node.js event loop
// nuestro programa se queda colgado :-(
```

SUMA ASÍNCRONA CON PROMISE.ALL

```
const fs = require('fs')

const getData = (fileName, type) => new Promise(
  (resolve, reject) => {
    fs.readFile(fileName, type, (err, data) => {
        err ? reject(err) : resolve(parseInt(data))
    })
  }
)

var promise1 = getData('numero1', 'utf-8')
var promise2 = getData('numero2', 'utf-8')
Promise.all([promise1, promise2]).
then((arrayValues) => {
  let sum = arrayValues.reduce((sum, x) => sum +x)
```

OTRAS OPCIONES EN PROMESAS

- Usar el módulo fs-extra
- Que un módulo soporte las promesas es un punto a favor
  - fs-extra vs fs
  - request vs axios

- Usar algún tipo de promisify

```
var Promise = require("bluebird")
var fs = require("fs")
Promise.promisifyAll(fs)
fs.readFileAsync("file.js", "utf8").then(...)
```

- Uso de Async-Await (nuestro principal objetivo)

CREACIÓN DE PROMESAS

DESCRIPCIÓN

- Vamos a trabajar con objetos json
  - Crearemos nosotros las promesas
  - Es habitual usar funciones de terceros que nos devuelvan promesas:
    - En la petición API
    - En la petición a la bbdd
- Convertiremos código a async-await

OBTENCIÓN DE DATOS DE OPOSITOR

- Utilizaremos fichero *promises.js* del proyecto actual.
- Creamos una promesa que obtenga los datos de un opositor a partir de un id

```
const getOpositor = (id) => {
  return new Promise((resolve, reject) => {
    const opositor = opositores.find((opositor) => opositor.id
    if (opositor) {
      resolve(opositor)
    } else {
      reject(new Error(`No se ha encontrado al opositor con id
    }
  })
}
```

- Comprobamos que funcione añadiendo el siguiente código:

```
getOpositor(1).then((opositor) => {
  console.log(opositor)
}).catch((e) => {
  console.log(e)
})
```

- Ejecutamos:

```
node promises
```

OBTENCIÓN NOTAS OPOSITOR

- Crea la función *getNotas* que obtenga las notas de un opositor
- La función debe recibir un parámetro (id del opositor)

IMPLEMENTACIÓN FUNCIÓN GETNOTAS

```
const getNotas = (id) => {
  return new Promise((resolve, reject) => {
    const notasOpositor = notas.filter((nota) => nota.id === i
    if (notasOpositor.length) {
      resolve(notasOpositor)
    } else {
      reject(new Error(`No se ha encontrado notas del opositor
    }
  })
}
```

OBTENCIÓN DE LOS DATOS DE UN OPOSITOR Y SUS NOTAS

- Crea una función *getResultado* que:
  - Muestre un texto del tipo:

```
Pepe tiene una media de 5 en la oposición de Informática
```

  - Debes utilizar las funciones getOpositor y getNotas definidas anteriormente.
  - Debes utilizar las funciones map y reduce para tratar los datos.

IMPLEMENTACIÓN FUNCIÓN GETRESULTADO

```
const getResultado = (id) => {
  let opositor
  return getOpositor(id).then((data) => {
    opositor = data
    return getNotas(opositor.id)
  }).then((notas) => {
    let media = 0
    if (notas.length > 0) {
      media = notas.map((nota) => nota.nota).reduce((a, b) =>
    }
    return `${opositor.nombre} tiene una media de ${media} en
  })
}
```

ASYNC - AWAIT

- Vamos a implementar la función getResultado mediante async-await
- Se leerá mejor:
  - Evitamos el encadenamiento de las promesas
  - El código se ve más síncrono (secuencial)

SINTAXIS ASYNC AWAIT

- Utilizamos las etiquetas *async* y *await*
- La función se etiqueta como *async*
  - Requerimiento para utilizar await en su cuerpo
  - La función **siempre devuelve una promesa***

```
const getResultado = async (id) => {
  await sentence 1
  sentence 2
  await sentence 3
  return <promise>
}
```

RETORNO FUNCIÓN ASYNC

- Comprueba la salida de esta función:

```
const getResultado = async () => {
  return 'Resultado'
}
```

- Sería equivalente a:

```
const getResultado = async () => {
return new Promise((resolve, reject) =
  resolve('Resultado')
})
}
```

- Y se recogerá mediante

```
getResultado().then((data)=>...)
```

REJECT EN FUNCIÓN ASYNC

```
const promiseFunction = () => {
  new Promise((resolve, reject) => {
    reject(new Error('Error al ejecutar promesa'))
  })
}
```

```
const AsyncFunction = () => {
  throw new Error('Error al ejecutar promesa')
}
```

- Y se recogerá mediante

```
getResultado().then((data) =>...).catch((e) =>...)
```

- En funciones async:
  - *return = resolve*
  - *throw = reject*

AWAIT

- Si cambiamos nuestra función getResultado:

```
const getResultado = async (id) => {
  const opositor = await getOpositor(id)
  console.log(notas)
}
```

- await esperará el resolve de la promesa para guardarlo en la variable
- Si quitamos await, recogeremos una promesa y no su resultado

CONSUMIR APIS

OBJETIVO

- Implementar una función *covertCurrency* que:
  - Haga conversiones entre divisas:
  - Muestre los paises que utilizan la divisa de destino
- Parámetros de entrada:

```
<código moneda origen>, por ej. EUR
<código moneda destino>, por ej. USD
<cantidad a cambiar>, por ej. 100
```

- Salida requerida, por ej:

```
Vendiendo 100 EUR obtienes 115 USD. Los puedes utilizar en los
```

SERVICIOS DE API'S

- Fixer
  - Para obtener los tipos de cambio
  - Es necesario autenticarse
- La cuenta gratuita solo permite EUR como moneda base:
  - Si nuestra moneda base no es EUR, tendremos que implementar la conversión

- Necesitaremos crear una función del tipo

```
const exchangeRate = (from, to) => {
  // consulta a la API de Fixer
}
```

- Rest Countries
  - Nos interesará para obtener los paises que utilizan una determinada moneda
  - No es necesario autenticarse
  - Hay un endpoint específico para monedas
- Necesitaremos crear una función del tipo

```
const getCountries = (currencyCode) => {
  // consulta a la API de Rest Countries, endpoint de divisas
}
```

LIBRERÍAS PARA HTTP REQUESTS

- Instalaremos axios (usa promesas)

```
npm i -S axios
```

- Nos ofrece directamente los datos parseados (no es necesario *JSON.parse*)

IMPLEMENTACIÓN FUNCIÓN GETEXCHANGERATE

```
const getExchangeRate = (from, to) => {
  return axios.get('http://data.fixer.io/api/latest?access_key
    const euro = 1 / response.data.rates[from]
    const rate = euro * response.data.rates[to]
    return rate
  })
}

getExchangeRate('USD', 'CAD').then((rate) => {
  console.log(rate)
})
```

IMPLEMENTACIÓN FUNCIÓN GETEXCHANGERATE CON ASYNC-AWAIT

```
const getExchangeRate = async (from, to) => {
  const response = await axios.get('http://data.fixer.io/api/l
  const euro = 1 / response.data.rates[from]
  const rate = euro * response.data.rates[to]
  return rate
}

getExchangeRate('USD', 'CAD').then((rate) => {
  console.log(rate)
})
```

IMPLEMENTACIÓN FUNCIÓN GETCOUNTRIES

```
const getCountries = async (currencyCode) => {
  return axios.get(`https://restcountries.eu/rest/v2/currency/
)}

getCountries('CAD').then((countries) => {
  console.log(countries)
})
```

- Intenta ahora implementar getCountries con async-await

IMPLEMENTACIÓN FUNCIÓN GETCOUNTRIES CON ASYNC-AWAIT

```
const getCountries = async (currencyCode) => {
  const response = await axios.get(`https://restcountries.eu/r
  return response.data.map((country) => country.name)
}

getCountries('CAD').then((countries) => {
  console.log(countries)
})
```

IMPLEMENTACIÓN FUNCIÓN CONVERTCURRENCY

- Utilizaremos las funciones *getExchangeRates* y *getCountries*

```
const convertCurrency = (from, to, amount) => {
  getExchangedRate(from, to).then((rate)=>{
    const = convertedAmount = (amount)*rate.toFixed(2)
    return getCountries(to)
  }).then ((countries))=>{
    console.log(countries)
    // return `Vendiendo ${amount} ${from} obtienes ${converte
  }
}

convertCurrency('USD', 'USD', 20).then((message) => {
  console.log(message);
})
```

PROBLEMA DE SCOPE

- Para devolver el mensaje hay variables que están fuera del scope
- Ya nos pasó al hacer la suma de números de ficheros... ¿sabrías solucionarlo?

PROPUESTA FINAL

- Implementar *convertCurrency* mediante async-await

SOLUCIÓN CONVERTCURRENCY

```
const convertCurrency = async (from, to, amount) => {
  const rate = await getExchangeRate(from, to)
  const countries = await getCountries(to)
  const convertedAmount = (amount * rate).toFixed(2)
  return `Vendiendo ${amount} ${from} obtienes ${convertedAmou
}

convertCurrency('USD', 'USD', 20).then((message) => {
  console.log(message)
})
```

SOLUCIÓN COMPLETA CON GESTIÓN DE ERRORES

```
const axios = require('axios')

const getExchangeRate = async (from, to) => {
  try {
    const response = await axios.get('http://data.fixer.io/api
    const euro = 1 / response.data.rates[from]
    const rate = euro * response.data.rates[to]

    if (isNaN(rate)) {
      throw new Error()
    }

    return rate
  } catch (e) {
    throw new Error(`Unable to get exchange rate for ${from} a
```