```
Bens-iMac:unhandled-reject ben$
Bens-iMac:unhandled-reject ben$ node test.js
[PROCESS] Unhandled Promise Rejection
- - - - - - - - - - - - - - - - - - - - -
Error: Something went wrong.
    at /Users/ben/testing-nodejs/unhandled-reject/test.js:32:
    at process._tickCallback (internal/process/next_tick.js:1
    at Module.runMain (module.js:609:11)
    at run (bootstrap_node.js:420:7)
    at startup (bootstrap_node.js:139:9)
    at bootstrap_node.js:535:3
- -
Bens-iMac:unhandled-reject ben$
Bens-iMac:unhandled-reject ben$
```

Tarea 6: Logging, Testing

3 de Abril

En esta tarea le pondremos un logger a la aplicación, y testing End to End.

Logging

El registro de eventos es básico en las aplicaciones web, en la fase de desarrollo, y mucho más en la de producción. Un buen sistema de logging facilita la depuración de errores, la seguridad, el control de errores,incluso puede ser obligatorio por normativa.

Usaremos winston, siguiendo A Complete Guide to Winston Logging in Node.js. Usarlo es muy simple:

```
logger.debug(`La variable x: ${x}`)  // mensaje al canal de debug
logger.info('Info message');         // mensaje al canal de info
logger.error('Error message');       // mensaje al canal de error
```

La única dificultad es configurarlo para la salida en los distintos canales y para el formato de los mensajes. Una configuración para empezar:

```
// logger.mjs
import winston from "winston"
```

```
const { combine, timestamp, printf, align} = winston.format

const logger = winston.createLogger({

  level: process.env.LOG_LEVEL || 'debug',
  format: combine(
    timestamp({
      format: 'YYYY-MM-DD hh:mm:ss',
    }),
    align(),
    printf(info => `[${info.timestamp}] ${info.level.toUpperCase()}: ${info.message}`)
  ),
  transports: [
    new winston.transports.Console(),        // siempre a consola

    new winston.transports.File({
      filename: 'app.log',                   // salida a partir de info
      level: 'info',
    }),

    new winston.transports.File({
      filename: 'error.log',                // salida para error
      level: 'error',
    }),
  ],
});

export default logger
```

A partir de aquí se pueden usar plugins para usar logs rotatorios, mandar a un agregador, a telegram, etc.

Testing

Vamos a usar Playwright para las pruebas. Seguimos Playwright Testing Essentials: A Beginner's Guide Primero hay que desinstalar la versión que se instaló para el scrapping, porque da problemas con la versión más completa para testing.

```
> pnpm rm playwright
> pnpm create playwright@latest
```

y pasar ahora el test que viene de prueba:

```
> pnpm exec playwright test
> pnpm exec playwright test --UI
> pnpm exec playwright show-report
```

o con la extensión de vscode.

Un test podría ser:

```
// museo-spec.js
import { test, expect } from '@playwright/test';

test('Título de la página', async ({ page }) => {
  await page.goto('http://localhost:8000/');
  await expect(page).toHaveTitle(/SSBW/);
});

test('Busco oro', async ({ page }) => {
  await page.goto('http://localhost:8000/obras/buscar?b%C3%BAsqueda=oro');
  await expect(page.getByRole('heading', { name: 'Reproducción de diadema' })).toBeVis
});

test('Logging pp', async ({ page }) => {
  await page.goto('http://localhost:8000/usuarios/login');

  await page.fill("#email", "pp@pp.com");
  await page.fill("#password", "pp");
  await page.click("#login");
  await expect(page.getByRole('heading', { name: 'Obras destacadas' })).toBeVisible();
});
```

Referencias:

- Winston: A Better Way To Log
- Writing tests
- Playwright Guide: Submitting Forms
- API > Classes > Locator