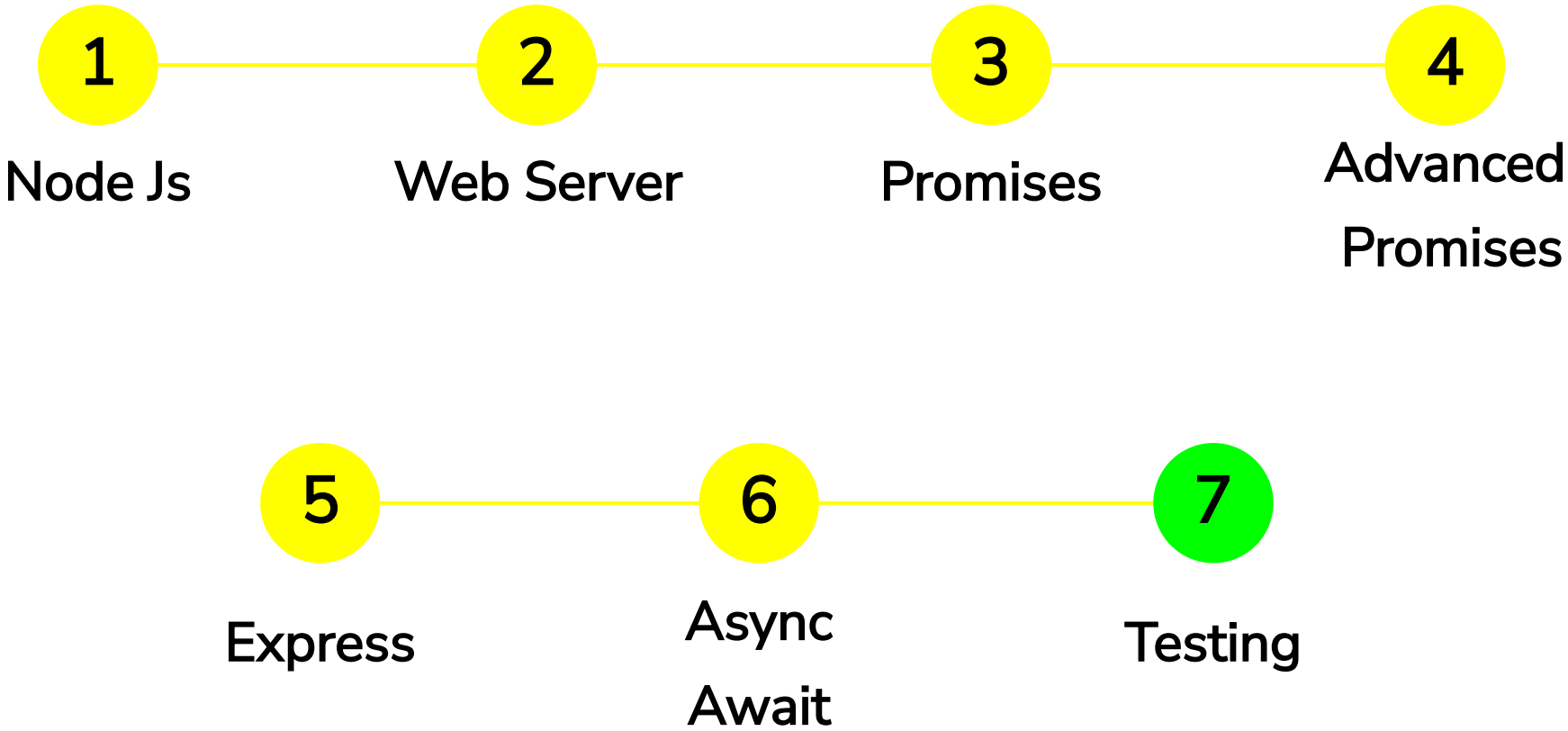

HENRY

Lecture

Testing

Contenido M3



Recapitulemos...

1. Generator Functions -> *Función especial que nos permite tener control en la ejecución de la función, mediante un objeto Generator .*
2. next() -> *Método del objeto Generator que muestra el siguiente valor en un generador.*
3. yield -> *Operador que pausa la función generadora.*
4. return -> Finaliza la ejecución mostrando su valor.
5. throw -> *Finaliza la ejecución generando un error.*

Recapitulemos...

6

Estados del objeto Generator ->

- *suspended -> la función generadora se detiene pero el proceso no finaliza.*
- *closed -> La función generadora termina por un return, por la iteración de todos los valores o surge un error.*

7

async/await *-> Sintaxis que simplifica la programación asincrónica. Con la palabra async definimos una función asíncrona y await espera y recibe la resolución de la promesa, si la promesa se resuelve recibe su valor, si se rechaza recibe la razón del rechazo.*

Recapitulemos...

8

try/catch -> *Bloque de instrucciones para el manejo de errores, donde try ejecuta el código y catch detecta los errores.*

OBJETIVO DE LA CLASE

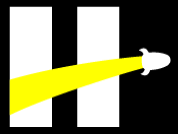
Recordar los temas vistos en el módulo 1 de testing y profundizar conceptos para realizar testing a nuestro servidor y nuestras rutas con el fin de tener nuestro código más eficiente.

¿Qué veremos hoy?

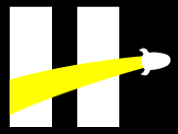
- Testing Frameworks
- Jest
- Código asíncrono
- Hooks
- Mock Functions
- Supertest

Espacio de interacción

Recuerden hacer las preguntas en el **Q&A** con contexto para que no se pierdan en el chat.



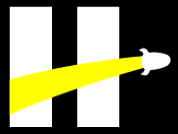
Testing



Testing Frameworks

VS

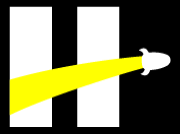
- Assertion Library included
- Needs other libraries (assertion, mocking)
- More flexibility



Jest

Configuration

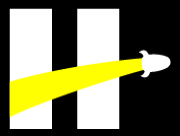
```
1  // Instalation
2  npm install --save-dev jest
3
4  // package.json configuration
5  ...
6
7    "scripts": {
8      "test": "jest"
9    }
10
11 ...
```



Jest

CLI Options

```
// Run only tests matching
jest test-pattern    // pattern
jest path/to/test.js // filename
jest -t name-spec    // describe or test name

// Run in watch mode
jest --watch // by dafault only changed files
jest --watchAll

// Show summary of each test file
// Running one file only --> automatically verbose
jest --verbose
```

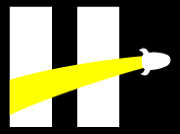


Jest

First Example

```
// sum.js

function sum(a, b) {
  return a + b;
}

module.exports = sum;
```

```
// sum.test.js

const sum = require('./sum');

// it === test
it('should return 8 if adding 3 and 5', () => {
  expect(sum(3, 5)).toBe(8);
});
```



Jest

Matchers

expect devuelve un "expectation" object sobre el cual se pueden invocar los matchers

- **toBe**: igualdad exacta
- **toEqual**: verificación recursiva de cada propiedad del objeto o elemento del arreglo
- **toBeNull**, **toBeUndefined**, **toBeDefined**
- **toBeTruthy**: verifica que el valor de veracidad sea verdadero sin necesariamente ser literalmente **true**
- **toBeFalsy**: verifica que el valor de veracidad sea falso sin necesariamente ser literalmente **false**

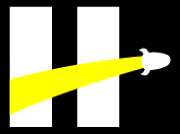
Jest

Matchers

- **toBeGreaterThan**, **toBeGreaterThanOrEqual**, **toBeLessThan**, **toBeLessTahnOrEqaul** para números
- **toBeCloseTo** para números con decimales
- **toMatch**: compara contra una expresión regular
- **toContain**: verifica si dentro de un arreglo existe un elemento
- **toThrow**: verifica si la función arroja un error

Y más ...

<matchers.test.js />



Jest

Running Options

- **xit**: evita que ese test en particular se ejecute
- **describe**: permite agrupar varios tests dentro de una misma temática o categoría (acepta mas de un nivel de anidación)
- **xdescribe**: evita que todos los tests de ese grupo se ejecuten
- **it.only**: hace que ese test sea el único en ejecutarse

Asynchronous Code

Resolved Promises

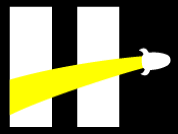
```
1 it('should resolve to Henry Promise', () => {
2   promisifiedFunction(false).then(data => {
3     expect(data).toBe('asd');
4   });
5 });
```


```
PASS ./promise.test.js
  ✓ should resolve to Henry Promise (1 ms)
```

```
1 it('should resolve to Henry Promise', () => {
2   return promisifiedFunction(true).then(data => {
3     expect(data).toBe('Henry Promise');
4   });
5 });
```

Si se omite el return el test va a completarse antes de que la promesa se complete

<promises.test.js />



Asynchronous Code

Rejected Promises

Jest

```
1  it('should reject to Rejected Promise', () => {
2    return promisifiedFunction(true).catch(e => {
3      expect(e).toMatch('Rejected Promise')
4    });
5  });
```

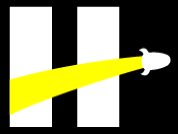

```
PASS ./promise.test.js
  ✓ should reject to Rejected Promise (2004 ms)
```

```
1  it('should reject to Rejected Promise', () => {
2    expect.assertions(1);
3    return promisifiedFunction(false).catch(e => {
4      expect(e).toMatch('Rejected Promise')
5    });
6  });
```

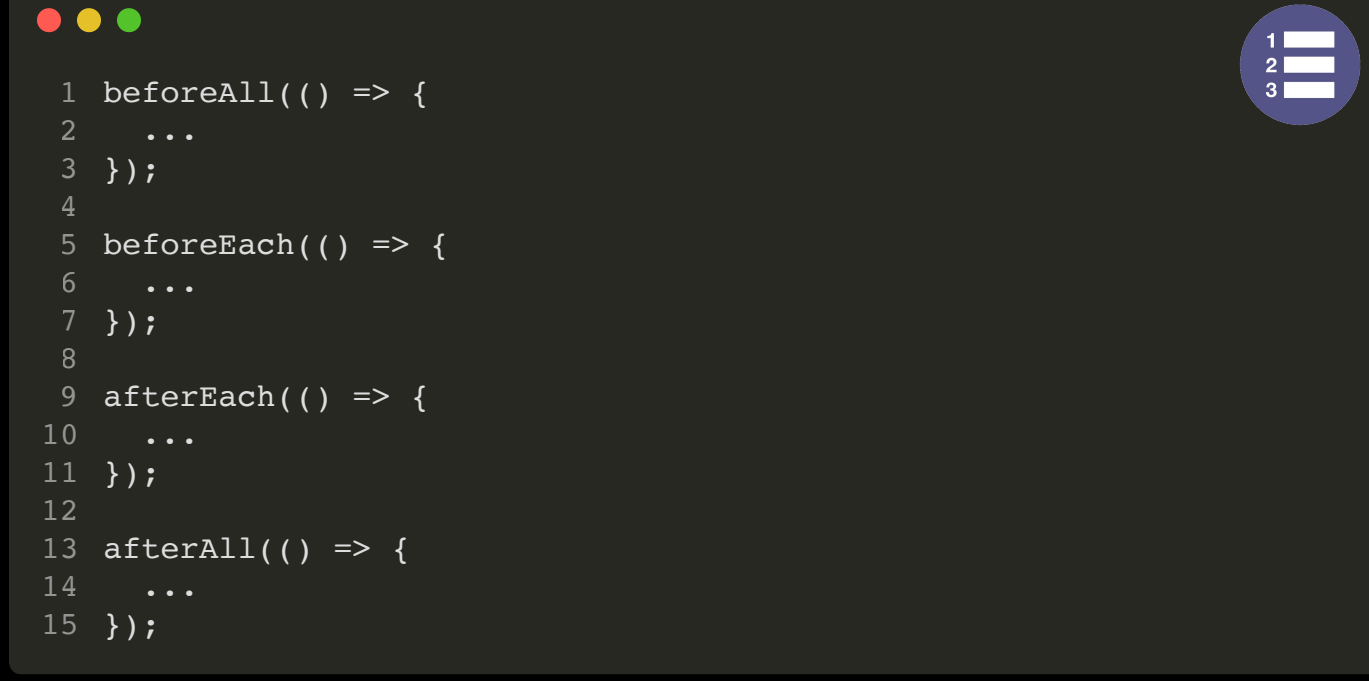
+ sobre assertions

Si se omite el assertions una promesa cumplida no hará que el test falle

await <promise.test.js />

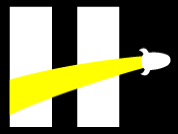


Hooks



```
1  beforeAll(() => {
2    ...
3  });
4
5  beforeEach(() => {
6    ...
7  });
8
9  afterEach(() => {
10   ...
11 });
12
13 afterAll(() => {
14   ...
15 });
```


<hooks.test.js />



Mock Functions

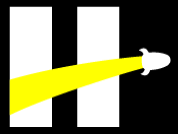
a.k.a "spies"

Permite testear el comportamiento de una función que indirectamente fue ejecutada por otra

```
1 const mockFunction = jest.fn(person => person.age > 18);
```

- **.mock.calls**: array con todas las invocaciones a la función donde cada elemento contiene otro array con los argumentos pasados
- **.mock.results**: array con todas los resultados devueltos por la función donde cada elemento contiene un objeto con el valory el tipo de retorno

<mock.test.js />



Supertest

Permite testear los request a nuestro servidor de forma autocontenida sin necesdad de levantar nuestra app

- **statusCode**: podemos verificar si el código de respuesta es el adecuado
- **response**: podemos verificar si la respuesta del endpoint coincide con lo esperado (Puede ser por text o body)
- **type**: podemos verificar si el content type devuelto es el correcto

Si el **.listen** de express se encuentra en el archivo requerido en el testing va a generar que el test no termine de ejecutar nunca

Resumen

- **Jest:** Recordamos cómo instalar y configurar este framework que nos ayuda a realizar nuestros test. npm i jest

- **Matchers:** Antes debemos usar el expect quien devuelve un objeto con el que podemos invocar los métodos matchers. Algunos matchers:
  - toBe: Busca la igualdad exacta.
  - toEqual: Verifica de forma recursiva cada propiedad de un objeto o cada elemento de un array.
  - toBeTruthy: Verifica que el valor sea verdadero, puede ser un string, un número, arreglo, booleano, etc.
  - toBeFalsy: Es el antónimo de toBeTruthy, verifica que el valor sea falsy.

Resumen

- **x:** Con la "x" ignoramos un test si la anteponemos al it o un grupo de test anteponiéndola a un describe.
- **it.only:** Solo ejecuta ese test.
- **Testing en promesas:** No olvidar el return para que nuestros test estén correctos y esperen la resolución de la promesa.
- **Hooks:** Nos ayudan a ejecutar una función antes o después de que se ejecuten los test. Entre ellos están el beforeAll, beforeEach, afterAll y afterEach.

Resumen

- **funciones Mock:** Nos permite testear el comportamiento de una función que indirectamente fue ejecutada por otra función(por ejemplo, simula un callback).
- **Supertest:** Nos permite testear las request de nuestro servidor de forma autocontenida sin necesidad de levantar el servidor.
  Recuerden que:
  - .listen siempre debe encontrarse en otro archivo para que nuestros test puedan terminar de ejecutarse.
  - Las request son asíncronas, por lo que en nuestros test podemos usar async await para esperar el resultado de las rutas.

¿Preguntas?

Homework

Pair Programming

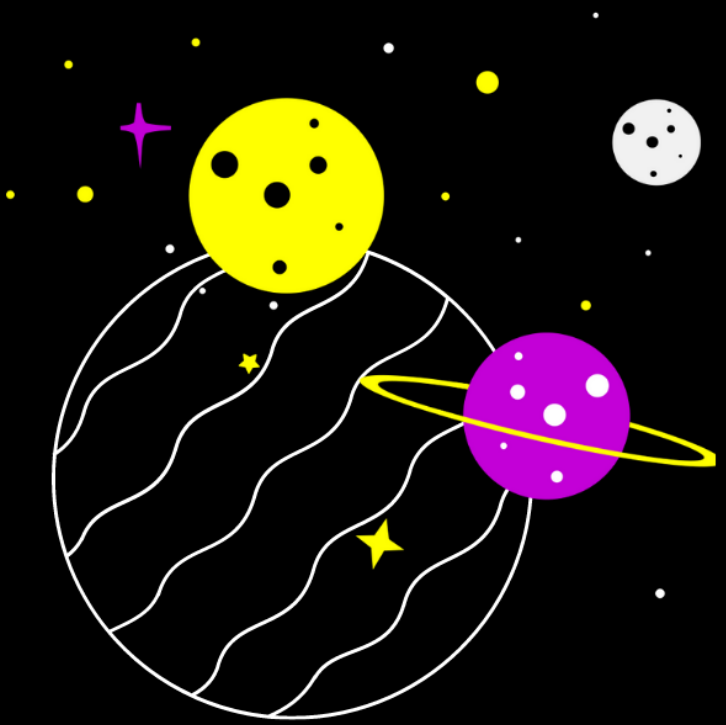
SUP

¿Preguntas?

PROXIMA CLASE

Repaso

HENRY



¡Muchas gracias!