

EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: HACIENDO DEBUGGING CON VISUAL STUDIO CODE.
- EXERCISE 2: TESTING.

EXERCISE 1: HACIENDO DEBUGGING CON VISUAL STUDIO CODE

A partir del siguiente código, el cual calcula la hipotenusa, perímetro, y área de un triángulo:

```
1 const ladoUno = 3;
2 const ladoDos = 3;
3 const ladoTres = 3;
4 let base = 2
5 let altura = 3
6 let divisor = 3
7
8 const calculaArea = (base, altura, peso) => {
9   return (base * peso) / divisor;
10 }
11 const calculaPerimetro = (ladoUno, ladoDos, ladoTres) => {
12   base++;
13   return ladoUno + ladoDos + ladoTres;
14 }
15 const calculaHipotenusa = (catetoUno, catetoDos) => {
16   const catetoUnoCuadrado = catetoUno * catetoUno;
17   const catetoDosCuadrado = catetoDos * catetoDos;
18   return Math.sqrt(catetoUnoCuadrado + catetoDosCuadrado);
19 }
20
21 console.log("");
22 console.log("Valores iniciales:");
23 console.log("");
24 console.log(`Lado 1 es igual a: ${ladoUno}`)
25 console.log(`Lado 2 es igual a: ${ladoDos}`)
26 console.log(`Lado 3 es igual a: ${ladoTres}`)
27 console.log(`Altura es igual a: ${altura}`)
28 console.log(`Base es igual a: ${base}`)
29 console.log("");
30 console.log(`Resultados:`);
31 console.log("");
32 console.log(`La hipotenusa del triangulo es igual a:
33 ${calculaHipotenusa()}`);
34 console.log(`El perimetro del triangulo es igual a:
35 ${calculaHipotenusa()}`);
36 console.log(`El area del triangulo es igual a:
37 ${calculaHipotenusa()}`);
```

El resultado correcto que deberíamos obtener de nuestro programa, es el siguiente:

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\debugging>node programaCorrecto.js

Valores iniciales:

Lado 1 es igual a: 3
Lado 2 es igual a: 3
Altura es igual a: 3
Base es igual a: 2

Resultados:

La hipotenusa del triangulo es igual a: 4.242640687119285
El perimetro del triangulo es igual a: 10.242640687119284
El area del triangulo es igual a: 3

C:\Users\debugging>
```

Pero, el resultado obtenido con el código actual, es:

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\debugging>node programaRoto.js

Valores iniciales:

Lado 1 es igual a: 3
Lado 2 es igual a: 3
Altura es igual a: 3
Base es igual a: 2

Resultados:

La hipotenusa del triangulo es igual a: 4.58257569495584
El perimetro del triangulo es igual a: 10.582575694955839
El area del triangulo es igual a: NaN

C:\Users\debugging>
```

Estas son las fórmulas para calcular la hipotenusa, perímetro, y área:

Hipotenusa: $hipotenusa^2 = Lado\ uno^2 + Lado\ dos^2$

Perímetro: $perimetro = a + b + c$

Área: $area = \frac{base \cdot altura}{2}$

Para poder encontrar los errores en este código, de forma más rápida y eficiente, utilizaremos la herramienta de **debugging** de VS Code

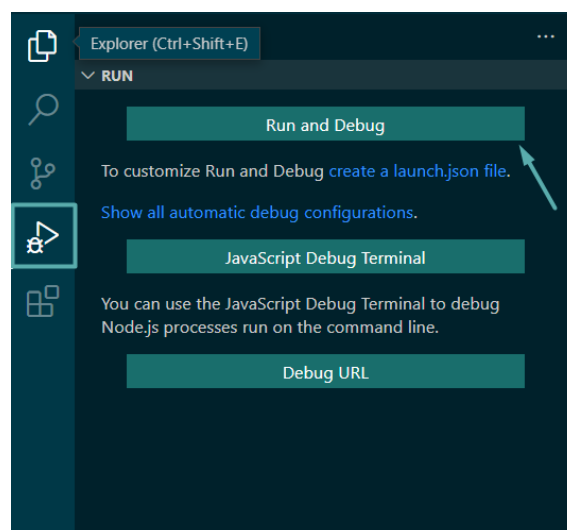
Primero, agregaremos un **breakpoint** en la línea 34, donde el programa invoca la primera función que calcula la hipotenusa.

```

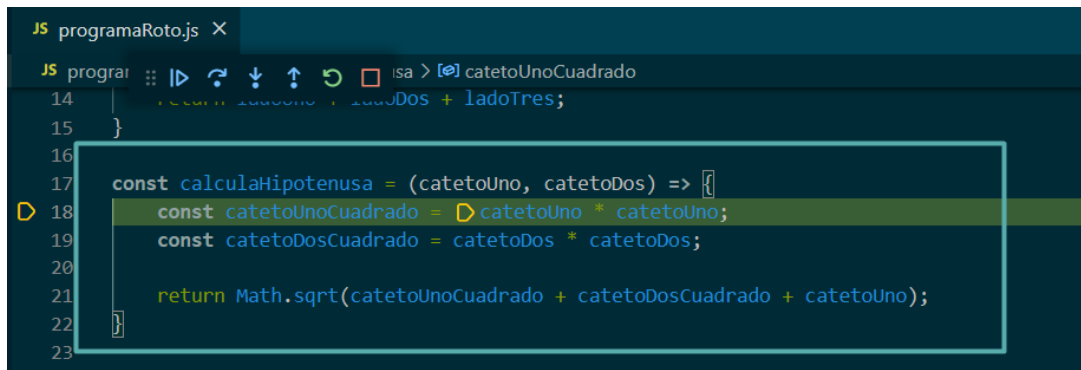
25 console.log("Valores iniciales:");
26 console.log("");
27 console.log(`Lado 1 es igual a: ${ladoUno}`)
28 console.log(`Lado 2 es igual a: ${ladoDos}`)
29 console.log(`Altura es igual a: ${altura}`)
30 console.log(`Base es igual a: ${base}`)
31 console.log("");
32 console.log("Resultados:");
33 console.log("");
34 console.log(`La hipotenusa del triangulo es igual a: ${calculaHipotenusa(ladoUno, ladoDos)}`);
35 console.log(`El perimetro del triangulo es igual a: ${calculaPerimetro(ladoUno, ladoDos, calculaHipotenusa(ladoUno,
36 console.log(`El area del triangulo es igual a: ${calculaArea(base, altura, peso)}`);
37
38

```

Y luego, iniciamos una sesión de depuración.



Una vez iniciada la sesión, presionaremos el botón **“Step into”** para ir dentro de la función, y verificar la razón del resultado erróneo en nuestro programa.



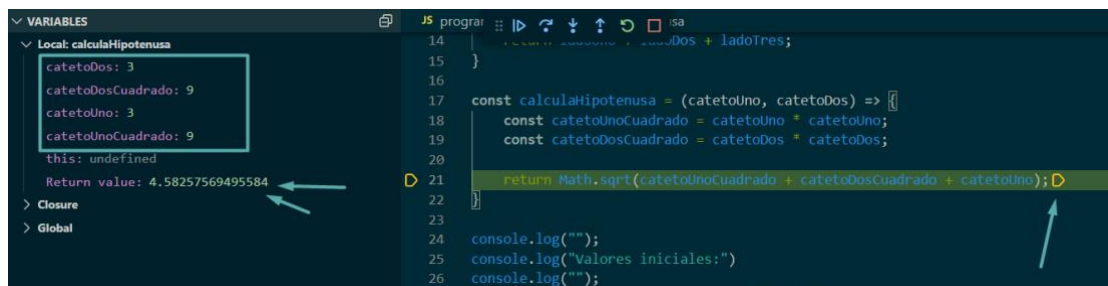
The screenshot shows a code editor with a file named 'programaRoto.js'. The code defines a function 'calculaHipotenusa' that takes 'catetoUno' and 'catetoDos' as arguments. Inside the function, it calculates 'catetoUnoCuadrado' and 'catetoDosCuadrado', and then returns the square root of their sum plus 'catetoUno'. A green box highlights the function definition, and a yellow arrow points to the 'Step into' button in the toolbar.

```

14  {
15  }
16
17  const calculaHipotenusa = (catetoUno, catetoDos) => {
18    const catetoUnoCuadrado = catetoUno * catetoUno;
19    const catetoDosCuadrado = catetoDos * catetoDos;
20
21    return Math.sqrt(catetoUnoCuadrado + catetoDosCuadrado + catetoUno);
22  }
23

```

Estando aquí, podemos observar visualmente la función, o avanzar hasta su punto de retorno con el botón **“Step Over”**.



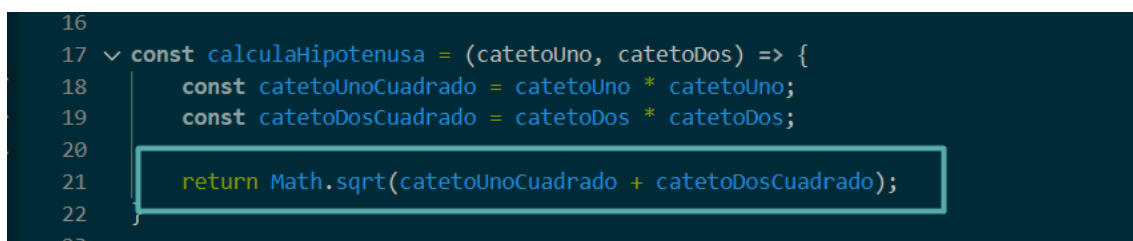
The screenshot shows the same code editor with the 'Step Over' button highlighted in the toolbar. The 'Variables' panel on the left shows the local variables 'catetoDos: 3', 'catetoDosCuadrado: 9', 'catetoUno: 3', and 'catetoUnoCuadrado: 9'. The 'Return value' is shown as '4.58257569495584'. A yellow arrow points to the 'Step Over' button, and another yellow arrow points to the return statement in the code.

```

14  {
15  }
16
17  const calculaHipotenusa = (catetoUno, catetoDos) => {
18    const catetoUnoCuadrado = catetoUno * catetoUno;
19    const catetoDosCuadrado = catetoDos * catetoDos;
20
21    return Math.sqrt(catetoUnoCuadrado + catetoDosCuadrado + catetoUno);
22  }
23
24  console.log("");
25  console.log("Valores iniciales:");
26  console.log("");

```

Al avanzar, podemos observar que el cuadrado de los catetos se encuentra bien calculado, pero en la sentencia **return**, existe un valor en la suma que no corresponde a la fórmula correcta; por lo tanto, eliminaremos la variable "catetoUno" de la suma de la sentencia **return**.



The screenshot shows the corrected code where the 'catetoUno' variable has been removed from the return statement. A green box highlights the corrected return statement.

```

16
17  const calculaHipotenusa = (catetoUno, catetoDos) => {
18    const catetoUnoCuadrado = catetoUno * catetoUno;
19    const catetoDosCuadrado = catetoDos * catetoDos;
20
21    return Math.sqrt(catetoUnoCuadrado + catetoDosCuadrado);
22  }
23

```

Corregimos el código, reiniciamos la sesión con el botón **“Restart”**, y avanzamos hasta el mismo punto para confirmar que nuestra corrección se hizo bien.

Al continuar verificando las líneas de nuestro código, y calcular el perímetro, la herramienta de depuración entrará nuevamente a la función para calcular la hipotenusa, pues ya se está utilizando ese valor para calcular el tercer lado, con el cual podremos obtener el perímetro del triángulo.

Al revisar el detalle dentro de la función para calcular el perímetro, podemos observar que la variable base estaba aumentando su valor en uno, por lo tanto, eliminaremos esta línea de código. Ese es el único error de esta función.

Continuamos al siguiente paso de la sesión de depuración, para revisar el detalle de la función para calcular el área.

```

1  const ladoUno = 3;
2  const ladoDos = 3;
3  let base = 2;
4  let altura = 3;
5  let divisor = 3;
6  let peso;
7
8  const calculaArea = (base, altura, peso) => {
9    return (base * peso) / divisor;
10 }
11

```

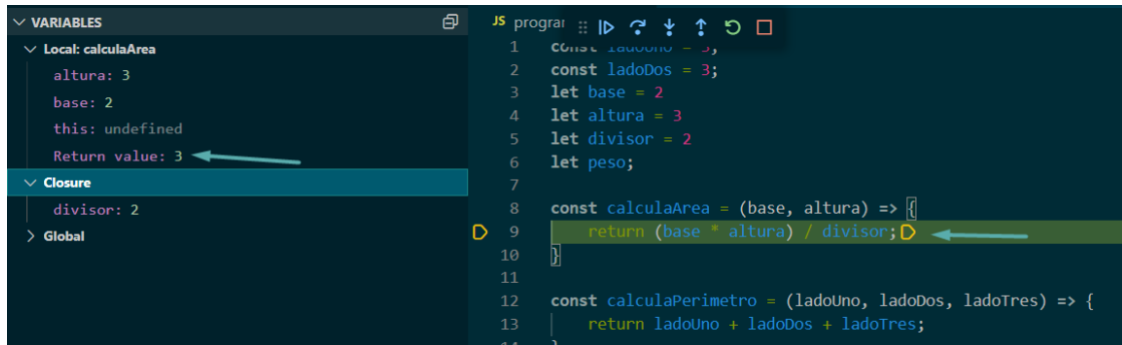
En este punto del código, podemos observar que se está utilizando la variable peso para calcular el área. Por una parte, el peso no tiene ninguna relación con el cálculo del área de un triángulo, y por otra, el valor se encuentra indefinido. Es por ello que modificaremos la función para utilizar la formula correcta, y avanzaremos un paso más para revisar el valor retornado.

```

1  const ladoUno = 3;
2  const ladoDos = 3;
3  let base = 2;
4  let altura = 3;
5  let divisor = 3;
6  let peso;
7
8  const calculaArea = (base, altura) => {
9    return (base * altura) / divisor;
10 }
11

```

Como podemos observar, el valor de retorno aún no es el correcto, y si verificamos el de todas las variables utilizadas en el cuerpo de esta función, en el panel izquierdo, veremos que la variable “divisor” tiene como valor 3, cuando acorde a la fórmula, debería ser 2. Cambiaremos el valor, y lanzaremos el programa una vez más.



Ahora, el resultado final será el correcto.

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\debugging>node programaRoto.js

Valores iniciales:

Lado 1 es igual a: 3
Lado 2 es igual a: 3
Altura es igual a: 3
Base es igual a: 2

Resultados:

La hipotenusa del triangulo es igual a: 4.242640687119285
El perimetro del triangulo es igual a: 10.242640687119284
El area del triangulo es igual a: 3

C:\Users\debugging>
  
```

EXERCISE 2: TESTING

Utilizaremos el código del servidor creado en el ejercicio del CUE – Introducción a **API Rest**, y generaremos las pruebas para cada uno de los verbos HTTP implementados. Recuerda iniciar un proyecto **npm**, configurar tu archivo **package.json** para utilizar mocha, e instalar **mocha**, **chai** y **chai-http**, con el comando: **“npm install mocha chai chai-http --save-dev”**

Archivo index.js.

```
1 const http = require('http');
2 const fs = require('fs/promises');
3
4 const { creaComic } = require('./crea');
5
6 const servidor = http.createServer(async (req, res) => {
7   const { searchParams, pathname } = new URL(req.url,
8     `http://${req.headers.host}`);
9   const params = new URLSearchParams(searchParams);
10
11   if(pathname == '/comics' && req.method == 'GET'){
12     const lecturaArchivo = await fs.readFile('comics.txt');
13     res.write(lecturaArchivo);
14     res.end();
15   }
16
17   if(pathname == '/comics' && req.method == 'POST'){
18     let datosComic;
19
20     req.on('data', (data) => {
21       datosComic = JSON.parse(data);
22     })
23     req.on('end', async () => {
24       await creaComic(datosComic);
25       res.write("Comic agregado exitosamente");
26       res.end()
27     })
28   }
29
30   if(pathname == '/comics' && req.method == 'PUT'){
31     const id = params.get('id');
32     const datosArchivo = await fs.readFile('comics.txt');
33     const objetoArchivoOriginal = JSON.parse(datosArchivo);
34     let datosParaModificar;
35     req.on('data', (datos) => {
36       datosParaModificar = JSON.parse(datos);
37     })
38     req.on('end', async () => {
39       const comicOriginal = objetoArchivoOriginal[id]
```



```
40     const comicActualizado = {...comicOriginal,
41 ...datosParaModificar }
42
43     objetoArchivoOriginal[id] = comicActualizado;
44
45     await fs.writeFile('comics.txt',
46 JSON.stringify(objetoArchivoOriginal, null, 2));
47
48     res.write("Los datos han sido modificados
49 exitosamente");
50     res.end();
51   })
52 }
53
54 if(pathname == '/comics' && req.method == 'DELETE'){
55   const comicsOriginales = await fs.readFile('comics.txt');
56   const objetoComicsOriginal = JSON.parse(comicsOriginales);
57   const id = params.get('id');
58   delete objetoComicsOriginal[id];
59
60   await fs.writeFile('comics.txt',
61 JSON.stringify(objetoComicsOriginal, null, 2));
62
63   res.write("El comic ha sido eliminado exitosamente");
64   res.end();
65 }
66
67 })
68
69 servidor.listen(3000, function(){
70   console.log("Servidor iniciado en puerto 3000");
71 });
```

Archivo crea.js

```
1  const fs = require('fs/promises');
2  const { v4: uuidv4 } = require('uuid');
3
4  const creaComic = async (nuevoComic) => {
5    const archivoOriginal = await fs.readFile('comics.txt');
6    const datosOriginales = JSON.parse(archivoOriginal);
7    const id = uuidv4();
8
9    datosOriginales[id] = nuevoComic;
10    await fs.writeFile('comics.txt',
11 JSON.stringify(datosOriginales, null, 2));
12  }
13  module.exports = { creaComic: creaComic };
14
```

Hemos realizado unos pequeños cambios al código, para guardar nuestro servidor en una variable, y luego exportarla, e importarla en nuestro test.

```
4  const { creaComic } = require('./crea');
5
6
7  const servidor = http.createServer(async (req, res) => {
8    const { searchParams, pathname } = new URL(req.url, `http://${req.heade
9    const params = new URLSearchParams(searchParams);
10
11    if(pathname == '/comics' && req.method == 'GET'){
12      const lecturaArchivo = await fs.readFile('comics.txt');
13      res.statusCode = 200;
14      res.write(lecturaArchivo);
15    }
16  });
```

```
66
67  })
68
69  servidor.listen(3000, function(){
70    console.log("Servidor iniciado en puerto 3000");
71  });
72
73  module.exports = { servidor };
74
75
```

Añadiremos los códigos HTTP para las respuestas exitosas en cada uno de los verbos HTTP implementados.

GET

```
if(pathname == '/comics' && req.method == 'GET'){
  const lecturaArchivo = await fs.readFile('comics.txt');
  res.statusCode = 200;
  res.write(lecturaArchivo);
  res.end();
}
```

POST

```
17
18   if(pathname == '/comics' && req.method == 'POST'){
19       let datosComic;
20
21       req.on('data', (data) => {
22         datosComic = JSON.parse(data);
23       })
24       req.on('end', async () => {
25         console.log(datosComic);
26         await creaComic(datosComic);
27         res.statusCode = 200; ←
28         res.write("Comic agregado exitosamente");
29         res.end()
30       })
31     }
32
```

PUT

```
2
3   if(pathname == '/comics' && req.method == 'PUT'){
4     const id = params.get('id');
5     const datosArchivo = await fs.readFile('comics.txt');
6     const objetoArchivoOriginal = JSON.parse(datosArchivo);
7     let datosParaModificar;
8     req.on('data', (datos) => {
9       datosParaModificar = JSON.parse(datos);
10    })
11    req.on('end', async () => {
12      const comicOriginal = objetoArchivoOriginal[id]
13      const comicActualizado = {...comicOriginal, ...datosParaModificar }
14
15      objetoArchivoOriginal[id] = comicActualizado;
16
17      await fs.writeFile('comics.txt', JSON.stringify(objetoArchivoOriginal, null, 2));
18      res.statusCode = 200; ←
19      res.write("Los datos han sido modificados exitosamente");
20      res.end();
21    })
22  }
23
```

DELETE

```
if(pathname == '/comics' && req.method == 'DELETE'){
  const comicsOriginales = await fs.readFile('comics.txt');
  const objetoComicsOriginal = JSON.parse(comicsOriginales);
  const id = params.get('id');
  delete objetoComicsOriginal[id];

  await fs.writeFile('comics.txt', JSON.stringify(objetoComicsOriginal, null, 2));
  res.statusCode = 200;
  res.write("El comic ha sido eliminado exitosamente");
  res.end();
}
```

Ahora, crearemos la carpeta test, y un archivo en su interior llamado nombre get.js. Luego, importaremos **chai** y **chai-http**, configuraremos chai para que haga uso de chai http, e importaremos nuestro servidor.

```
JS get.js  X
test > JS get.js > ...
1  const chai = require('chai')
2  const chaiHttp = require('chai-http');
3  const { servidor } = require('../index');
4
5  chai.use(chaiHttp);
6
```

Al igual que en los tests anteriores, utilizaremos los métodos **describe()** e **it()**.

```
JS get.js  ●
test > JS get.js > ...
1  const chai = require('chai')
2  const chaiHttp = require('chai-http');
3  const { servidor } = require('../index');
4
5  chai.use(chaiHttp);
6
7  describe('Probando respuesta de servidor para metodo GET /comics', () => {
8    it('Comprueba que metodo GET responde con codigo 200', () => {
9
10
11    })
12  })
13
14
15
```

Utilizaremos el método `request()`, y pasaremos el servidor como argumento. Luego, utilizaremos el método `get()` para realizar la consulta a la ruta `"/comics"`.

```
6
7 describe('Probando respuesta de servidor para metodo GET /comics', () => {
8   it('Comprueba que metodo GET responde con codigo 200', () => {
9
10    chai.request(servidor).get('/comics').end((error, respuesta) => {
11
12    })
13  })
14 })
15 })
16
17
18
```

El método `expect` nos permite pasar directamente la respuesta de la consulta, para luego utilizar la sintaxis: `"to.have.status(código-de-respuesta)"`.

```
6
7 describe('Probando respuesta de servidor para metodo GET /comics', () => {
8   it('Comprueba que metodo GET responde con codigo 200', () => {
9
10    chai.request(servidor).get('/comics').end((error, respuesta) => {
11      chai.expect(respuesta).to.have.status(200);
12    })
13  })
14 })
15 })
16
17
```

El último paso que debemos hacer para que nuestro test funcione correctamente, es utilizar el método `done()`. La documentación nos advierte que, si corremos el test en una petición con tareas asíncronas, de la forma en que está escrito anteriormente, éste será exitoso antes de obtener la respuesta. Por lo tanto, debemos usar el método `done()`, después de verificar la respuesta, lo cual envía una señal a mocha haciéndole saber que el test ha terminado.

```

6
7 describe('Probando respuesta de servidor para metodo GET /comics', () => {
8   it('Comprueba que metodo GET responde con codigo 200', (done) => {
9
10     chai.request(servidor).get('/comics').end((error, respuesta) => {
11       chai.expect(respuesta).to.have.status(200);
12       done();
13     })
14   })
15 })
16

```

Probamos nuestro test con el comando `npm run test` (recuerda agregar `mocha` a la sección scripts, propiedad test).

```

16 },
17 "scripts": {
18   "test": "mocha"
19 },

```

```

C:\Users\testing-ejercicio>npm run test

> testing-ejercicio@1.0.0 test C:\Users\testing-ejercicio
> mocha

Servidor iniciado en puerto 3000

Probando respuesta de servidor para metodo GET /comics
✓ Comprueba que metodo GET responde con codigo 200

1 passing (27ms)

C:\Users\testing-ejercicio>

```

Para el método `post`, escribiremos un test similar. La única diferencia, es que debemos agregar el método `send` en la cadena de enviar la información para crear un nuevo comic, la cual pasaremos como argumento. Éste lo escribiremos en un nuevo archivo, llamado `post.js`, y lo guardaremos dentro de la misma carpeta `test`.

```
test > JS postjs > ...
1  const chai = require('chai')
2  const chaiHttp = require('chai-http');
3  const { servidor } = require('../index');
4
5  chai.use(chaiHttp);
6
7  describe('Probando respuesta de servidor para metodo POST /comics', () => {
8    it('Comprueba que respuesta de metodo POST es codigo 200', (done) => {
9
10     chai
11     .request(servidor)
12     .post('/comics')
13     .send({
14       "titulo": "El intrepido hombre araña",
15       "autor": "Stan Lee",
16       "issn": "12341234",
17       "cantidad": 340
18     })
19     .end((error, respuesta) => {
20       chai.expect(respuesta).to.have.status(200);
21       done();
22     })
23   })
24 })
25 })
```

Volveremos a ejecutar el comando **"npm run test"**. Es necesario mencionar que los test realizados son reales, por lo tanto, tu archivo comics.txt será modificado cada vez que realices uno que involucre nueva información, o eliminación de datos.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\testing-ejercicio>npm run test

> testing-ejercicio@1.0.0 test C:\Users\testing-ejercicio
> mocha

Servidor iniciado en puerto 3000

Probando respuesta de servidor para metodo GET /comics
✓ Comprueba que metodo GET responde con codigo 200

Probando respuesta de servidor para metodo POST /comics
{
  titulo: 'El intrepido hombre araña',
  autor: 'Stan Lee',
  issn: '12341234',
  cantidad: 340
}
✓ Comprueba que respuesta de metodo POST es codigo 200

2 passing (65ms)

C:\Users\testing-ejercicio>
```

Para el test del método **put**, también haremos uso del método **send()**, para enviar la información que queremos modificar. En este caso, debemos simular una URL con **querystring**, con un id válido, el cual puedes copiar desde tu archivo comics.txt.

```
testrrr > JS putjs > ...
1 const chai = require('chai')
2 const chaiHttp = require('chai-http');
3 const { servidor } = require('../index');
4
5 chai.use(chaiHttp);
6
7 describe('Probando respuesta de servidor para metodo PUT /comics', () => {
8   it('Comprueba que respuesta de metodo PUT es codigo 200', (done) => {
9
10     chai
11       .request(servidor)
12       .put('/comics?id=1a266aea-8ef1-48e9-be5a-cfa644f53769') ←
13       .send({
14         "titulo": "Capitan America",
15         "autor": "Stan Lee",
16         "issn": "43214321",
17         "cantidad": 230
18       })
19       .end((error, respuesta) => {
20         chai.expect(respuesta).to.have.status(200);
21         done();
22       })
23     })
24   })
25 })
26
```

Realizamos una vez más el test.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\testing-ejercicio>npm run test

> testing-ejercicio@1.0.0 test C:\Users\testing-ejercicio
> mocha

Servidor iniciado en puerto 3000

Probando respuesta de servidor para metodo GET /comics
✓ Comprueba que metodo GET responde con codigo 200

Probando respuesta de servidor para metodo POST /comics
{
  titulo: 'El intrepido hombre araña',
  autor: 'Stan Lee',
  issn: '12341234',
  cantidad: 340
}
✓ Comprueba que respuesta de metodo POST es codigo 200

Probando respuesta de servidor para metodo PUT /comics ←
✓ Comprueba que respuesta de metodo PUT es codigo 200

3 passing (77ms)

C:\Users\testing-ejercicio>
```

Y, por último, para el método **delete**, usaremos también una ruta con un id de prueba. Recuerda que, al realizar este test, el comic con el id utilizado en la URL será eliminado.


```
test > JS deletejs > ...
1  const chai = require('chai')
2  const chaiHttp = require('chai-http');
3  const { servidor } = require('../index');
4
5  chai.use(chaiHttp);
6
7  describe('Probando respuesta de servidor para metodo DELETE /comics', () => {
8    it('Comprueba que respuesta de metodo PUT es codigo 200', (done) => {
9
10     chai.request(servidor)
11       .delete('/comics?id=efab5c8e-fd3b-45b2-bc54-649bd647e734')
12       .end((error, respuesta) => {
13         chai.expect(respuesta).to.have.status(200);
14         done();
15       })
16     })
17   })
18 })
19
20
21
22
```

Realizando la última prueba, obtenemos.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\testing-ejercicio>npm run test

> testing-ejercicio@1.0.0 test C:\Users\testing-ejercicio
> mocha

Servidor iniciado en puerto 3000

Probando respuesta de servidor para metodo DELETE /comics
  ✓ Comprueba que respuesta de metodo PUT es codigo 200 (40ms)

Probando respuesta de servidor para metodo GET /comics
  ✓ Comprueba que metodo GET responde con codigo 200

Probando respuesta de servidor para metodo POST /comics
{
  titulo: 'El intrepido hombre araña',
  autor: 'Stan Lee',
  issn: '12341234',
  cantidad: 340
}
  ✓ Comprueba que respuesta de metodo POST es codigo 200

Probando respuesta de servidor para metodo PUT /comics
  ✓ Comprueba que respuesta de metodo PUT es codigo 200

4 passing (85ms)

C:\Users\testing-ejercicio>
```