



UNIVERSIDAD AUTONOMA DE COAHUILA



FACULTAD DE SISTEMAS



CALIDAD Y PRUEBAS DE SOFTWARE



## **Documentación de API de star wars con pruebas de software**

Integrantes:

Karina Alejandra Treviño Cabello

Fernanda Rodríguez Rodríguez

Azul Daniela Llamas Torres

Miriam Guadalupe Gómez Salazar

Erika Ruby Estrada Castillo

FECHA: 22/05/23

## Resumen

Este proyecto se centra en el desarrollo de un programa en JavaScript que se conecta a la API de Star Wars para obtener datos de naves, planetas y personajes. A través de esta conexión, se obtienen respuestas en formato JSON que contienen información detallada sobre los diferentes aspectos del universo de Star Wars.

El programa utiliza la biblioteca de axios para realizar solicitudes HTTP a la API y obtener los datos necesarios. Además, se emplea la biblioteca de axios-mock-adapter para realizar pruebas simulando respuestas de la API.

El proyecto también incorpora diversas técnicas de prueba, como el mock testing y mutation testing, con el fin de garantizar la calidad y robustez del código. Estas pruebas permiten simular situaciones y errores, evaluar la capacidad de respuesta del programa.

El mock testing se utiliza para simular respuestas de la API de Star Wars y asegurar que el programa maneje adecuadamente los datos obtenidos. Por otro lado, las pruebas de mutación permiten detectar modificaciones en el código fuente para evaluar la eficacia de las pruebas implementadas.

Este proyecto ofrece una manera eficiente y confiable de obtener datos de la API de Star Wars y ha sido sometido a pruebas rigurosas para asegurar su correcto funcionamiento y la calidad del código.

## Introducción

El proyecto de Conexión a la API de Star Wars y Pruebas es una aplicación en JavaScript diseñada para conectarse a la API de Star Wars y obtener datos relacionados con naves, planetas y personajes del universo de Star Wars. Esta API proporciona una amplia gama de información detallada sobre el universo de Star Wars, incluyendo datos sobre películas, personajes, naves espaciales, planetas y más.

Además de la funcionalidad de conexión a la API, el proyecto también se centra en la implementación de diferentes técnicas de prueba para garantizar la calidad del código y la confiabilidad de la aplicación. Se han utilizado técnicas como la prueba de mock testing y la prueba de mutación para evaluar la capacidad de respuesta del programa, identificar posibles errores.

## ¿Qué es una API?

Una API (Interfaz de Programación de Aplicaciones, por sus siglas en inglés) es un conjunto de reglas y protocolos que permite que diferentes aplicaciones se comuniquen entre sí. Proporciona una forma estandarizada y estructurada para que los desarrolladores puedan acceder y utilizar las funcionalidades o datos de un sistema, servicio o plataforma.

Una API define los métodos, parámetros y formatos de datos que se deben utilizar para realizar solicitudes y recibir respuestas. Actúa como un intermediario que permite la interacción y el intercambio de información entre diferentes aplicaciones, ya sea en el mismo sistema o a través de la red.

Las API se utilizan ampliamente en el desarrollo de software para diferentes propósitos, como acceder a servicios en la nube, interactuar con bases de datos, consumir datos de otras aplicaciones o servicios web, realizar operaciones en sistemas remotos, entre otros. Permiten que las aplicaciones se integren y se beneficien de las funcionalidades y datos proporcionados por otras aplicaciones sin tener que preocuparse por los detalles de implementación internos.

También es importante mencionar que las APIs se basan en protocolos de comunicación estándar, como HTTP (Hypertext Transfer Protocol), y suelen utilizar formatos de datos comunes como JSON (JavaScript Object Notation) o XML (eXtensible Markup Language) para representar la información intercambiada.

En resumen, una API es un conjunto de reglas y protocolos que permiten que las aplicaciones se comuniquen entre sí de manera estructurada y estandarizada, facilitando la integración y el intercambio de datos y funcionalidades.

## Starwars.js

Este proyecto es un programa en JavaScript que utiliza la biblioteca Axios para realizar solicitudes HTTP y obtener datos de la API de Star Wars (SWAPI). El objetivo del proyecto es obtener información sobre personajes, naves espaciales y planetas de Star Wars.

El código comienza importando la biblioteca Axios mediante la declaración `const axios = require('axios');`.

Luego, hay tres funciones principales:

- **getStarWarsCharacters():** Esta función realiza una solicitud GET a la URL ['https://swapi.dev/api/people/'](https://swapi.dev/api/people/) para obtener los datos de los personajes de Star Wars. Utiliza la función `axios.get()` para hacer la solicitud y espera a que la respuesta llegue. Cuando la respuesta llega, se ejecuta la función de devolución de llamada (callback) proporcionada en el método `then()`. Dentro de la función de devolución de llamada, se imprime el encabezado "Personajes de Star Wars:" y luego se itera sobre los resultados de los personajes obtenidos en la respuesta para imprimir el nombre de cada personaje.
- **getStarWarsStarships():** Esta función realiza una solicitud GET a la URL ['https://swapi.dev/api/starships/'](https://swapi.dev/api/starships/) para obtener los datos de las naves espaciales de Star Wars. Al igual que en la función anterior, se utiliza `axios.get()` para hacer la solicitud y se espera a que la respuesta llegue. Cuando la respuesta llega, se ejecuta la función de devolución de llamada proporcionada en el método `then()`. Dentro de la función de devolución de llamada, se imprime el encabezado "Naves espaciales de Star Wars:" y luego se itera sobre los resultados de las naves espaciales obtenidos en la respuesta para imprimir el nombre, el modelo y el fabricante de cada nave.
- **getStarWarsPlanetsAndResidents():** Esta función utiliza la palabra clave `async` para indicar que es una función asíncrona. Realiza una solicitud GET a la URL ['https://swapi.dev/api/planets/'](https://swapi.dev/api/planets/) para obtener los datos de los planetas de Star Wars. Al igual que en las funciones anteriores, se utiliza

axios.get() para hacer la solicitud y se espera a que la respuesta llegue. Cuando la respuesta llega, se asignan los resultados de los planetas a la variable planets. Luego, se itera sobre cada planeta y se imprime su nombre. Después, se itera sobre los URLs de los residentes de cada planeta y se realiza una solicitud GET a cada URL para obtener los datos de los residentes. Si la solicitud es exitosa, se imprime el nombre del residente.

En general, este proyecto utiliza Axios para realizar solicitudes HTTP a la API de Star Wars y obtener información sobre personajes, naves espaciales y planetas. Luego, imprime esta información en la consola.

## Starwars.test.js

Después de lo mencionado anteriormente ahora podemos proceder a agregar pruebas unitarias y utilizar las herramientas de mock testing y mutation testing.

### Paso 1: Configurar el entorno

- Para empezar este proyecto fue hecho en Visual Studio Code.
- Otro punto a tomar en cuenta es que utilizamos Node.js y NPM (Node Package Manager) los cuales son extensiones instaladas en el sistema.

### Paso 2: Instalar las herramientas de prueba

- Abrimos la terminal en Visual Studio Code.
- Ejecutamos los siguientes comandos para instalar las herramientas necesarias para las simulaciones:

```
...
```

```
npm install --save-dev jest axios-mock-adapter stryker-cli
```

```
...
```

Esto instalará Jest como framework de pruebas, Axios Mock Adapter para simular las solicitudes HTTP y Stryker para realizar mutation testing.

### Paso 3: Crear archivos de prueba

- En la carpeta llamada "tests" del proyecto.
- Dentro de la carpeta "tests", se crearon los siguientes archivos:
  - `starwars.test.js`: utilizado para las pruebas del archivo principal.

- ``axiosMock.js``: utilizado para configurar el mock de Axios.

#### Paso 4: Escribir pruebas unitarias con mock testing

- En el archivo ``starwars.test.js``, podemos comenzar a escribir las pruebas unitarias utilizando Jest y Axios Mock Adapter (los cuales se instalaron con anterioridad).

...

El código anterior es una descripción de las pruebas unitarias utilizando el framework de pruebas Jest. Ahora se explicará el código paso a paso y sus importaciones:

##### 1. Importaciones de módulos:

- ``const axios = require('axios')``: Importa el módulo ``axios``, que se utiliza para hacer solicitudes HTTP.

- ``const MockAdapter = require('axios-mock-adapter')``: Importa el módulo ``axios-mock-adapter``, que se utiliza para simular respuestas de solicitudes HTTP en las pruebas unitarias.

- ``const { getStarWarsCharacters, getStarWarsStarships, getStarWarsPlanetsAndResidents } = require('../main')``: Importa las funciones que se van a probar desde el archivo ``main.js``.

2. ``describe``: Define un bloque de pruebas relacionadas y proporciona una descripción para ese bloque. En este caso, el bloque se llama "Star Wars API Tests".

3. ``let mock``: Declara una variable ``mock`` que se utilizará para simular las respuestas de las solicitudes HTTP en las pruebas.

4. ``beforeEach``: Esta función se ejecuta antes de cada prueba y crea una nueva instancia de ``MockAdapter`` utilizando ``axios`` y asigna esa instancia a la variable ``mock``. Esto asegura que cada prueba comience con un estado limpio y sin respuestas simuladas previas.

5. ``afterEach``: Esta función se ejecuta después de cada prueba y restablece la instancia ``mock``, eliminando las respuestas simuladas previas.

6. ``afterAll``: Esta función se ejecuta después de todas las pruebas y restaura ``mock`` a su estado original, asegurándose de que no haya efectos secundarios en otras pruebas o en el entorno.

7. ``it``: Define una prueba individual. Cada prueba tiene un nombre y una función que contiene el código de la prueba.

- La primera prueba, llamada ``'should fetch and display Star Wars characters'``, prueba la función ``getStarWarsCharacters``. Simula una respuesta exitosa de la API que contiene datos ficticios de personajes de Star Wars y verifica que se muestren correctamente en la consola.

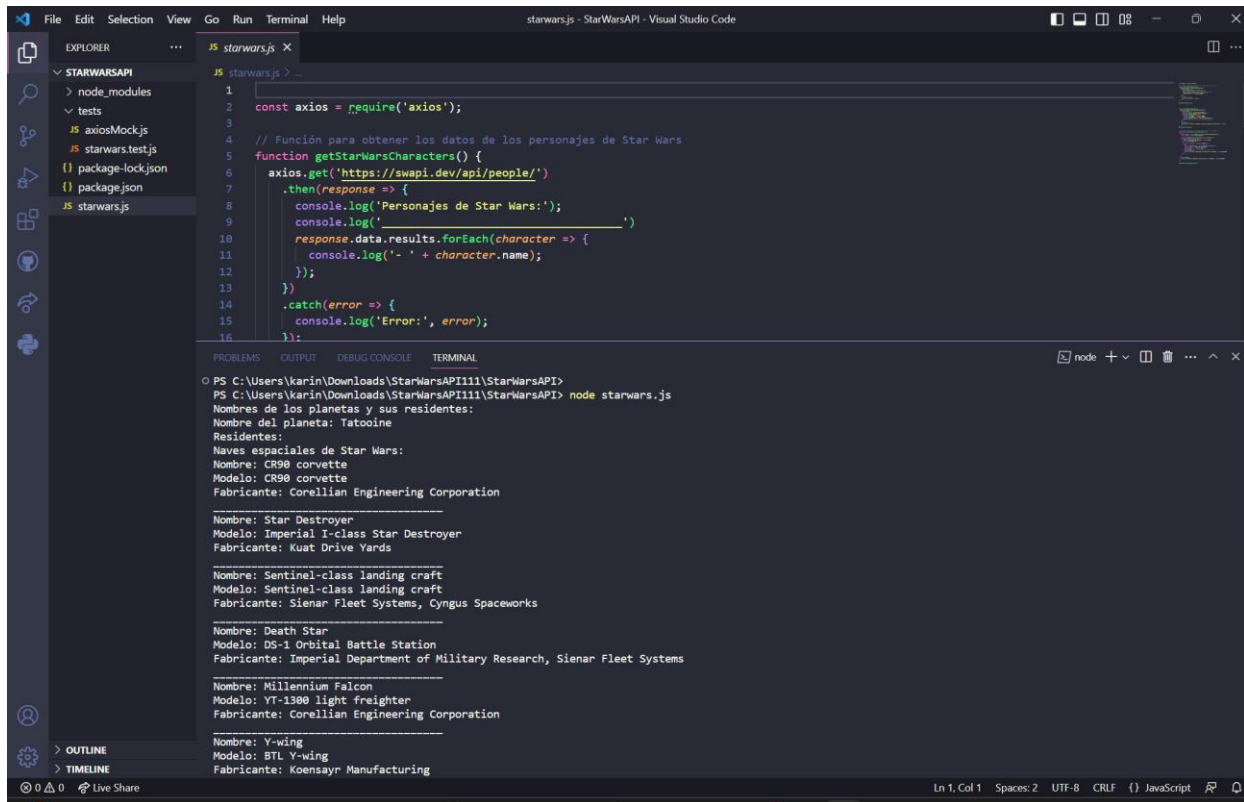
- La segunda prueba, llamada ``'should fetch and display Star Wars starships'``, prueba la función ``getStarWarsStarships``. Simula una respuesta exitosa de la API que contiene datos ficticios de naves espaciales de Star Wars y verifica que se muestren correctamente en la consola.

- La tercera prueba, llamada ``'should fetch and display Star Wars planets and residents'``, prueba la función ``getStarWarsPlanetsAndResidents``. Simula respuestas exitosas de la API que contienen datos ficticios de planetas y residentes de Star Wars, y verifica que se muestren correctamente en la consola.

Cada prueba utiliza ``mock.onGet()`` para simular la respuesta de una solicitud HTTP a una URL específica y ``expect`` para verificar que se muestren los resultados esperados en la consola utilizando ``console.log``.

Y de esta forma se estructuran las pruebas unitarias utilizando Jest y el módulo ``axios-mock-adapter``.

# Test y Ejecución del código



```
1
2 const axios = require('axios');
3
4 // Función para obtener los datos de los personajes de Star Wars
5 function getStarWarsCharacters() {
6   axios.get('https://swapi.dev/api/people/')
7     .then(response => {
8       console.log('Personajes de Star Wars:');
9       console.log('-----');
10      response.data.results.forEach(character => {
11        console.log('- ' + character.name);
12      });
13    })
14    .catch(error => {
15      console.log('Error:', error);
16    });
17 }
```

PS C:\Users\karin\Downloads\StarWarsAPI111\StarWarsAPI> node starwars.js

Nombres de los planetas y sus residentes:

Nombre del planeta: Tatooine

Residentes:

Naves espaciales de Star Wars:

Nombre: CR90 corvette

Modelo: CR90 corvette

Fabricante: Corellian Engineering Corporation

-----

Nombre: Star Destroyer

Modelo: Imperial I-class Star Destroyer

Fabricante: Kuat Drive Yards

-----

Nombre: Sentinel-class landing craft

Modelo: Sentinel-class landing craft

Fabricante: Sienar Fleet Systems, Cyngus Spaceworks

-----

Nombre: Death Star

Modelo: DS-1 Orbital Battle Station

Fabricante: Imperial Department of Military Research, Sienar Fleet Systems

-----

Nombre: Millennium Falcon

Modelo: YT-1300 light freighter

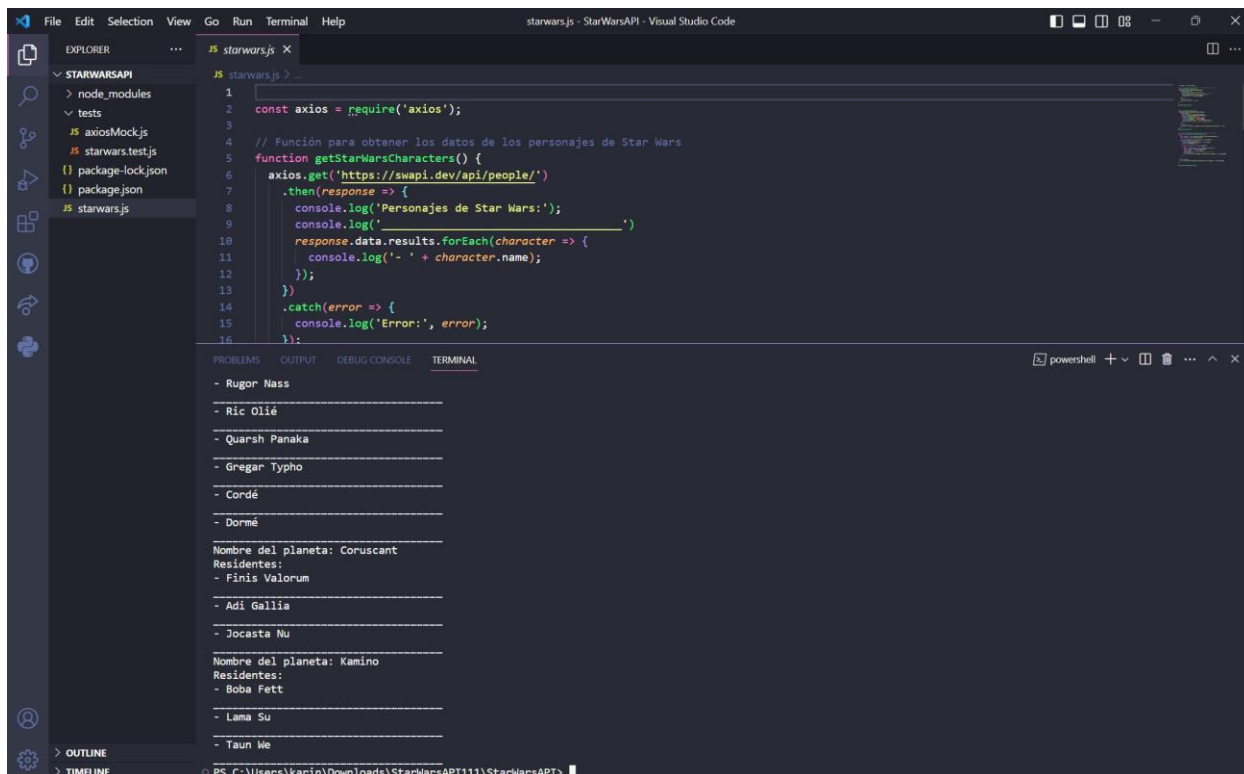
Fabricante: Corellian Engineering Corporation

-----

Nombre: Y-wing

Modelo: BTL Y-wing

Fabricante: Koensayr Manufacturing



```
1
2 const axios = require('axios');
3
4 // Función para obtener los datos de los personajes de Star Wars
5 function getStarWarsCharacters() {
6   axios.get('https://swapi.dev/api/people/')
7     .then(response => {
8       console.log('Personajes de Star Wars:');
9       console.log('-----');
10      response.data.results.forEach(character => {
11        console.log('- ' + character.name);
12      });
13    })
14    .catch(error => {
15      console.log('Error:', error);
16    });
17 }
```

- Rugor Nass

-----

- Ric Olié

-----

- Quarsh Panaka

-----

- Gregar Typho

-----

- Cordé

-----

- Dormé

-----

Nombre del planeta: Coruscant

Residentes:

- Finis Valorum

-----

- Adi Gallia

-----

- Jocasta Nu

-----

Nombre del planeta: Kamino

Residentes:

- Boba Fett

-----

- Lama Su

-----

- Taun We



File Edit Selection View Go Run Terminal Help starwars.test.js - StarWarsAPI - Visual Studio Code

EXPLORER

- STARWARSAPI
  - node\_modules
  - tests
    - starwars.test.js
  - package-lock.json
  - package.json
  - starwars.js

```
tests > JS starwars.test.js > describe('Star Wars API Tests') callback
1  const axios = require('axios');
2  const MockAdapter = require('axios-mock-adapter');
3  const { getStarWarsCharacters, getStarWarsStarships, getStarWarsPlanetsAndResidents } = require('../main');
4
5  describe('Star Wars API Tests', () => {
6    let mock;
7
8    beforeEach(() => {
9      mock = new MockAdapter(axios);
10    });
11
12    afterEach(() => {
13      mock.reset();
14    });
15
16    afterAll(() => {
17      mock.restore();
18    });
19
20    it('should fetch and display Star Wars characters', async () => {
21      const charactersData = {
22        results: [
23          { name: 'Luke Skywalker' },
24          { name: 'Darth Vader' },
25          { name: 'Princess Leia' }
26        ]
27      };
28
29      mock.onGet('https://swapi.dev/api/people/').reply(200, charactersData);
30
31      const consoleSpy = jest.spyOn(console, 'log');
32
33      await getStarWarsCharacters();
34
35      expect(consoleSpy).toHaveBeenCalledTimes(4); // Incluye el título "Personajes de Star Wars:"
36      expect(consoleSpy).toHaveBeenNthCalledWith(2, '- Luke Skywalker');
37      expect(consoleSpy).toHaveBeenNthCalledWith(3, '- Darth Vader');
38      expect(consoleSpy).toHaveBeenNthCalledWith(4, '- Princess Leia');
39    });
40
41    it('should fetch and display Star Wars starships', async () => {
42      const starshipsData = {
```

Ln 7, Col 1 Spaces: 2 UTF-8 CRLF JavaScript

The screenshot shows a VS Code editor window with the following details:

- Top Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Explorer Sidebar:**
  - starwars.test.js (selected)
  - STARWARSAPI
    - node\_modules
    - tests
      - axiosMock.js
      - starwars.test.js (selected)
      - package-lock.json
      - package.json
      - starwars.js
- Main Editor:**

```

describe('Star Wars API Tests') callback

await getStarWarsShips();

expect(consoleSpy).toHaveBeenCalledTimes(10); // Incluye el título "Naves espaciales de Star Wars:" y las divisiones "
expect(consoleSpy).toHaveBeenCalledNthCalledWith(2, 'Nombre:', 'Starship 1');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(3, 'Modelo:', 'Model 1');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(4, 'Fabricante:', 'Manufacturer 1');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(5, ' ');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(6, 'Nombre:', 'Starship 2');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(7, 'Modelo:', 'Model 2');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(8, 'Fabricante:', 'Manufacturer 2');
expect(consoleSpy).toHaveBeenCalledNthCalledWith(9, ' ');
});

it('should fetch and display Star Wars planets and residents', async () => {
  const planetsData = {
    results: [
      { name: 'Planet 1', residents: ['https://swapi.dev/api/people/1/', 'https://swapi.dev/api/people/2/'] },
      { name: 'Planet 2', residents: ['https://swapi.dev/api/people/3/'] },
    ]
  };

  const residentsData = {
    name: 'Resident Name'
  };

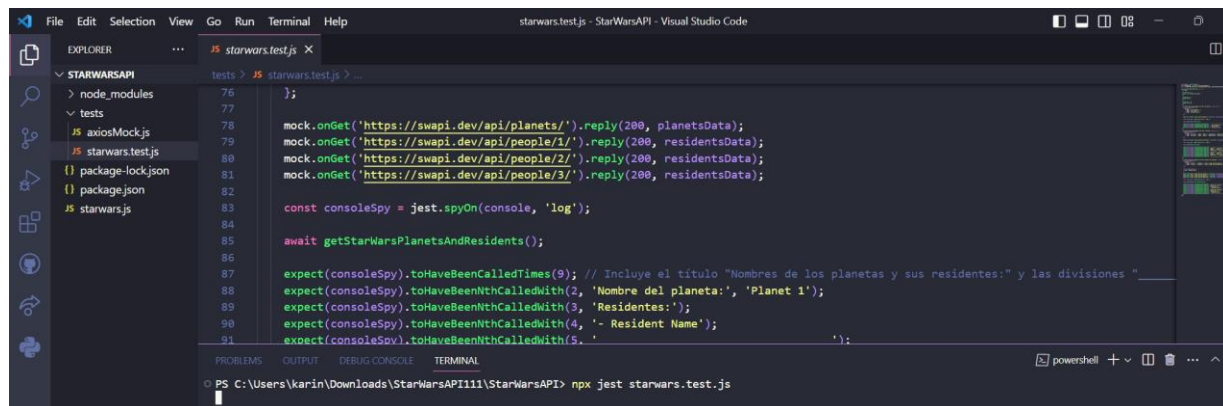
  mock.onGet('https://swapi.dev/api/planets/').reply(200, planetsData);
  mock.onGet('https://swapi.dev/api/people/1/').reply(200, residentsData);
  mock.onGet('https://swapi.dev/api/people/2/').reply(200, residentsData);
  mock.onGet('https://swapi.dev/api/people/3/').reply(200, residentsData);

  const consoleSpy = jest.spyOn(console, 'log');

  await getStarWarsPlanetsAndResidents();

  expect(consoleSpy).toHaveBeenCalledTimes(9); // Incluye el título "Nombres de los planetas y sus residentes:" y las divisiones "
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(2, 'Nombre del planeta:', 'Planet 1');
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(3, 'Residentes:');
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(4, '- Resident Name');
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(5, ' ');
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(6, 'Nombre del planeta:', 'Planet 2');
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(7, 'Residentes:');
  expect(consoleSpy).toHaveBeenCalledNthCalledWith(8, '- Resident Name');

```
- Bottom Bar:** OUTLINE, TIMELINE



```
File Edit Selection View Go Run Terminal Help
starwars.test.js - StarWarsAPI - Visual Studio Code

EXPLORER
  STARWARSAPI
    > node_modules
    > tests
      JS axiosMock.js
      JS starwars.test.js
    package-lock.json
    package.json
    JS starwars.js

starwars.test.js
76
77
78
79 mock.onGet('https://swapi.dev/api/planets/').reply(200, planetsData);
80 mock.onGet('https://swapi.dev/api/people/1/').reply(200, residentsData);
81 mock.onGet('https://swapi.dev/api/people/2/').reply(200, residentsData);
82 mock.onGet('https://swapi.dev/api/people/3/').reply(200, residentsData);
83
84 const consoleSpy = jest.spyOn(console, 'log');
85
86 await getStarWarsPlanetsAndResidents();
87
88 expect(consoleSpy).toHaveBeenCalledTimes(9); // Incluye el título "Nombres de los planetas y sus residentes:" y las divisiones "
89 expect(consoleSpy).toHaveBeenCalledWith(2, 'Nombre del planeta:', 'Planet 1');
90 expect(consoleSpy).toHaveBeenCalledWith(3, 'Residentes:');
91 expect(consoleSpy).toHaveBeenCalledWith(4, '- Resident Name');
92 expect(consoleSpy).toHaveBeenCalledWith(5, '');
```



```
at Array.forEach (<anonymous>)
at starwars.js:8:27

PS C:\Users\karin\Downloads\StarWarsAPI111\StarWarsAPI> npx jest starwars.js
No tests found, exiting with code 1
Run with --passWithNoTests to exit with code 0
In C:\Users\karin\Downloads\StarWarsAPI111\StarWarsAPI
  5 files checked.
  testMatch: **/_tests_/**/*.[jt]s?(x), **/?(*.)(spec|test).[jt]s?(x) - 1 match
  testPathIgnorePatterns: \\node_modules\\ - 5 matches
  testRegex: - 0 matches
  Pattern: starwars.js - 0 matches

PS C:\Users\karin\Downloads\StarWarsAPI111\StarWarsAPI>
```

## Conclusión

Como ya se mencionó anteriormente en el proyecto de Conexión a la API de Star Wars y Pruebas, se desarrolló una aplicación funcional y eficiente que permite conectarse a la API de Star Wars para obtener datos relevantes.

La incorporación de diferentes técnicas de prueba, como la prueba de simulación y la prueba de mutación, ha demostrado ser fundamental para garantizar la calidad del código y la confiabilidad de la aplicación. Estas pruebas permitieron simular situaciones reales y detectar posibles errores, lo que ayudó a fortalecer la robustez y el rendimiento del programa.

El uso del mock testing permitió simular respuestas de la API y validar la correcta manipulación de los datos por parte del programa. Esto aseguró que la aplicación funcionará correctamente incluso en diferentes escenarios de respuesta de la API.

El Mutation Testing reveló ser una técnica efectiva para evaluar la efectividad de las pruebas implementadas. La detección de cambios en el código fuente no pudo identificar áreas de mejora en las pruebas y garantizar una cobertura adecuada del código.

En conclusión, el proyecto de Conexión a la API de Star Wars y Pruebas ha demostrado ser exitoso en la obtención de datos de la API de Star Wars y en la implementación de pruebas efectivas para garantizar la calidad del código. Esta

aplicación proporciona una solución confiable y eficiente para acceder a la información del universo de Star Wars y puede ser utilizada como base para futuros proyectos o como referencia para implementaciones similares.