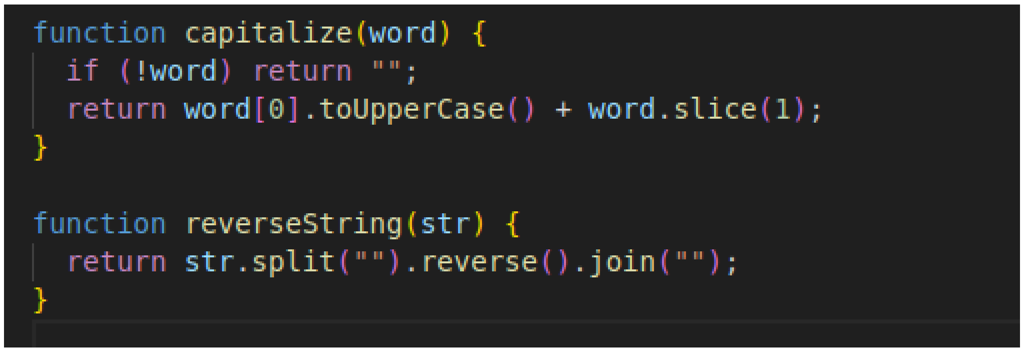
Testing Exercises

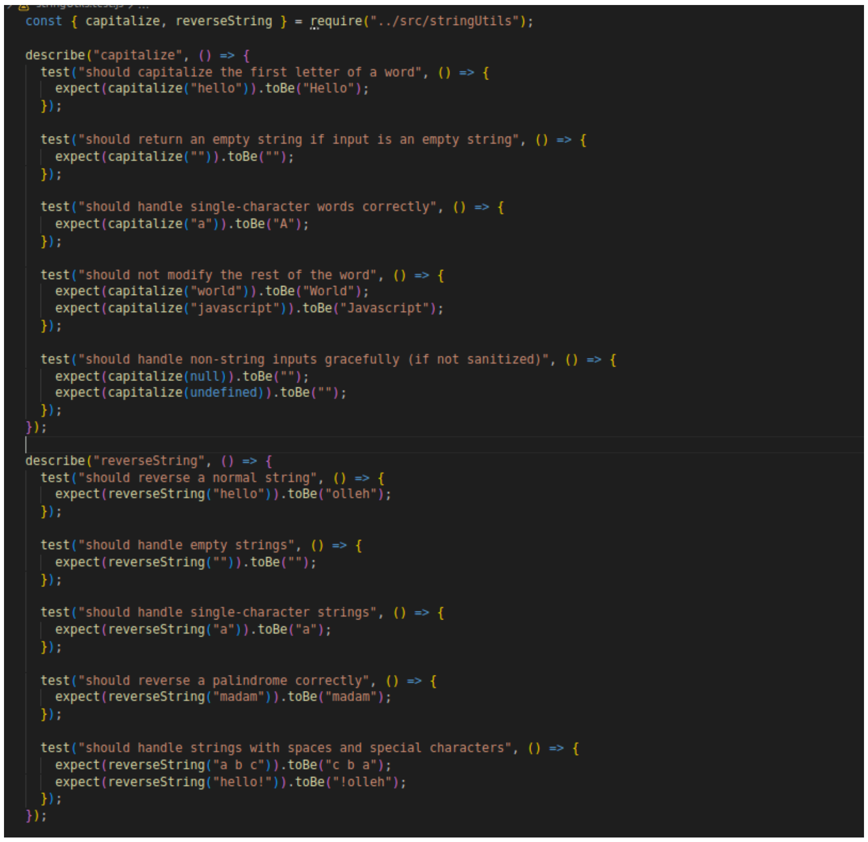
1. Fundamentals: String Utilities

Objective: Write unit tests for string utility functions.
Code to Test:



```
function capitalize(word) {
  if (!word) return "";
  return word[0].toUpperCase() + word.slice(1);
}

function reverseString(str) {
  return str.split("").reverse().join("");
}
```

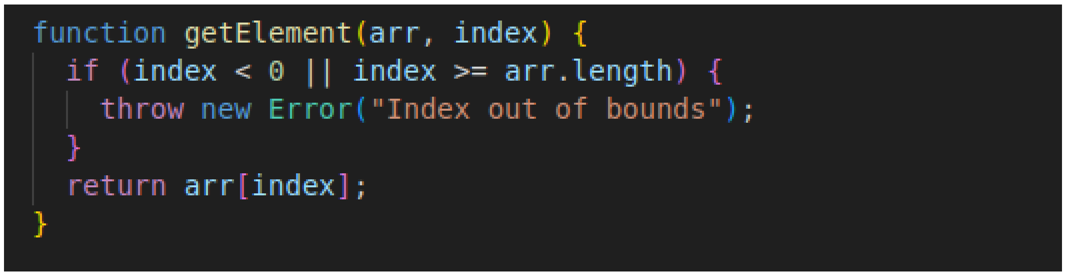
Exercise:
•Write tests to validate the capitalize function, including handling empty strings and single-character words.
•Write tests for reverseString, including edge cases with palindromes and empty strings.



```
const { capitalize, reverseString } = require("../src/stringUtils");

describe("capitalize", () => {
  test("should capitalize the first letter of a word", () => {
    expect(capitalize("hello")).toBe("Hello");
  });

  test("should return an empty string if input is an empty string", () => {
    expect(capitalize("")).toBe("");
  });

  test("should handle single-character words correctly", () => {
    expect(capitalize("a")).toBe("A");
  });

  test("should not modify the rest of the word", () => {
    expect(capitalize("world")).toBe("World");
    expect(capitalize("javascript")).toBe("Javascript");
  });

  test("should handle non-string inputs gracefully (if not sanitized)", () => {
    expect(capitalize(null)).toBe("");
    expect(capitalize(undefined)).toBe("");
  });
});

describe("reverseString", () => {
  test("should reverse a normal string", () => {
    expect(reverseString("hello")).toBe("olleh");
  });

  test("should handle empty strings", () => {
    expect(reverseString("")).toBe("");
  });

  test("should handle single-character strings", () => {
    expect(reverseString("a")).toBe("a");
  });

  test("should reverse a palindrome correctly", () => {
    expect(reverseString("madam")).toBe("madam");
  });

  test("should handle strings with spaces and special characters", () => {
    expect(reverseString("a b c")).toBe("c b a");
    expect(reverseString("hello!")).toBe("!olleh");
  });
});
```

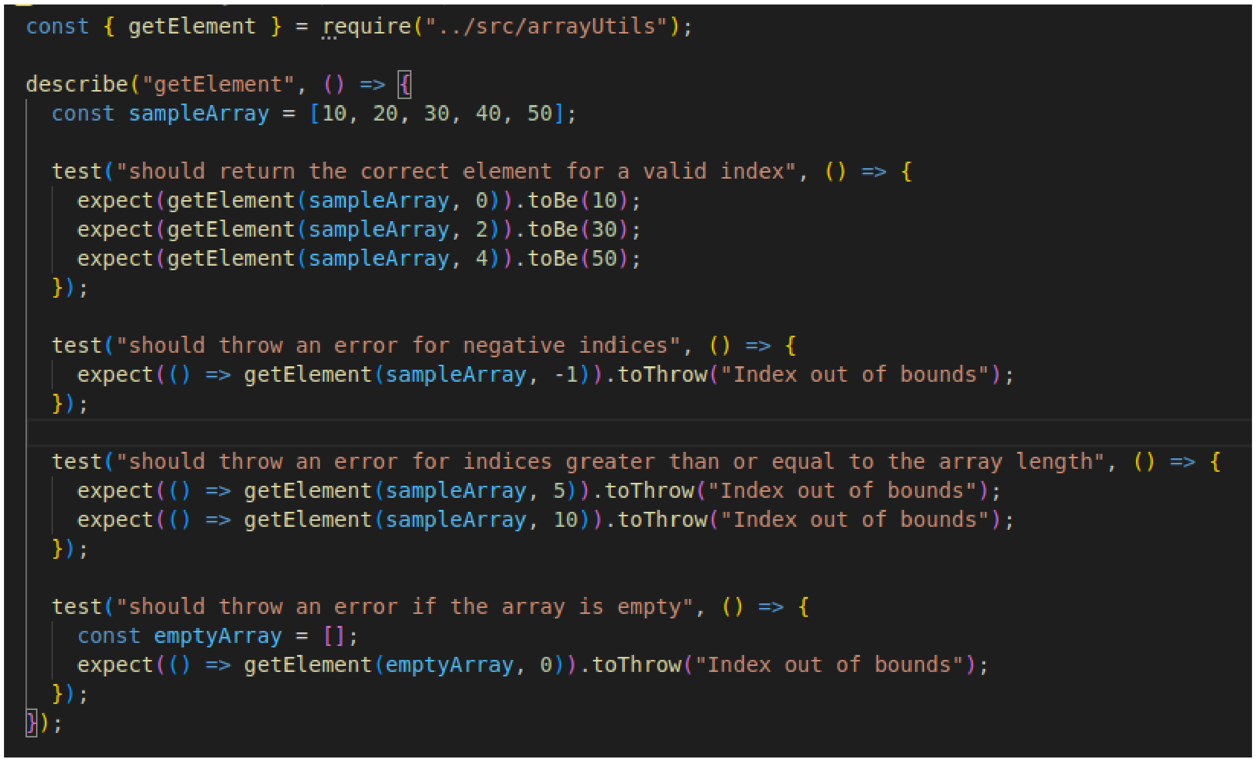
2. Error Handling: Array Index

Objective: Test a function that accesses an array by index and handles out-of-bounds
cases.
Code to Test:



```javascript
function getElement(arr, index) {
  if (index < 0 || index >= arr.length) {
    throw new Error("Index out of bounds");
  }
  return arr[index];
}
```

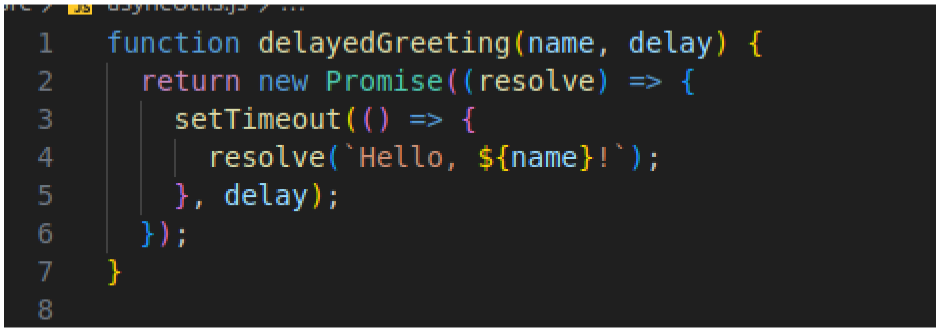
Exercise:•Write tests for valid index values.
•Write tests to check if the error is thrown for negative indices and out-of-range indices.



```javascript
const { getElement } = require("../src/arrayUtils");

describe("getElement", () => {
  const sampleArray = [10, 20, 30, 40, 50];

  test("should return the correct element for a valid index", () => {
    expect(getElement(sampleArray, 0)).toBe(10);
    expect(getElement(sampleArray, 2)).toBe(30);
    expect(getElement(sampleArray, 4)).toBe(50);
  });

  test("should throw an error for negative indices", () => {
    expect(() => getElement(sampleArray, -1)).toThrow("Index out of bounds");
  });

  test("should throw an error for indices greater than or equal to the array length", () => {
    expect(() => getElement(sampleArray, 5)).toThrow("Index out of bounds");
    expect(() => getElement(sampleArray, 10)).toThrow("Index out of bounds");
  });

  test("should throw an error if the array is empty", () => {
    const emptyArray = [];
    expect(() => getElement(emptyArray, 0)).toThrow("Index out of bounds");
  });
});
```

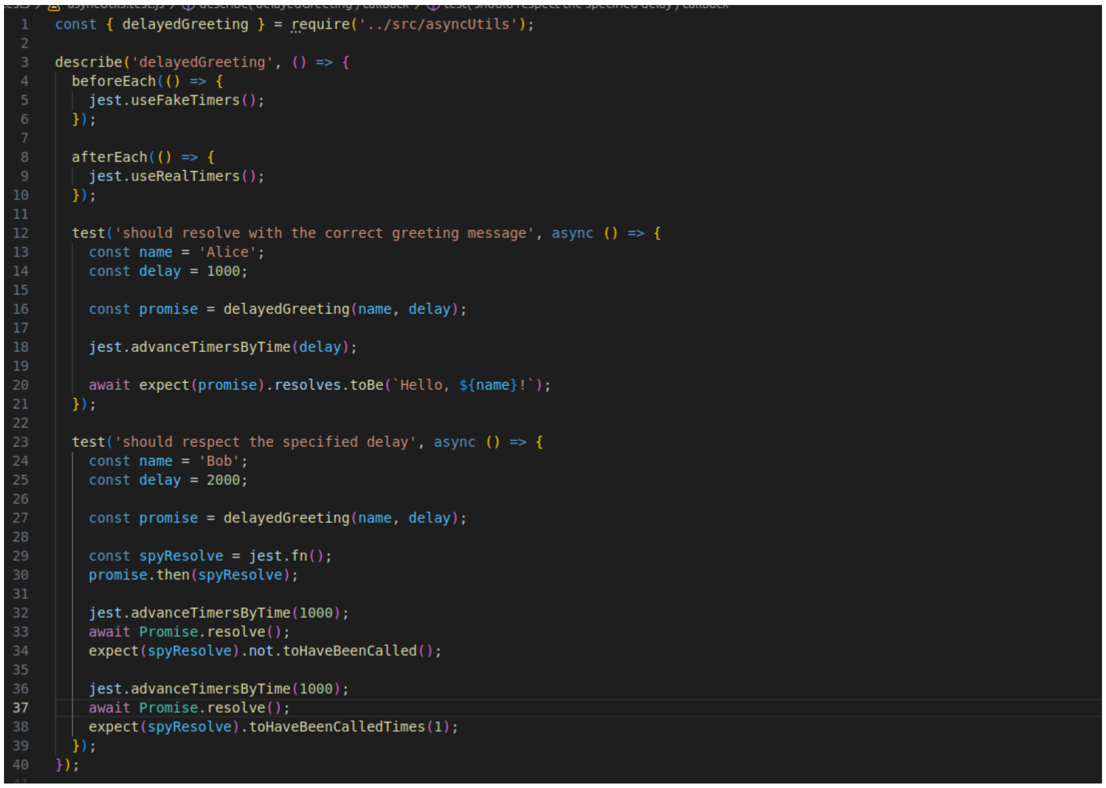
3. Async Functions: Delayed Greeting

Objective: Test an asynchronous function with a delay.
Code to Test:



```javascript
function delayedGreeting(name, delay) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(`Hello, ${name}!`);
    }, delay);
  });
}

```

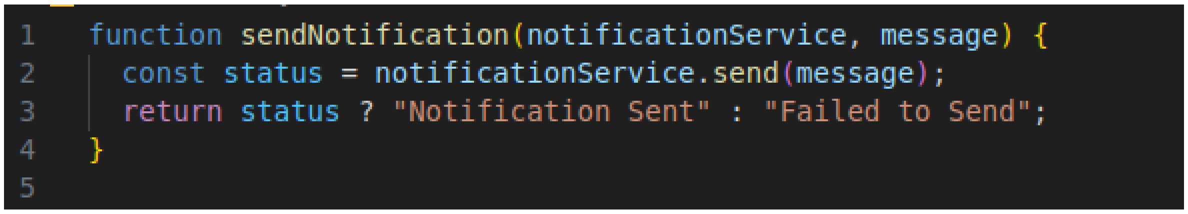
Exercise:
•Write tests for the resolved greeting message.
•Use a mock timer to validate that the function respects the delay.



```javascript
const { delayedGreeting } = require('../src/asyncUtils');

describe('delayedGreeting', () => {
  beforeEach(() => {
    jest.useFakeTimers();
  });

  afterEach(() => {
    jest.useRealTimers();
  });

  test('should resolve with the correct greeting message', async () => {
    const name = 'Alice';
    const delay = 1000;

    const promise = delayedGreeting(name, delay);

    jest.advanceTimersByTime(delay);

    await expect(promise).resolves.toBe(`Hello, ${name}!`);
  });

  test('should respect the specified delay', async () => {
    const name = 'Bob';
    const delay = 2000;

    const promise = delayedGreeting(name, delay);

    const spyResolve = jest.fn();
    promise.then(spyResolve);

    jest.advanceTimersByTime(1000);
    await Promise.resolve();
    expect(spyResolve).not.toHaveBeenCalled();

    jest.advanceTimersByTime(1000);
    await Promise.resolve();
    expect(spyResolve).toHaveBeenCalledTimes(1);
  });
});
```

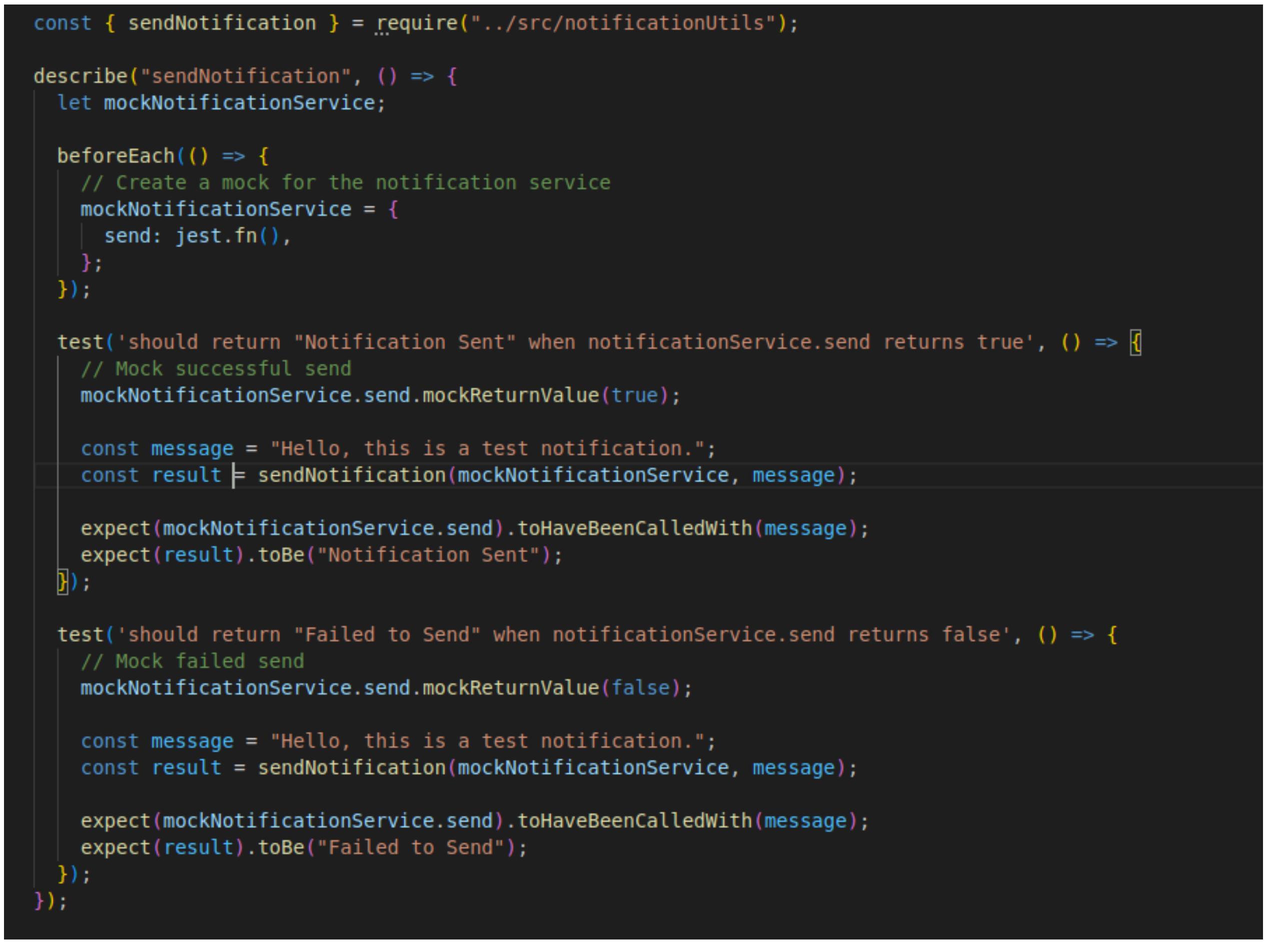
4. Mocking: Notification Service

Objective: Test a notification function using mocks.
Code to Test:



```javascript
function sendNotification(notificationService, message) {
  const status = notificationService.send(message);
  return status ? "Notification Sent" : "Failed to Send";
}

```

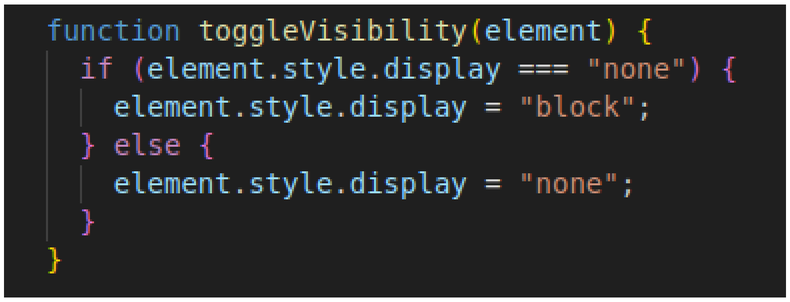
Exercise:
•Mock notificationService to simulate both successful and failed notification sending.
•Write tests to ensure the return message matches the scenario.



```javascript
const { sendNotification } = require("../src/notificationUtils");

describe("sendNotification", () => {
  let mockNotificationService;

  beforeEach(() => {
    // Create a mock for the notification service
    mockNotificationService = {
      send: jest.fn(),
    };
  });

  test('should return "Notification Sent" when notificationService.send returns true', () => {
    // Mock successful send
    mockNotificationService.send.mockReturnValue(true);

    const message = "Hello, this is a test notification.";
    const result = sendNotification(mockNotificationService, message);

    expect(mockNotificationService.send).toHaveBeenCalledWith(message);
    expect(result).toBe("Notification Sent");
  });

  test('should return "Failed to Send" when notificationService.send returns false', () => {
    // Mock failed send
    mockNotificationService.send.mockReturnValue(false);

    const message = "Hello, this is a test notification.";
    const result = sendNotification(mockNotificationService, message);

    expect(mockNotificationService.send).toHaveBeenCalledWith(message);
    expect(result).toBe("Failed to Send");
  });
});
```

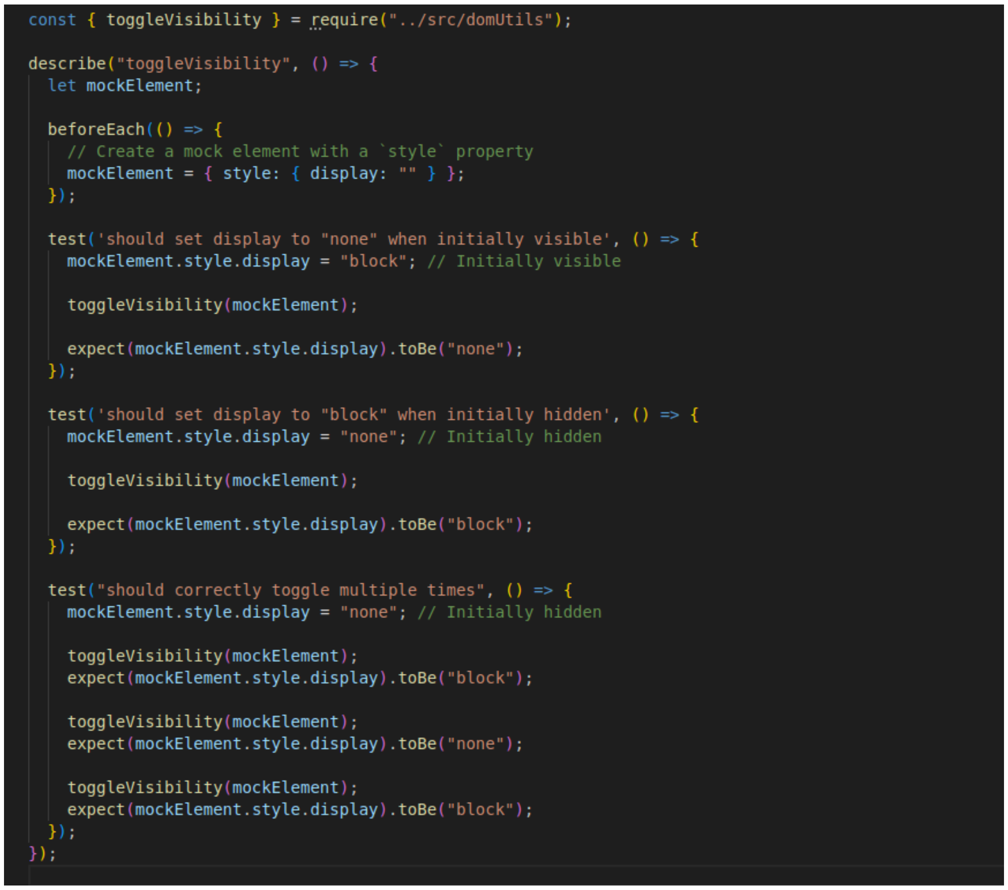
5. Spying: DOM Manipulation

Objective: Test a DOM manipulation function using spies.
Code to Test:



```javascript
function toggleVisibility(element) {
  if (element.style.display === "none") {
    element.style.display = "block";
  } else {
    element.style.display = "none";
  }
}
```

Exercise:
•Use a spy to check if the style.display property changes correctly.
•Write tests to validate toggling visibility when the element is initially visible or hidden.

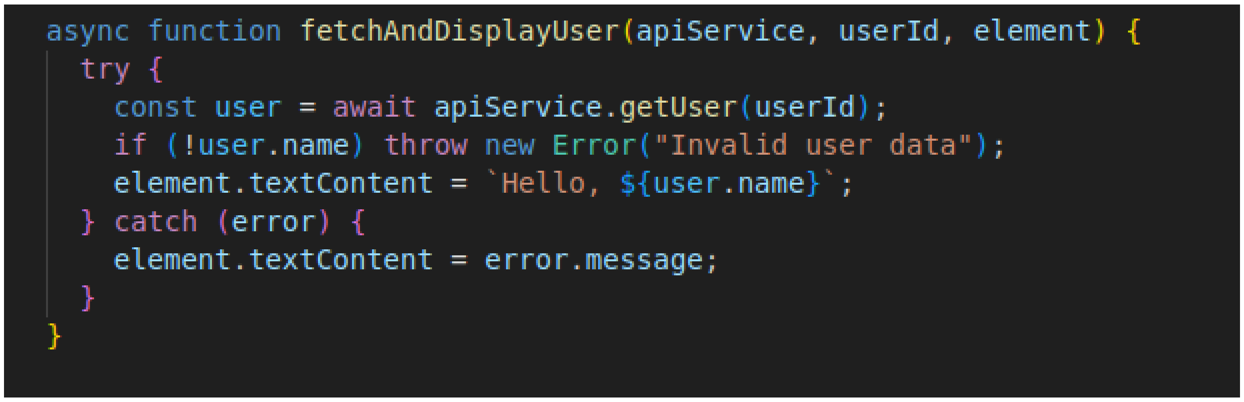


```javascript
const { toggleVisibility } = require("../src/domUtils");

describe("toggleVisibility", () => {
  let mockElement;

  beforeEach(() => {
    // Create a mock element with a `style` property
    mockElement = { style: { display: "" } };
  });

  test('should set display to "none" when initially visible', () => {
    mockElement.style.display = "block"; // Initially visible

    toggleVisibility(mockElement);

    expect(mockElement.style.display).toBe("none");
  });

  test('should set display to "block" when initially hidden', () => {
    mockElement.style.display = "none"; // Initially hidden

    toggleVisibility(mockElement);

    expect(mockElement.style.display).toBe("block");
  });

  test("should correctly toggle multiple times", () => {
    mockElement.style.display = "none"; // Initially hidden

    toggleVisibility(mockElement);
    expect(mockElement.style.display).toBe("block");

    toggleVisibility(mockElement);
    expect(mockElement.style.display).toBe("none");

    toggleVisibility(mockElement);
    expect(mockElement.style.display).toBe("block");
  });
});
```

Bonus Challenge: Integrate All Concepts

Objective: Create a function that fetches user data, validates it, and displays it in the
DOM.

Code to Test:

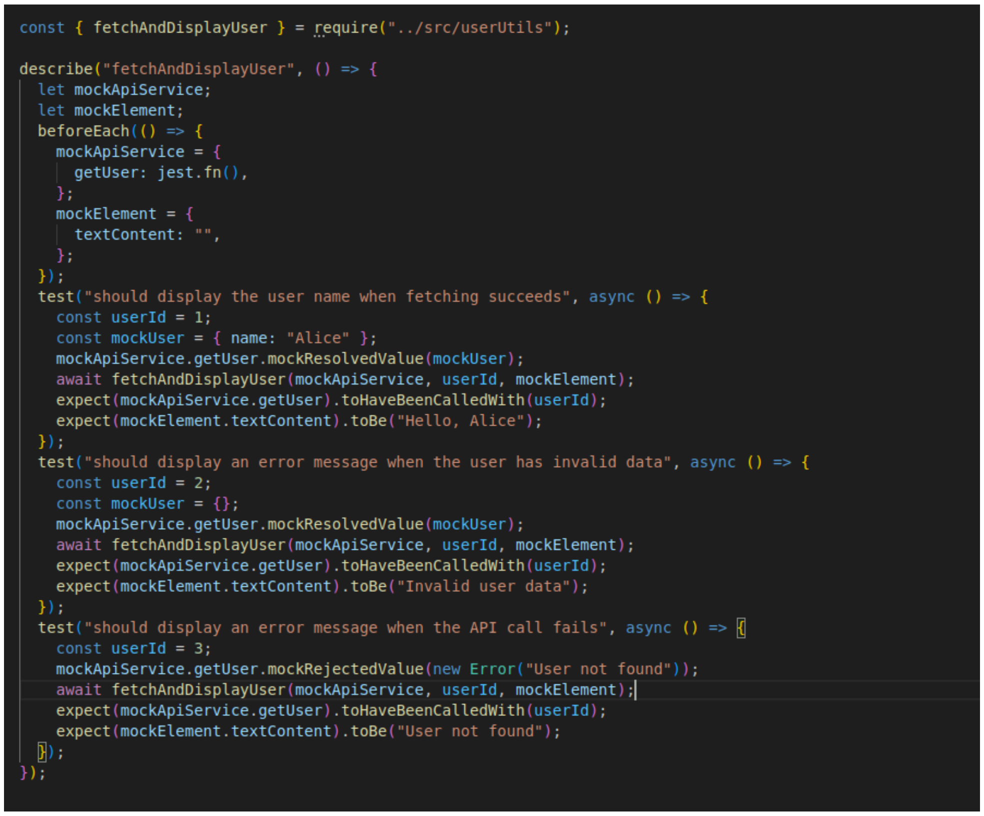


```javascript
async function fetchAndDisplayUser(apiService, userId, element) {
  try {
    const user = await apiService.getUser(userId);
    if (!user.name) throw new Error("Invalid user data");
    element.textContent = `Hello, ${user.name}`;
  } catch (error) {
    element.textContent = error.message;
  }
}
```

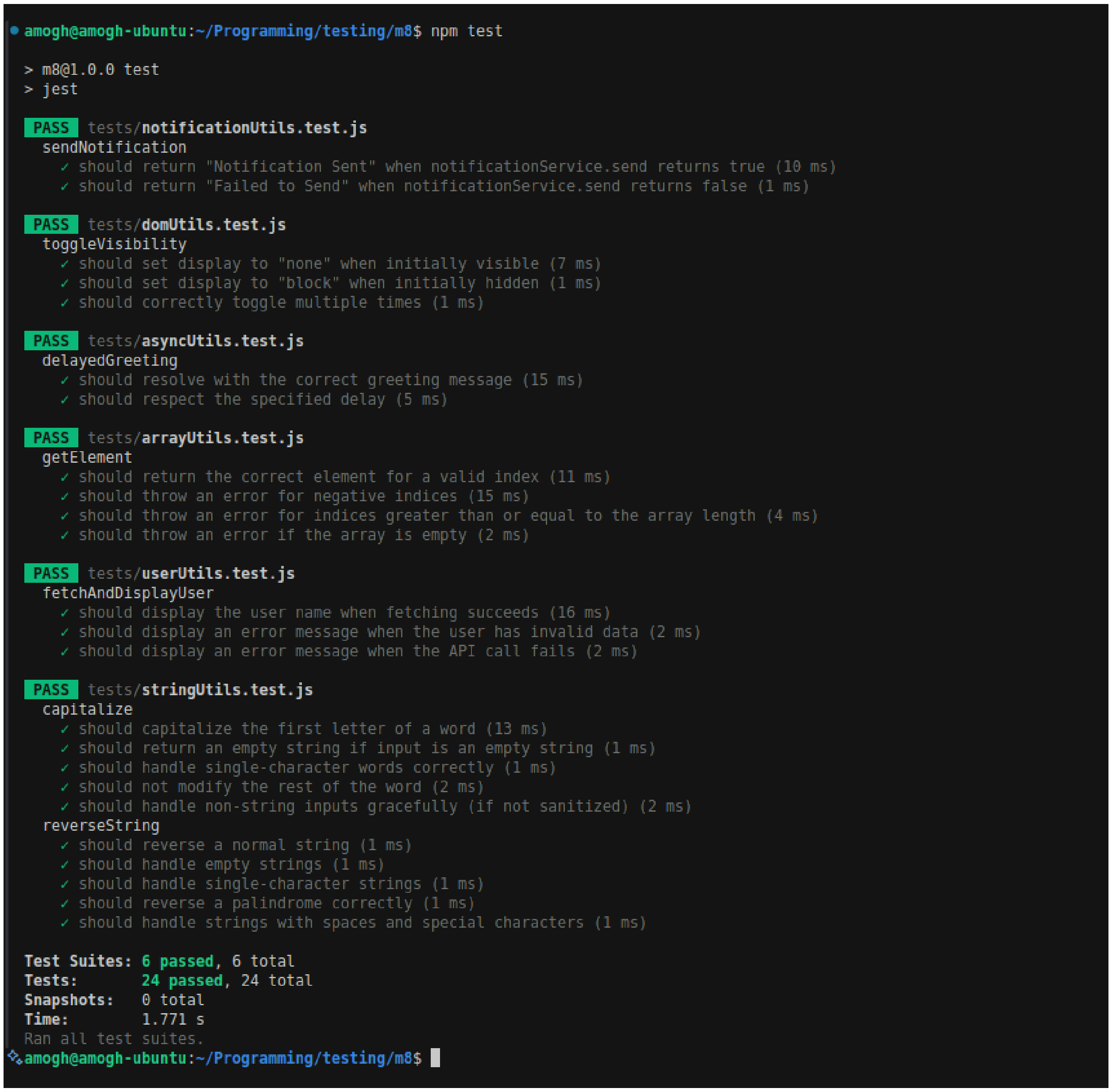
Exercise:

•Mock the apiService to test successful and failed user fetch scenarios.

•Spy on the DOM element's textContent property to validate correct content updates.



```javascript
const { fetchAndDisplayUser } = require("../src/userUtils");

describe("fetchAndDisplayUser", () => {
  let mockApiService;
  let mockElement;
  beforeEach(() => {
    mockApiService = {
      getUser: jest.fn(),
    };
    mockElement = {
      textContent: "",
    };
  });
  test("should display the user name when fetching succeeds", async () => {
    const userId = 1;
    const mockUser = { name: "Alice" };
    mockApiService.getUser.mockResolvedValue(mockUser);
    await fetchAndDisplayUser(mockApiService, userId, mockElement);
    expect(mockApiService.getUser).toHaveBeenCalledWith(userId);
    expect(mockElement.textContent).toBe("Hello, Alice");
  });
  test("should display an error message when the user has invalid data", async () => {
    const userId = 2;
    const mockUser = {};
    mockApiService.getUser.mockResolvedValue(mockUser);
    await fetchAndDisplayUser(mockApiService, userId, mockElement);
    expect(mockApiService.getUser).toHaveBeenCalledWith(userId);
    expect(mockElement.textContent).toBe("Invalid user data");
  });
  test("should display an error message when the API call fails", async () => {
    const userId = 3;
    mockApiService.getUser.mockRejectedValue(new Error("User not found"));
    await fetchAndDisplayUser(mockApiService, userId, mockElement);
    expect(mockApiService.getUser).toHaveBeenCalledWith(userId);
    expect(mockElement.textContent).toBe("User not found");
  });
});
```



```
● amogh@amogh-ubuntu:~/Programming/testing/m8$ npm test

  > m8@1.0.0 test
  > jest

 PASS  tests/notificationUtils.test.js
  sendNotification
    ✓ should return "Notification Sent" when notificationService.send returns true (10 ms)
    ✓ should return "Failed to Send" when notificationService.send returns false (1 ms)

 PASS  tests/domUtils.test.js
  toggleVisibility
    ✓ should set display to "none" when initially visible (7 ms)
    ✓ should set display to "block" when initially hidden (1 ms)
    ✓ should correctly toggle multiple times (1 ms)

 PASS  tests/asyncUtils.test.js
  delayedGreeting
    ✓ should resolve with the correct greeting message (15 ms)
    ✓ should respect the specified delay (5 ms)

 PASS  tests/arrayUtils.test.js
  getElement
    ✓ should return the correct element for a valid index (11 ms)
    ✓ should throw an error for negative indices (15 ms)
    ✓ should throw an error for indices greater than or equal to the array length (4 ms)
    ✓ should throw an error if the array is empty (2 ms)

 PASS  tests/userUtils.test.js
  fetchAndDisplayUser
    ✓ should display the user name when fetching succeeds (16 ms)
    ✓ should display an error message when the user has invalid data (2 ms)
    ✓ should display an error message when the API call fails (2 ms)

 PASS  tests/stringUtils.test.js
  capitalize
    ✓ should capitalize the first letter of a word (13 ms)
    ✓ should return an empty string if input is an empty string (1 ms)
    ✓ should handle single-character words correctly (1 ms)
    ✓ should not modify the rest of the word (2 ms)
    ✓ should handle non-string inputs gracefully (if not sanitized) (2 ms)
  reverseString
    ✓ should reverse a normal string (1 ms)
    ✓ should handle empty strings (1 ms)
    ✓ should handle single-character strings (1 ms)
    ✓ should reverse a palindrome correctly (1 ms)
    ✓ should handle strings with spaces and special characters (1 ms)

 Test Suites: 6 passed, 6 total
 Tests:       24 passed, 24 total
 Snapshots:   0 total
 Time:        1.771 s
 Ran all test suites.
amogh@amogh-ubuntu:~/Programming/testing/m8$
```