JavaScript Unit Testing Expertise

Objective: To evaluate the candidate's ability to write and implement unit tests for JavaScript functions using a testing framework like Jest or Mocha.

Problem Statement: You are required to design and implement unit tests for a set of JavaScript functions that simulate common operations in a web application. The task focuses on testing individual functions for correctness, edge cases, and error handling.

Task Details:

1. **Functions to Test**
   - Implement unit tests for the following functions:
     - **`calculateDiscount(price, discount)`**: Calculates the final price after applying a discount percentage.
       - Input: `price (number)`, `discount (number)`
       - Output: Final price (number).

```
function calculateDiscount(price, discount) {
    if (price < 0 || discount < 0 || discount > 100) {
        throw new Error('Invalid input');
    }
    return price - (price * discount / 100);
}
```

     - **`filterProducts(products, query)`**: Filters a list of products by name based on a search query.
       - Input: `products (array of objects with name and price)`, `query (string)`
       - Output: Filtered array of products.

```
function filterProducts(products, query) {
        if (!Array.isArray(products) || typeof query !==
    'string') {
            throw new Error('Invalid input');
        }
        return products.filter(product =>
    product.name.toLowerCase().includes(query.toLowerCase()))
    ;
    }
```

     - **`sortProducts(products, key)`**: Sorts a list of products by a specified key (`name` or `price`).
       - Input: `products (array of objects with name and price)`, `key (string)`
       - Output: Sorted array of products.

```
function sortProducts(products, key) {
    if (!Array.isArray(products) || (key !== 'name' &&
key !== 'price')) {
        throw new Error('Invalid input');
    }
    return products.sort((a, b) => {
        if (key === 'price') {
            return a.price - b.price;
        } else {
            return a.name.localeCompare(b.name);
        }
    });
}
```

- **`validateEmail(email)`**: Validates an email address format.
  - Input: `email (string)`
  - Output: Boolean (`true` if valid, `false` otherwise).

```
function validateEmail(email) {
    if (typeof email !== 'string') {
        throw new Error('Invalid input');
    }
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailRegex.test(email);
}
```

2. **Testing Requirements**
   - Cover both positive and negative test cases.
   - Use a testing framework like Jest, Mocha, or Jasmine.
   - Mock any dependencies or external calls if needed.
3. **Reporting**
   - Generate a test coverage report.
   - Highlight any failing tests with detailed error logs.
4. **Bonus Tasks**
   - Parameterize tests to handle multiple inputs dynamically.
   - Test asynchronous functions by adding a mocked API call and writing corresponding tests.
   - Integrate the test suite with a CI/CD pipeline (e.g., GitHub Actions).

Submission Guidelines:

1. Provide the source code in a GitHub repository.
2. Include a README file with:
   - Steps to execute the tests.
   - Frameworks and tools used.
   - Any assumptions or constraints.
3. Attach the test coverage report as part of the submission.

Evaluation Criteria:

- **Coverage:** Completeness of test cases and edge case handling.
- **Code Quality:** Adherence to best practices and readability.
- **Testing:** Effectiveness of the unit tests and coverage achieved.
- **Bonus Points:** Successfully completing bonus tasks.

Hints and Tips:

- Use mock data to simulate input for the functions.
- Structure tests to be reusable and modular.
- Document assumptions and constraints clearly in the README.

Good luck!