Test report for the application: an E-commerce store

Software testing COMP.SE.200

Teemu Ruonakoski 152116224

Aleksi Kehusmaa 150323035

GitHub: https://github.com/tcteru/software_testing.git

Coveralls: https://coveralls.io/github/tcteru/software_testing

Table of contents

1. Introduction 4
2. Implementation of CI-pipeline and tests 4
   2.1 Tests locally 4
   2.2 Tests using CI-pipeline 5
   2.3 Results of self- and AI-implemented tests 5
   2.4 Comparison of self- and AI-implemented tests 7
3. Findings and conclusions 7
   3.1 Bugs found during the test implementation 7
   3.2 Quality evaluation 8
   3.3 Test coverage estimates 8
4. AI and testing 9
   4.1 Pros and cons 9
   4.2 How we followed our plan to use AI 9
   4.3 AI usage in this work 9
5. Course feedback & reflection 9
   5.1 Reflection Teemu 9
   5.2 Reflection Aleksi 10
6. References 10
7. Appendix 11

Definitions, acronyms and abbreviations

AI	artificial intelligence
CI	continuous integration
E2E	end-to-end
Jest	JavaScript testing framework used for our tests
LCOV	Coverage format generated by Jest
Coveralls	Online service to visualize coverage reports
PR	Pull Request
src	source code

1. Introduction

This document is a report on the implementation of the plan in Appendix Figure 2: Test plan and presents how the testing was carried out and what observations were made based on the tests. The test plan covered testing the entire e-commerce application, but this document focuses on the implementation of unit tests for the functions selected in the plan from the provided utility library.

Chapter 2 of the document discusses the tests, their implementation method, the local test environment and using continuous integration (CI) pipeline. The chapter describes how the reader can run the tests we have done if they wish. The chapter also examines how the tests planned in the plan compare to the tests created by artificial intelligence (AI) for the same functions. Chapter 3 examines the results of the tests. The chapter includes bug reports. Based on the tests carried out, it is assessed whether the function library used is suitable for implementing the e-commerce store and whether its functions can be trusted. The test coverage is also examined. Chapter 4 focuses on combining AI and testing. It examines how AI was utilized or how it would have been worth utilizing it. Chapter 5 contains feedback and reflections on the course within which the testing of the e-commerce store application was carried out.

The purpose of this document is to provide a comprehensive description of how unit tests can be performed, what can be concluded from them, and how they should be implemented. The document also provides indications of how AI testing is impacting the testing industry. This therefore serves as a report document on the implemented e-commerce testing.

2. Implementation of CI-pipeline and tests

The test plan presented end-to-end (E2E) scenarios, based on which the testing was planned. Based on these, the tools to be used and the environments in which the tests were performed were selected. However, due to resource constraints, testing of the E2E scenarios was limited to unit testing of ten utility library [1] functions. Therefore, the E2E scenario test environment Cypress [2], which was selected in the plan, was excluded from the implementation presented in this document.

Test preparations were carried out as follows. To perform and document the testing, a public GitHub [3] repository was created to store the files and manage changes made to the files. Next, the utility library was copied into local files, after which the files were pushed to the created repository.

2.1 Tests locally

After this, the files were prepared locally for testing using the Visual Studio Code [4] application as follows. A folder was created for the tests, and files for each function were created there with the name <function name>.test.js. In addition, the package.json file was modified to comply with Appendix Figure 3: The contents of the file package.json. So, the Jest testing framework [5] and lines for measuring test coverage are added to the file. In addition, functions that are not tested were removed from the coverage measurement. This step required some investigation and did not proceed completely straight forward due to confusion.

At this stage, tests could be started to be written to the files and run locally. Locally, tests are run in the file folder using terminal commands “npm install” and “npm test”. In this case, the test results and coverages are displayed in the terminal.

2.2 Tests using CI-pipeline

Once it was possible to run tests locally, a CI-pipeline could be implemented using GitHub actions [6]. It runs tests automatically whenever changes are pushed to the GitHub repository. In the repository tab "Actions" you can create a new workflow. Select the "Node.js" workflow and edit the content according to Appendix Figure 4: The contents of the workflow file coverage.yml.

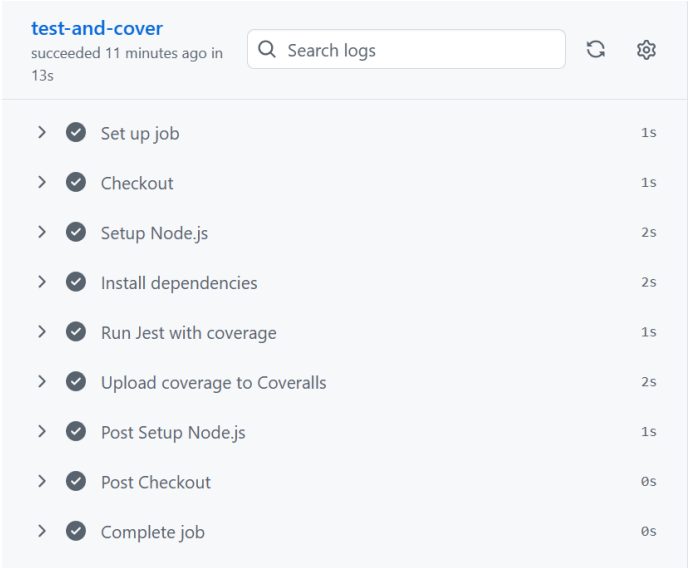


Figure 1: A working workflow in GitHub action

This step required a lot of familiarization and repeated experimentation due to the problems, confusion and work of using GitHub to get the workflow working. Without the line "continue-on-error: true" the workflow will fail if an error is found, so adding it allowed us to move forward. A working workflow progression is shown in figure Figure 1: A working workflow in GitHub action. So every time new changes are pushed the workflow runs the tests and the coverage results are sent to Coveralls [7], where they are visualized.

2.3 Results of self- and AI-implemented tests

The test plan includes tests for the functions "divide.js" and "filter.js". The tests were first carried out as planned. Self-designed test codes are in Appendix Figure 5: The contents of the file divide.test.js, Figure 6: The contents of the file filter.test.js and results of those who failed are in Table 1: Results of failed tests from file divide.test.js, Table 2: Results of failed tests from file filter.test.js.

Table 1: Results of failed tests from file divide.test.js

situation type	dividend (param)	divider(param)	expected return	received return
positive numbers	6	2	3	1
negative numbers	-6	2	-3	1
dividend zero	0	5	0	1
decimal values	0.06	0.02	3	1
string dividend	“a”	2	isNaN = true	isNaN = false

Table 2: Results of failed tests from file filter.test.js

situation type	array (param)	predicate (param)	expected return	received return
normal	[{'item':'meat', 'sold':false} {'item': 'potato', 'sold': 'true'}]	({ sold }) => sold	[“potato”]	{”item”: "potato", "sold": "true"}
predicate empty	[{'item':'meat', 'sold':false} {'item': 'potato', 'sold': 'true'}]	({}) => ()	toThrow()	[[]]

For these two functions for comparison, we asked AI (Microsoft 365 Copilot bizchat.20251202.48.2) to write tests. The copilot was given the command: “Here is function <function name> to be tested. Write unit tests for it and provide the code to copy to the file <function>.test.js. The function <function name>: <function contents>“. Also, filter.js command had to include “no jest.fn()” so that tests would work without problems. Codes given by AI are in Appendix Figure 7: The contents of the file divide.test.js that are implemented by AI and Figure 8: The contents of the file filter.test.js that are implemented by AI. Failed results of the AI implemented tests are in Table 3: Results of failed tests from AI implemented file divide.test.js and Table 4: Results of failed tests from AI implemented file filter.test.js.

Table 3: Results of failed tests from AI implemented file divide.test.js

situation type	dividend (param)	divider (param)	expected return	received return
positive number	6	4	1.5	1
negative dividend	-6	3	-2	1
negative divisor	6	-3	-2	1
both negative	-8	-2	4	1
zero dividend	0	5	0	1
zero divisor	5	0	infinity	NaN
floating point	0.3	0.1	close to 3	1
large numbers	1e12	1e6	1000000	1
the result zero sign is the same as the parameter zero sign	+0	2	is(poszero, 0) = true	is(poszero, 0) = false

Table 4: Results of failed tests from AI implemented file filter.test.js

situation type	array (param)	predicate (param)	expected return	received return
no match	[1, 2, 3]	() => false	[]	[[]]
empty array	[]	() =>{ called += 1; return true;}	[] called = 0	[[]]
undefined array	null	() =>{ called += 1; return true;}	[] called = 0	[[]]

The self-designed tests were run at an earlier stage than the AI implemented tests were added, so they can be found as is in the GitHub history. This is because the function tests were modified to deviate from the plan to increase test coverage before the pure AI implementation was added. Test results are in Appendix Figure 9: Total test results when coverage is 100% and Figure 10: Test results for self-planned tests. Coverage results can be seen on coveralls.io (first page link).

2.4 Comparison of self- and AI-implemented tests

Based on the above results, it can be stated that self-designed tests mainly test the basic operation of the function, but there were no specific test cases. AI differs in that it implements detailed tests, such as positive and negative plus, which is not necessarily necessary. Some AI tests are also complex to implement. It is also noted that self-designed tests are better at checking so-called incorrect inputs, while AI tests only intended inputs. AI can quite reliably implement almost any kind of test it wants, if it gives precise commands. However, by self-designing the tests, all necessary tests are done and on the other hand the number of tests does not get too high, as AI easily does.

AI mostly implements tests that they have a expectation that the test would pass. That was a problem whit these buggy src [1] files in some cases. These tests were good to notice these bugs since they had an expected result. AI generated quickly very throughout tests that test a lot of different cases in given test suites. AI can generate quickly a lot of different inputs that test different strings and objects etc. However, there are a few downsides, since AI might sometimes forget our choices that were made in part 1 of this assignment, so therefore the AI generated tests might include some unnecessary test cases. So, in a lot of cases a human eye is required. So, the upsides for self-implemented tests are the prioritisation on which tests are critical for this given e-commerce online store, and our part 1 of this assignment. So, our quality control was a bit better. The effectiveness of the AI with its pace to generate these tests was very impressive, but always a last check and some little fixes were needed to be made ourselves. It can therefore be stated that, with a little guidance, artificial intelligence is a very effective and suitable tool for testing.

3. Findings and conclusions

3.1 Bugs found during the test implementation

During our unit testing of the chosen ten utility files, we found bugs on at least two of the files in Divide and Filter as shown in big reports in Table 5: filter.js and divide.js bug reports. These functions were the subject of the most critical check. Our thought process on this was to firstly implement the tests as planned, so we would get the report of the fails, and then also tried to adapt the tests so they could pass without any fails. Figure 111: Divide.test.js for 0 failed tests is the divide adapted to the buggy src file, and also Figure 122: Filter.test.js for 0 failed tests is the filter adapted to the buggy src [1] file.

Table 5: filter.js and divide.js bug reports

ID:	0001	0002
name:	Divide_1	Filter_array
day found:	7.12.2025	7.12.2025
file:	divide.js	filter.js
definition:	Divide function returns 1 although the result can be calculated	Filter returns the item with all its information even though it should only return the items in the array

Environment	Windows 11 running Jest	Windows 11 running Jest
Repriduce:	Call the function with parameter numbers divide(4,2), for example.	Call the function with array and predicator parameters, the result of which is not empty, and examine the format of the result. [{'item':'meat', 'sold':false} {'item': 'potato', 'sold': 'true'}]
Exceptet result:	2	[‘potato’]
Actual result:	1	{'item': 'potato', 'sold': 'true'}
Error severity:	very serious	quite small
Priority to fix:	High	Low

The problem with many tests was that they didn't know what the result should be due to incomplete function documentation. Therefore, tests were performed with some assumption, and when it was seen that the result was different but logical, the expected result of the test was changed.

3.2 Quality evaluation

The provided utility [1] library contains a mix of correct implementations, and also a few severe bugs that caused problems during the testing. These bugs are very harmful, and so before fixing those src [1] files we do not recommend the e-commerce online store to make any changes to their website, and also our test report shouldn’t be given forward to them. This is because the tested functions contain high-risk errors, such as an incorrect result for divide, which could lead to financial losses, for example, as stated in the test plan.

Before the library can be used, all its functions must be tested in a similar way, critical errors must be fixed, and documentation must be improved. Currently, the library does not meet the quality requirements.

3.3 Test coverage estimates

We ran our tests and pushed the coverage to Coveralls [7]. During the testing process we multiple times wrote more adapted tests to increase the coverages of our tests. We go every single one of them to give 100 coverages in everything so that we’ll say was a big success. Full coverage is not necessarily a viable goal in testing in general, as 90% is already comprehensive. However, the purpose of this testing was to get a quality picture of the entire library content based on ten functions, so full test coverage was easy to implement here and then you get a better overall picture of the library.

For additional testing that wasn’t a part of this work we would recommend to do stress tests to test how the e-commerce online store handles operating beyond its normal limits. Also adding some integration tests would be helpful so that the integration between the front-end and the back-end would be still covered as well as possible. This is because correcting and changing library functions also affects other test areas.

4. AI and testing

4.1 Pros and cons

AI was mainly used in working with the buggy divide and filter tests. So we would get the idea on how to deal with those bugs since the src [1] files weren't meant to be modified. Also for us in the beginning we had some problems to get the repository to work for both of us so there it was helpful to get some instructions on how to set up the repository. Also, for a while we had some issues with the GitHub Actions [6] on giving failed runs, and the AI wasn't very helpful on that so in the end we had to set it up ourselves to get it working.

4.2 How we followed our plan to use AI

We planned to use AI to propose unit tests for some functions, this was done, however obviously we found that they had to me modified quite a bit so in the end product you can see quite a lot of manual coding as it should be. Mainly AI gave us lots of new perspectives on how to make our planned unit tests and also gave us some useful information about the JavaScript syntax since this was a relatively new programming language for both of us.

4.3 AI usage in this work

Application:	**Version:**
Open AI ChatGPT	GPT 5.1
Microsoft 365 Copilot	bizchat.20251202.48.2
Intended use:	**Areas where used:**
Troubleshooting and fixing	coverage.yml package.json
Writing tests	divide.test.js filter.test.js
JavaScript syntax	General coding

5. Course feedback & reflection

5.1 Reflection Teemu

In my opinion the weekly exercises were very helpful towards this second part of the assignment, since there were quite a bit of different test cases taught and that was indeed very useful. In this assignment there was a quite a lot of work, but luckily there was also a long time to do the both parts of this assignment. Only downside I would say was that the deadline for this second part is during the exam week, but luckily there was time to do a lot of this before those exams. A lot of new things were learned about software testing, but luckily in previous programming courses there were some testing taught, so I had at least some background towards this assignment. Probably the hardest parts were setting up the Github since this was a first course for me where this was used since on the other courses Gitlab was used instead. Also we had some problems during the testing but nothing too major and writing these reports isn't very easy.

5.2 Reflection Aleksi

Overall, the course felt heavy compared to an average course. The total number of hours in the course was high. However, the course was very educational and the content was necessary in every way and I feel that software testing is now familiar to me in some way.

The lectures contained important material, but they covered a lot of things that were not so-called learnable information, so those things that were learnable remained among the text mass. The information to be learned should somehow be presented more clearly. The guest lectures were very interesting and useful. The weekly exercises were the most important in terms of learning. They went through things that really had to be learned. They made it much easier to understand. This assignment was educational, but confusing. The instructions should be clarified more clearly so that it would be clear how extensive the work is, what should be taken into account in it and what not. The confusion is also visible in our work because sometimes it was unclear what we were doing and what we were not doing. However, this work was really educational and it shows in the progress of this work. If I were to do this again, the end result would be much higher quality.

In general, the challenge in the first exercise was that I was using a programming language that was unfamiliar to me, and that I had not encountered in programming courses, as some others had. This caused a slight setback right at the beginning.

6. References

[1] otula, “Utility library COMP.SE.200-2024-2025-1,” [Online]. Available: https://github.com/otula/COMP.SE.200-2024-2025-1?tab=readme-ov-file.

[2] “Browser testing for modern teams,” Cypress, [Online]. Available: https://www.cypress.io/.

[3] “About GitHub and Git,” GitHub Docs, [Online]. Available: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

[4] “The open source AI code editor,” Visual Studio Code, [Online]. Available: https://code.visualstudio.com/.

[5] “Jest is a delightful JavaScript Testing Framework with a focus on simplicity,” Jest, [Online]. Available: https://jestjs.io/.

[6] “Automate your workflow from idea to production,” GitHub, [Online]. Available: https://github.com/features/actions.

[7] “Deliver better code,” Coveralls, [Online]. Available: https://coveralls.io/.

7. Appendix

Test plan

Figure 2: Test plan

{} package.json > ...

```
{
  "name": "software-testing-assignment",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  ▷Debug
  "scripts": {
    "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js",
    "test:coverage": "node --experimental-vm-modules node_modules/jest/bin/jest.js --coverage",
    "coverage:merge": "lcov-result-merger \"coverage/**/*.lcov.info\" coverage/merged.lcov"
  },
  "jest": {
    "collectCoverage": true,
    "collectCoverageFrom":
    ["src/**.js",
    "!src/at.js",
    "!src/camelCase.js",
    "!src/castArray.js",
    "!src/ceil.js",
    "!src/chunk.js",
    "!src/countBy.js",
    "!src/defaultTo.js",
    "!src/defaultToAny.js",
    "!src/drop.js",
    "!src/endsWith.js",
    "!src/eq.js",
    "!src/every.js",
    "!src/get.js",
    "!src/isArguments.js",
    "!src/isArrayLike.js",
    "!src/isArrayLikeObject.js",
    "!src/isBoolean.js",
    "!src/isBuffer.js",
    "!src/isDate.js",
    "!src/isEmpty.js",
    "!src/isLength.js",
    "!src/isObject.js",
    "!src/isObjectLike.js",
    "!src/isSymbol.js",
    "!src/isTypedArray.js",
    "!src/keys.js",
    "!src/map.js",
    "!src/memoize.js",
    "!src/slice.js",
    "!src/toFinite.js",
    "!src/toInteger.js",
    "!src/toString.js",
    "!src/words.js"
  ],
    "coverageDirectory": "coverage/jest",
    "coverageReporters": ["lcov", "text", "html"],
    "testEnvironment": "node",
    "testMatch": ["**/test/**/*.test.js"],
    "transform": {}

  },
  "devDependencies": {
    "jest": "^29.7.0",
    "lcov-result-merger": "^3.3.0"
  }

}
```

Figure 3: The contents of the file package.json

.github > workflows > ! coverage.yml

```yaml
name: Coverage

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  test-and-cover:
    runs-on: ubuntu-latest

    permissions:
      contents: read
      pull-requests: write

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: "20.x"
          cache: "npm"

      - name: Install dependencies
        run: npm install

      - name: Run Jest with coverage
        run: npm run test:coverage
        continue-on-error: true

      - name: Upload coverage to Coveralls
        uses: coverallsapp/github-action@v2
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          path-to-lcov: coverage/jest/lcov.info

```

Figure 4: The contents of the workflow file coverage.yml

```javascript
describe("divide (based on the phase 1 plan)", () => {
  test("divides positive numbers", () => {
    expect(divide(6, 2)).toBe(3);
  });

  test("divides negative numbers", () => {
    expect(divide(-6, 2)).toBe(-3);
    expect(divide(6, -2)).toBe(-3);
    expect(divide(-6, -2)).toBe(3);
  });

  test("division by zero yields Infinity/-Infinity (JS behavior)", () => {
    expect(Number.isNaN(divide(-6, 0))).toBe(true);
  });

  test("zero divided by non-zero is zero", () => {
    expect(divide(0, 5)).toBe(0);
  });

  test("desimal values", () => {
    expect(divide(0.06, 0.02)).toBe(3);
    expect(divide(1, 3)).toBe(1/3);
  });

  test("non-numeric inputs produce NaN", () => {
    expect(Number.isNaN(divide("a", 2))).toBe(true);
    expect(Number.isNaN(divide(2, "a"))).toBe(true);
    expect(Number.isNaN(divide({}, 2))).toBe(true);
  });
});
```

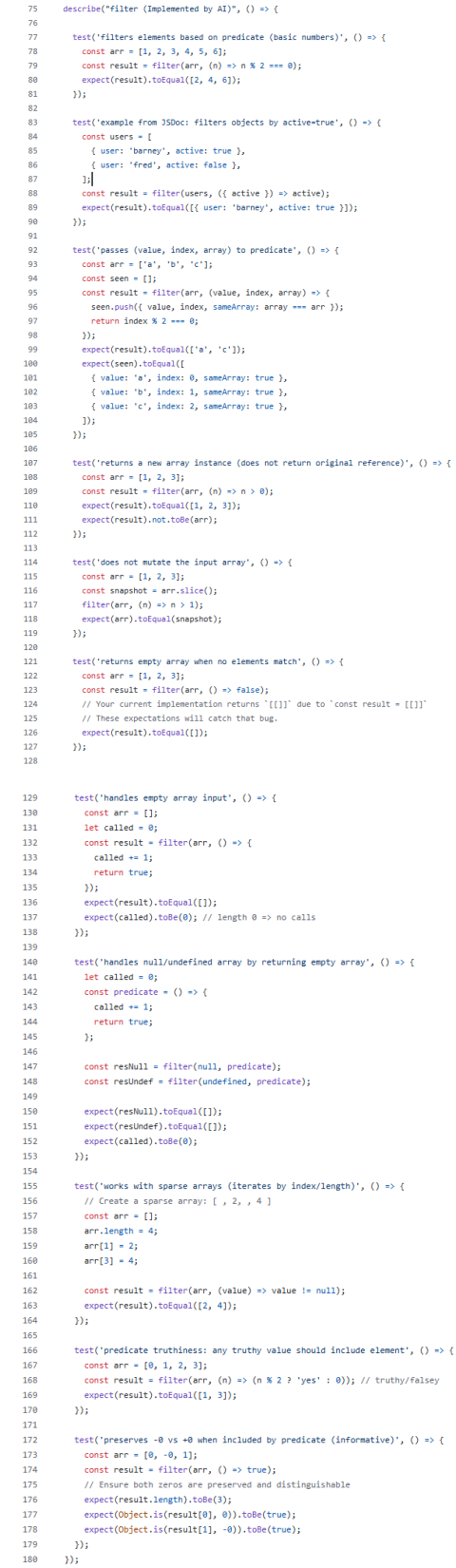
Figure 5: The contents of the file divide.test.js that are self-designed

```
3     describe("filter (based on the phase 1 plan)", () => {
4
5       test("filters elements matching", () => {
6         expect(filter([
7                       {'item':'meat', 'sold':false},
8                       {'item': 'potato', 'sold': 'true'}
9                     ],
10                     ({sold})=>sold)).toEqual(['potato']);
11      });
12
13      test("empty array", () => {
14        expect(filter([],
15                     ({sold})=>sold)).toEqual([[]]);
16      });
17
18      test("predicate empty", () => {
19        expect(filter([
20                    {'item':'meat', 'sold':false},
21                    {'item': 'potato', 'sold': 'true'}
22                  ],
23                     ({})=>{})).toThrow();
24      });
25
26      test("functional content not in array", () => {
27        expect(filter([
28                    {'item':'meat', 'sold':false},
29                    {'item': 'potato', 'sold': 'true'}
30                  ],
31                     ({reserved})=>reserved)).toEqual([[]]);
32      });
33    });
```

Figure 6: The contents of the file filter.test.js that are self-designed

```
34  describe("divide (Implemented by AI)", () => {
35    test('divides positive numbers (example from JSDoc: divide(6, 4) => 1.5)', () => {
36      expect(divide(6, 4)).toBe(1.5);
37    });
38
39    test('handles negative dividend', () => {
40      expect(divide(-6, 3)).toBe(-2);
41    });
42
43    test('handles negative divisor', () => {
44      expect(divide(6, -3)).toBe(-2);
45    });
46
47    test('handles both negative (result positive)', () => {
48      expect(divide(-8, -2)).toBe(4);
49    });
50
51    test('zero dividend yields 0 (when divisor != 0)', () => {
52      expect(divide(0, 5)).toBe(0);
53      expect(divide(0, -7)).toBe(0);
54    });
55
56    test('division by zero yields +/-Infinity depending on sign', () => {
57      expect(divide(5, 0)).toBe(Infinity);
58      expect(divide(-5, 0)).toBe(-Infinity);
59    });
60
61    test('0 / 0 yields NaN', () => {
62      expect(Number.isNaN(divide(0, 0))).toBe(true);
63    });
64
65    test('handles floating point inputs using toBeCloseTo', () => {
66      expect(divide(0.3, 0.1)).toBeCloseTo(3, 10);
67      expect(divide(1, 3)).toBeCloseTo(0.3333333333, 10);
68    });
69
70    test('handles large numbers', () => {
71      expect(divide(1e12, 1e6)).toBe(1e6);
72    });
73
74    test('preserves sign of zero in IEEE-754 terms (optional, informative)', () => {
75      // +0 / positive -> +0
76      const posZero = divide(0, 2);
77      expect(Object.is(posZero, 0)).toBe(true);
78
79      // -0 / positive -> -0
80      const negZero = divide(-0, 2);
81      expect(Object.is(negZero, -0)).toBe(true);
82    });
83  });
84
```

Figure 7: The contents of the file divide.test.js that are implemented by AI



```
describe("filter (Implemented by AI)", () => {

  test('filters elements based on predicate (basic numbers)', () => {
    const arr = [1, 2, 3, 4, 5, 6];
    const result = filter(arr, (n) => n % 2 === 0);
    expect(result).toEqual([2, 4, 6]);
  });

  test('example from JSDoc: filters objects by active=true', () => {
    const users = [
      { user: 'barney', active: true },
      { user: 'fred', active: false },
    ];
    const result = filter(users, ({ active }) => active);
    expect(result).toEqual([{ user: 'barney', active: true }]);
  });

  test('passes (value, index, array) to predicate', () => {
    const arr = ['a', 'b', 'c'];
    const seen = [];
    const result = filter(arr, (value, index, array) => {
      seen.push({ value, index, sameArray: array === arr });
      return index % 2 === 0;
    });
    expect(result).toEqual(['a', 'c']);
    expect(seen).toEqual([
      { value: 'a', index: 0, sameArray: true },
      { value: 'b', index: 1, sameArray: true },
      { value: 'c', index: 2, sameArray: true },
    ]);
  });

  test('returns a new array instance (does not return original reference)', () => {
    const arr = [1, 2, 3];
    const result = filter(arr, (n) => n > 0);
    expect(result).toEqual([1, 2, 3]);
    expect(result).not.toBe(arr);
  });

  test('does not mutate the input array', () => {
    const arr = [1, 2, 3];
    const snapshot = arr.slice();
    filter(arr, (n) => n > 1);
    expect(arr).toEqual(snapshot);
  });

  test('returns empty array when no elements match', () => {
    const arr = [1, 2, 3];
    const result = filter(arr, () => false);
    // Your current implementation returns `[[]]` due to `const result = [[]]`
    // These expectations will catch that bug.
    expect(result).toEqual([]);
  });



  test('handles empty array input', () => {
    const arr = [];
    let called = 0;
    const result = filter(arr, () => {
      called += 1;
      return true;
    });
    expect(result).toEqual([]);
    expect(called).toBe(0); // length 0 => no calls
  });

  test('handles null/undefined array by returning empty array', () => {
    let called = 0;
    const predicate = () => {
      called += 1;
      return true;
    };

    const resNull = filter(null, predicate);
    const resUndef = filter(undefined, predicate);

    expect(resNull).toEqual([]);
    expect(resUndef).toEqual([]);
    expect(called).toBe(0);
  });

  test('works with sparse arrays (iterates by index/length)', () => {
    // Create a sparse array: [ , 2, , 4 ]
    const arr = [];
    arr.length = 4;
    arr[1] = 2;
    arr[3] = 4;

    const result = filter(arr, (value) => value != null);
    expect(result).toEqual([2, 4]);
  });

  test('predicate truthiness: any truthy value should include element', () => {
    const arr = [0, 1, 2, 3];
    const result = filter(arr, (n) => (n % 2 ? 'yes' : 0)); // truthy/falsey
    expect(result).toEqual([1, 3]);
  });

  test('preserves -0 vs +0 when included by predicate (informative)', () => {
    const arr = [0, -0, 1];
    const result = filter(arr, () => true);
    // Ensure both zeros are preserved and distinguishable
    expect(result.length).toBe(3);
    expect(Object.is(result[0], 0)).toBe(true);
    expect(Object.is(result[1], -0)).toBe(true);
  });
});
```

Figure 8: The contents of the file filter.test.js that are implemented by AI

Figure 9: Total test results when coverage is 100%

Figure 10: Test results for self-planned tests

```
import divide from "../src/divide.js";

describe("divide (aligned to current implementation)", () => {

  test("returns 1 when divisor is a non-zero number", () => {
    expect(divide(6, 2)).toBe(1);
    expect(divide(-6, 2)).toBe(1);
    expect(divide(6, -2)).toBe(1);
    expect(divide(-6, -2)).toBe(1);
    expect(divide(0.06, 0.02)).toBe(1);
    expect(divide(1, 3)).toBe(1);
    expect(divide(0.3, 0.1)).toBe(1);
    expect(divide(1e12, 1e6)).toBe(1);
  });

  test("zero dividend with non-zero divisor still returns 1", () => {
    expect(divide(0, 5)).toBe(1);
    expect(divide(0, -7)).toBe(1);
  });

  test("division by zero yields NaN (divisor/divisor => 0/0)", () => {
    expect(Number.isNaN(divide(5, 0))).toBe(true);
    expect(Number.isNaN(divide(-5, 0))).toBe(true);
    expect(Number.isNaN(divide(0, 0))).toBe(true);
  });

  test("string args: trigger string path, operator uses divisor/divisor", () => {
    expect(divide('4', '2')).toBe(1);
    expect(divide('6', 3)).toBe(1);
    expect(divide(6, '3')).toBe(1);
  });

  test("non-numeric string behavior depends on which position is string", () => {
    expect(divide('a', 2)).toBe(1);
    expect(Number.isNaN(divide(2, 'a'))).toBe(true);
  });

  test("object numeric path behavior", () => {
    expect(divide({}, 2)).toBe(1);
    expect(Number.isNaN(divide(2, {}))).toBe(true);
  });

  test("undefined handling from createMathOperation", () => {
    expect(divide(undefined, undefined)).toBe(1);
    expect(divide(5, undefined)).toBe(5);
    expect(divide(undefined, 7)).toBe(7);
  });

  test("IEEE-754 zero sign is not preserved by current operator", () => {
    const posZero = divide(0, 2);
    const negZero = divide(-0, 2);
    expect(Object.is(posZero, 0)).toBe(false);
    expect(Object.is(negZero, -0)).toBe(false);
  });
});
```

Figure 111: Divide.test.js for 0 failed tests

```
import filter from "../src/filter.js";

describe('filter (aligned to current implementation)', () => {
  test('filters values where predicate is truthy', () => {
    const arr = [1, 2, 3, 4];
    const result = filter(arr, (v) => v % 2 === 0);
    expect(result).toEqual([2, 4]);
  });

  test('predicate receives (value, index, array)', () => {
    const arr = ['a', 'b', 'c'];
    const calls = [];
    const result = filter(arr, (v, i, a) => {
      calls.push([v, i, a]);
      return i < a.length - 1;
    });
    expect(result).toEqual(['a', 'b']);
    expect(calls).toEqual([
      ['a', 0, arr],
      ['b', 1, arr],
      ['c', 2, arr],
    ]);
  });

  test('does not mutate original array', () => {
    const arr = [1, 2, 3];
    const copy = [...arr];
    filter(arr, (v) => v > 1);
    expect(arr).toEqual(copy);
  });

  test('handles falsy values in input (truthy and falsy filters)', () => {
    const arr = [0, '', null, undefined, false, NaN, 'a', 1];

    const resultTruthy = filter(arr, (v) => v);
    expect(resultTruthy).toEqual(['a', 1]);

    const resultFalsy = filter(arr, (v) => !v || Number.isNaN(v));
    expect(resultFalsy).toEqual([0, '', null, undefined, false, NaN]);
  });

  test('predicate returning non-boolean values is coerced to boolean', () => {
    const arr = [1, 2, 3];
    const result = filter(arr, (v) => v - 2); // [-1, 0, 1] -> truthy for 1 and 3
    expect(result).toEqual([1, 3]);
  });

  test('sparse arrays: current implementation invokes predicate on holes', () => {
    const arr = [1, , 3, , 5];
    const calls = [];
    const result = filter(arr, (v, i) => {
      calls.push(i);
      return v % 2 === 1;
    });
    expect(result).toEqual([1, 3, 5]);
    expect(calls).toEqual([0, 1, 2, 3, 4]);
  });

  test('returns `[[]]` when no elements match', () => {
    const arr = [1, 3, 5];
    const result = filter(arr, (v) => v % 2 === 0);
    expect(result).toEqual([[]]);
  });

  test('handles empty array input (returns `[[]]`)', () => {
    const result = filter([], () => true);
    expect(result).toEqual([[]]);
  });

  test('null and undefined array inputs yield `[[]]` (current implementation)', () => {
    const predicate = () => true;
    const resNull = filter(null, predicate);
    const resUndef = filter(undefined, predicate);
    expect(resNull).toEqual([[]]);
    expect(resUndef).toEqual([[]]);
  });

  test('no thisArg support: passing context as 3rd arg does not bind `this`', () => {
    const arr = [1, 2, 3];
    const context = { threshold: 2 };
    function predicate(v) {
      return v > this.threshold;
    }
    expect(() => filter(arr, predicate, context)).toThrow(TypeError);
  });

  test('non-array inputs behavior', () => {
    expect(filter('string', () => true)).toEqual(['s', 't', 'r', 'i', 'n', 'g']);
    expect(filter(123, () => true)).toEqual([[]]);
  });
```

Figure 122: Filter.test.js for 0 failed tests