

Γεώργιος Μητράκης
1115 2014 00107

ΥΣ09: Τεχνολογία Λογισμικού



Frontend Testing

“Testing”

- Ο έλεγχος της εφαρμογής για το αν συμπεριφέρεται με τον επιθυμητό τρόπο.

Αυτοματοποιημένο testing

- Αποσφαλμάτωση
- Ευκρίνεια λαθών
- Εξοικονόμηση χρόνου
- Συνέπεια ελέγχων
- Διάσπαση περίπλοκων εξαρτήσεων
- Ασφαλέστερη ενσωμάτωση νέων χαρακτηριστικών
- Ποιοτικότερος κώδικας

Είδη ελέγχων

- Unit tests
 - απομονωμένοι έλεγχοι βασικών δομικών στοιχείων, (π.χ. το τι επιστρέφει μια συνάρτηση)
- Integration tests
 - έλεγχοι στοιχείων με εξαρτήσεις (π.χ. το τι επιστρέφει μια συνάρτηση που καλεί άλλες συναρτήσεις)
- End-to-End tests
 - έλεγχοι πλήρους ροής μιας ενέργειας (π.χ. σε μία φόρμα, η συμπλήρωση των πεδίων, η υποβολή και ο έλεγχος της επεξεργασίας της “απάντησης”)

Πόσα;

- Unit tests: ~70%
- Integration tests: ~20%
- E2E: ~10%

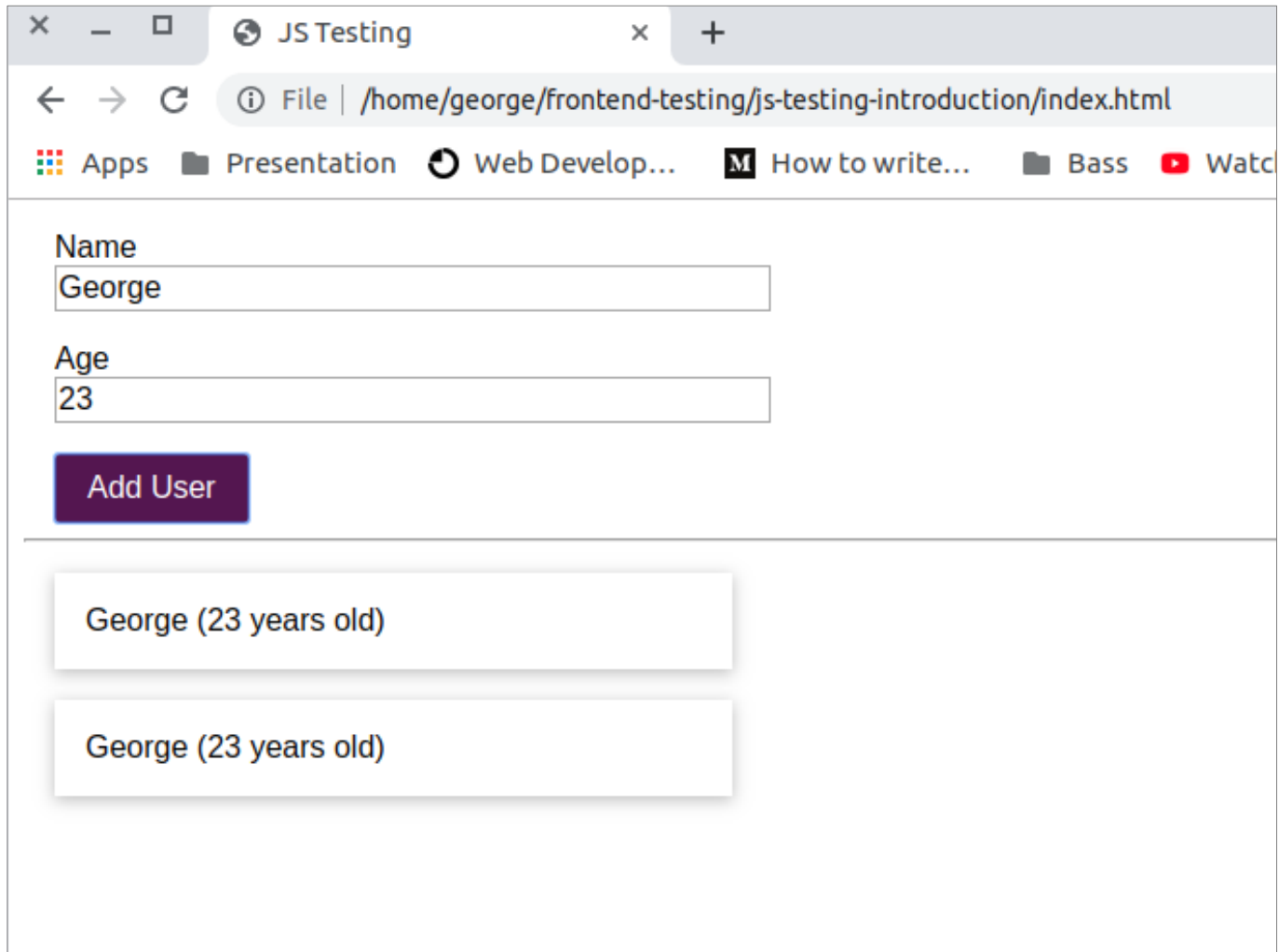
Setup

- Test Runner
- Assertion Library
- Headless Browser

Εργαλεία

- **Jest** (U+I), η πρόταση της React
- Mocha (Test Runner)
- Jasmine (U+I), η πρόταση της Angular
- **Puppeteer** (E2E)
- QUnit (U+I)
- Chai (Assertion Library)
- Selenium (E2E) + JUnit (U+I)

Παράδειγμα



A screenshot of a web browser window titled "JS Testing". The address bar shows the file path `/home/george/frontend-testing/js-testing-introduction/index.html`. The browser's tab bar includes "Apps", "Presentation", "Web Develop...", "How to write...", "Bass", and "Watch". The main content area contains a form with two input fields: "Name" with the value "George" and "Age" with the value "23". Below these fields is a purple "Add User" button. Underneath the button, there are two identical white boxes, each containing the text "George (23 years old)".

Name
George

Age
23

Add User

George (23 years old)

George (23 years old)

Παράδειγμα

The screenshot displays a VS Code editor with three open files: `app.js`, `main.js`, and `index.html`. The `EXPLORER` sidebar on the left shows the project structure, including `package-lock.json`, `package.json`, `styles.css`, and `util.js`.

app.js (lines 1-33):

```
1 const { generateText, createElement, validateInput } = require('./util');
2
3 const initApp = () => {
4   // Initializes the app, registers the button click listener
5   const newUserButton = document.querySelector('#btnAddUser');
6   newUserButton.addEventListener('click', addUser);
7 };
8
9 const addUser = () => {
10  // Fetches the user input, creates a new HTML element based on it
11  // and appends the element to the DOM
12  const newUserAgeInput = document.querySelector('input#age');
13  const newUserAgeInput = document.querySelector('input#age');
14
15  if (
16    !validateInput(newUserAgeInput.value, true, false) ||
17    !validateInput(newUserAgeInput.value, false, true)
18  ) {
19    return;
20  }
21
22  const userList = document.querySelector('.user-list');
23  const outputText = generateText(
24    newUserAgeInput.value,
25    newUserAgeInput.value
26  );
27  const element = createElement('li', outputText, 'user-item');
28  userList.appendChild(element);
29 };
30
31 // Start the app!
32 initApp();
33
```

main.js (lines 1-33):

```
1 const { generateText, createElement, validateInput } = require('./util');
2
3 const initApp = () => {
4   // Initializes the app, registers the button click listener
5   const newUserButton = document.querySelector('#btnAddUser');
6   newUserButton.addEventListener('click', addUser);
7 };
8
9 const addUser = () => {
10  // Fetches the user input, creates a new HTML element based on it
11  // and appends the element to the DOM
12  const newUserAgeInput = document.querySelector('input#age');
13  const newUserAgeInput = document.querySelector('input#age');
14
15  if (
16    !validateInput(newUserAgeInput.value, true, false) ||
17    !validateInput(newUserAgeInput.value, false, true)
18  ) {
19    return;
20  }
21
22  const userList = document.querySelector('.user-list');
23  const outputText = generateText(
24    newUserAgeInput.value,
25    newUserAgeInput.value
26  );
27  const element = createElement('li', outputText, 'user-item');
28  userList.appendChild(element);
29 };
30
31 // Start the app!
32 initApp();
33
```

index.html (lines 1-31):

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <link rel="stylesheet" href="styles.css">
9   <title>JS Testing</title>
10 </head>
11
12 <body>
13   <section class="control-panel">
14     <div class="input-container">
15       <label for="name">Name</label>
16       <input type="text" id="name">
17     </div>
18     <div class="input-container">
19       <label for="age">Age</label>
20       <input type="number" id="age">
21     </div>
22     <button id="btnAddUser" class="button">Add User</button>
23   </section>
24   <hr>
25   <section class="user-output">
26     <ul class="user-list"></ul>
27   </section>
28   <script src="dist/main.js"></script>
29 </body>
30
31 </html>
```

Terminal (bottom):

```
main.js 5.62 KiB main [emitted] main
Entrypoint main = main.js
[./app.js] 966 bytes {main} [built]
[./util.js] 644 bytes {main} [built]
```

Παράδειγμα

The screenshot shows a VS Code editor interface with the following components:

- EXPLORER:** Displays the file structure of the project. The 'JS' folder is expanded, showing files like `app.js`, `main.js`, `index.html`, and `util.js`.
- CODE EDITOR:** Contains two files:
 - `app.js`:

```
1 const { generateText, createElement, validateInput } = require('./util');
2
3 const initApp = () => {
4   // Initializes the app, registers the button click listener
5   const newUserButton = document.querySelector('#btnAddUser');
6   newUserButton.addEventListener('click', addUser);
7 };
8
9 const addUser = () => {
10  // Fetches the user input, creates a new HTML element based on it
11  // and appends the element to the DOM
12  const newUserNameInput = document.querySelector('input#name');
13  const newUserAgeInput = document.querySelector('input#age');
14
15  if (
16    !validateInput(newUserNameInput.value, true, false) ||
17    !validateInput(newUserAgeInput.value, false, true)
18  ) {
19    return;
20  }
21
22  const userList = document.querySelector('.user-list');
23  const outputText = generateText(
24    newUserNameInput.value,
25    newUserAgeInput.value
26  );
27  const element = createElement('li', outputText, 'user-item');
28  userList.appendChild(element);
29 };
30
31 // Start the app!
32 initApp();
33
```
 - `util.js`:

```
1 exports.generateText = (name, age) => {
2   // Returns output text
3   return `${name} (${age} years old)`;
4 };
5
6 exports.createElement = (type, text, className) => {
7   // Creates a new HTML element and returns it
8   const newElement = document.createElement(type);
9   newElement.classList.add(className);
10  newElement.textContent = text;
11  return newElement;
12 };
13
14 exports.validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
21   }
22   if (isNumber && +text === NaN) {
23     return false;
24   }
25   return true;
26 };
27
```
- TERMINAL:** Shows the output of the application:

```
main.js 5.62 KiB main [emitted] main
Entrypoint main = main.js
[./app.js] 966 bytes {main} [built]
[./util.js] 644 bytes {main} [built]
```

Unit Test

The screenshot displays the Visual Studio Code interface with a project named 'JS-TESTING-INTRODUCTI...'. The Explorer sidebar on the left shows the file structure, including 'util.js' and 'util.test.js'. The main editor area is split into two panes. The left pane shows the source code of 'util.js', which defines three functions: 'generateText', 'createElement', and 'validateInput'. The right pane shows the test file 'util.test.js', which uses Jest to test the 'generateText' function. A dropdown menu is open over the 'toBe' method, listing various Jest matchers. At the bottom, the 'TERMINAL' tab shows the test results, indicating that the test passed successfully.

```
JS util.js > ...
1 exports.generateText = (name, age) => {
2   // Returns output text
3   return `${name} (${age} years old)`;
4 };
5
6 exports.createElement = (type, text, className) => {
7   // Creates a new HTML element and returns it
8   const newElement = document.createElement(type);
9   newElement.classList.add(className);
10  newElement.textContent = text;
11  return newElement;
12 };
13
14 exports.validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
21   }
22   if (isNumber && +text === NaN) {
23     return false;
24   }
25 };
26
```

```
JS util.test.js > test('should output name and age') callback
1 //nodejs import syntax, compatible with jest:
2 const { generateText } = require('./util');
3
4
5 test(function defined by the Test Runner
6   'should output name and age',
7   () => {
8     const text = generateText('George', 23);
9
10    //function defined by the Assertion Library
11    expect(text).toBe('George (23 years old)');
12    expect(text).toBe
13  });
14
```

3: Task - test

PASS ./util.test.js
✓ should output name and age (1ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 0.355s, estimated 1s

Unit Test

The screenshot displays a VS Code workspace for a JavaScript project. The Explorer sidebar on the left shows the file structure, including `util.js`, `util.test.js`, and `package.json`. The main editor area is split into two panes. The left pane shows the source code of `util.js`, which defines three exported functions: `generateText`, `createElement`, and `validateInput`. The right pane shows the test code in `util.test.js`, which uses Jest to test the `generateText` function. The tests are: `should output name and age` (which passes) and `should output data-less text` (which fails). The Test Explorer at the bottom shows the failure details for the second test, indicating that the received value is `"George (23 years old)"` instead of the expected `"(null years old)"`.

```
JS util.js > generateText > exports.generateText
1 exports.generateText = (name, age) => {
2   // Returns output text
3   return `George (23 years old)`;
4 };
5
6 exports.createElement = (type, text, className) => {
7   // Creates a new HTML element and returns it
8   const newElement = document.createElement(type);
9   newElement.classList.add(className);
10  newElement.textContent = text;
11  return newElement;
12 };
13
14 exports.validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
21   }
22 }
```

```
JS util.test.js > ...
1 //nodejs import syntax, compatible with jest:
2 const { generateText } = require('./util');
3
4
5 test(function defined by the Test Runner
6   'should output name and age',
7   () => {
8     const text = generateText('George', 23);
9
10    //function defined by the Assertion Library
11    expect(text).toBe('George (23 years old)');
12  });
13
14
15 test('should output data-less text', () => {
16   const text = generateText('', null);
17   expect(text).toBe('(null years old)');
18 })
```

3: Task - test

FAIL ./util.test.js

- ✓ should output name and age (6ms)
- ✗ should output data-less text (6ms)

● **should output data-less text**

expect(received).toBe(expected) // Object.is equality

Expected: "(null years old)"

Received: "George (23 years old)"

Unit Test

FAIL ./util.test.js

- ✓ should output name and age (6ms)
- ✗ should output data-less text (6ms)

● **should output data-less text**

expect(received).toBe(expected) // Object.is equality

Expected: " (null years old)"

Received: "George (23 years old)"

```
15 | test('should output data-less text', () => {
16 |     const text = generateText('', null);
> 17 |     expect(text).toBe(' (null years old)');
    |                   ^
18 | })
```

at Object.toBe (util.test.js:17:18)

Test Suites: 1 failed, 1 total

Tests: 1 failed, 1 passed, 2 total

Snapshots: 0 total

Time: 0.307s, estimated 1s

Unit Test

The screenshot displays the VS Code interface with two editor windows and a terminal. The left window shows `util.js` with three exported functions: `generateText`, `createElement`, and `validateInput`. The right window shows `util.test.js` with two Jest tests. The terminal at the bottom shows the command `npm test` being executed, resulting in a 'PASS' status for `./util.test.js`. The test results indicate that both test cases passed: 'should output name and age' and 'should output data-less text (1ms)'. The terminal also shows the number of test suites and tests passed, the total number of snapshots, and the execution time.

EXPLORER

- OPEN EDITORS
 - GROUP 1
 - JS util.js
 - GROUP 2
 - JS util.test.js
 - package.json
- JS-TESTING-INTRODUCTI...
 - dist
 - main.js
 - node_modules
 - .gitignore
 - app.js
 - index.html
 - package-lock.json
 - package.json
 - styles.css
 - util.js
 - util.test.js
- OUTLINE
- MAVEN PROJECTS
- NPM SCRIPTS
 - package.json
 - start
 - test

JS util.js

```
1 exports.generateText = (name, age) => {
2   // Returns output text
3   return `${name} (${age} years old)`;
4 };
5
6 exports.createElement = (type, text, className) => {
7   // Creates a new HTML element and returns it
8   const newElement = document.createElement(type);
9   newElement.classList.add(className);
10  newElement.textContent = text;
11  return newElement;
12 };
13
14 exports.validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
```

JS util.test.js

```
1 //nodejs import syntax, compatible with jest:
2 const { generateText } = require('./util');
3
4
5 test(function defined by the Test Runner
6   'should output name and age',
7   () => {
8     const text = generateText('George', 23);
9
10    //function defined by the Assertion Library
11    expect(text).toBe('George (23 years old)');
12  });
13
14
15 test('should output data-less text', () => {
16   const text = generateText('', null);
17   expect(text).toBe(' (null years old)');
18 })
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PASS ./util.test.js

- ✓ should output name and age
- ✓ should output data-less text (1ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 0.206s, estimated 1s
Ran all test suites.

Watch Usage: Press w to show more.

Integration Test

The screenshot displays the VS Code interface with the following components:

- EXPLORER:** Shows the project structure with files like `app.js`, `main.js`, `index.html`, and `util.js`.
- JS app.js:** Contains the main application logic, including `generateText`, `createElement`, `validateInput`, `initApp`, and `addUser` functions.
- JS util.js:** Contains utility functions for generating text, creating elements, and validating input.
- TERMINAL:** Shows the output of the build process, indicating that the main.js file has been built successfully.

```
JS app.js > [e] addUser > [e] outputText
1  const { generateText, createElement, validateInput } = require('./util');
2
3  const initApp = () => {
4    // Initializes the app, registers the button click listener
5    const newUserButton = document.querySelector('#btnAddUser');
6    newUserButton.addEventListener('click', addUser);
7  };
8
9  const addUser = () => {
10   // Fetches the user input, creates a new HTML element based on it
11   // and appends the element to the DOM
12   const newUserNameInput = document.querySelector('input#name');
13   const newUserAgeInput = document.querySelector('input#age');
14
15   if (
16     !validateInput(newUserNameInput.value, true, false) ||
17     !validateInput(newUserAgeInput.value, false, true)
18   ) {
19     return;
20   }
21
22   const userList = document.querySelector('.user-list');
23   const outputText = generateText(
24     newUserNameInput.value,
25     newUserAgeInput.value
26   );
27   const element = createElement('li', outputText, 'user-item');
28   userList.appendChild(element);
29 };
30
31 // Start the app!
32 initApp();
33
```

```
JS util.js > ...
1  exports.generateText = (name, age) => {
2    // Returns output text
3    return `${name} (${age} years old)`;
4  };
5
6  exports.createElement = (type, text, className) => {
7    // Creates a new HTML element and returns it
8    const newElement = document.createElement(type);
9    newElement.classList.add(className);
10   newElement.textContent = text;
11   return newElement;
12 };
13
14 exports.validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
21   }
22   if (isNumber && +text === NaN) {
23     return false;
24   }
25   return true;
26 };
27
```

2: Task - start

Integration Test

The screenshot displays the VS Code editor interface with two JavaScript files open for editing: `app.js` and `util.test.js`.

EXPLORER

- OPEN EDIT... 2 UNSAVED
- GROUP 1
 - JS app.js
- GROUP 2
 - JS util.test.js
- JS TESTING-INTRODUCTI...
 - dist
 - main.js
 - node_modules
 - .gitignore
 - app.js
 - index.html
 - package-lock.json
 - package.json
 - styles.css
 - util.js
 - util.test.js
- OUTLINE
- MAVEN PROJECTS
- NPM SCRIPTS
 - package.json
 - start
 - test

JS app.js > addUser

```
7 };
8
9 const addUser = () => {
10   // Fetches the user input, creates a new HTML element
11   // and appends the element to the DOM
12   const newUserAgeInput = document.querySelector('input');
13   const newUserAgeInput = document.querySelector('input');
14
15   const outputText = checkAndGenerate(
16     newUserAgeInput.value,
17     newUserAgeInput.value
18   );
19
20   if (!outputText) {
21     return;
22   }
23
24   const userList =
25     document.querySelector('.user-list');
26
27   const element = createElement(
28     'li', outputText, 'user-item');
29   userList.appendChild(element);
30 };
31
```

JS util.test.js > checkAndGenerate

```
17   return false;
18 }
19 if (notEmpty && text.trim().length === 0) {
20   return false;
21 }
22 if (isNumber && +text === NaN) {
23   return false;
24 }
25 return true;
26 };
27
28 exports.checkAndGenerate = (name, age) => {
29   if (!validateInput(name, true, false)
30     || !validateInput(age, false, true)) {
31     return false;
32   }
33   return generateText(name, age);
34 };
35
36 exports.generateText = generateText;
37 exports.validateInput = validateInput;
38
```

TERMINAL

```
From https://github.com/academind/js-testing-introduction
* branch          unit-tests -> FETCH_HEAD
Already up-to-date.
george@tars:~/frontend-testing/js-testing-introduction
$
```


Integration Test

The screenshot displays the Visual Studio Code interface with a project named 'JS-TESTING-INTRODUCTI...'. The Explorer sidebar on the left shows the file structure, including 'app.js', 'util.js', and 'util.test.js'. The main editor window shows the content of 'util.test.js', which contains two Jest tests. The first test, 'should output name and age', uses the 'generateText' function. The second test, 'should generate a valid text output', uses the 'checkAndGenerate' function. The bottom panel shows the 'TERMINAL' tab with the command './util.test.js' executed successfully, resulting in two passed tests.

```
JS util.test.js > [?] checkAndGenerate
1 //nodejs import syntax, compatible with jest:
2 const { generateText, checkAndGenerate } = require('./util');
3
4
5 test(//function defined by the Test Runner
6     'should output name and age',
7     () => {
8         const text = generateText('George', 23);
9
10        //function defined by the Assertion Library
11        expect(text).toBe('George (23 years old)');
12    });
13
14
15 test('should generate a valid text output', () => {
16     const text = checkAndGenerate('George', 23);
17     expect(text).toBe('George (23 years old)');
18 })
19
```

3: Task - test

PASS ./util.test.js

- ✓ should output name and age (6ms)
- ✓ should generate a valid text output (3ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 0.272s, estimated 1s
Ran all test suites.

Integration Test

The screenshot displays the VS Code editor with three open files: `app.js`, `util.js`, and `util.test.js`.

util.js (Left Pane):

```
12 };
13
14 const validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
21   }
22   if (isNumber && +text === NaN) {
23     return false;
24   }
25   return true;
26 };
27
28 exports.checkAndGenerate = (name, age) => {
29   if (!validateInput(name, true, false)
30     || !validateInput(age, true, true)) {
31     return false;
32   }
33 }
```

util.test.js (Right Pane):

```
1 //nodejs import syntax, compatible with jest:
2 const { generateText, checkAndGenerate } = require('./util.js');
3
4
5 test('function defined by the Test Runner
6   should output name and age',
7   () => {
8     const text = generateText('George', 23);
9
10    //function defined by the Assertion Library
11    expect(text).toBe('George (23 years old)');
12  });
13
14
15 test('should generate a valid text output', () => {
16   const text = checkAndGenerate('George', 23);
17   expect(text).toBe('George (23 years old)');
18 });
19
```

Terminal Output (Bottom Pane):

```
FAIL ./util.test.js
  ✓ should output name and age (1ms)
  ✗ should generate a valid text output (4ms)

  ● should generate a valid text output

    TypeError: text.trim is not a function

      17 |     return false;
      18 |   }

```

Integration Test

EXPLORER

- JS app.js
- JS util.js
- JS util.test.js

OPEN EDITORS

- JS util.js > checkAndGenerate > exports.checkAndGenerate

```
12 };
13
14 const validateInput = (text, notEmpty, isNumber) => {
15   // Validate user input with two pre-defined rules
16   if (!text) {
17     return false;
18   }
19   if (notEmpty && text.trim().length === 0) {
20     return false;
21   }
22   if (isNumber && +text === NaN) {
23     return false;
24   }
25   return true;
26 };
27
28 exports.checkAndGenerate = (name, age) => {
29   if (validateInput(name, true, false)
30   || !validateInput(age, false, true)) {
31     return false;
32   }
33 }
```

JS util.test.js > checkAndGenerate

```
1 //nodejs import syntax, compatible with jest:
2 const { generateText, checkAndGenerate } = require('./util');
3
4
5 test(function defined by the Test Runner
6   'should output name and age',
7   () => {
8     const text = generateText('George', 23);
9
10    //function defined by the Assertion Library
11    expect(text).toBe('George (23 years old)');
12  });
13
14
15 test('should generate a valid text output', () => {
16   const text = checkAndGenerate('George', 23);
17   expect(text).toBe('George (23 years old)');
18 });
19
```

PROBLEMS

FAIL ./util.test.js

- ✓ should output name and age (1ms)
- ✗ should generate a valid text output (6ms)

● **should generate a valid text output**

expect(received).toBe(expected) // Object.is equality

Expected: "George (23 years old)"

Received: false

E2E Test

The screenshot shows a VS Code editor with the following components:

- EXPLORER:** Shows the file structure with `util.test.js` selected.
- OPEN EDITORS:** Shows `util.test.js` with the following code:

```
21 test('should create an element with text and correct class', async () => {
22   const browser = await puppeteer.launch({
23     headless: true,
24     // slowMo: 80,
25     // args: ['--window-size=1920,1080']
26   });
27   const page = await browser.newPage();
28   await page.goto(
29     'file:///home/george/frontend-testing/js-testing-introduction/index.html'
30   );
31   await page.click('input#name');
32   await page.type('input#name', 'George');
33   await page.click('input#age');
34   await page.type('input#age', '23');
35   await page.click('#btnAddUser');
36   const finalText = await page.$eval('//runs query.Selector of the first argument within the context
37     '.user-item', //CSS class of list element
38     el => el.textContent //el' has the result of query.Selector
39   );//returns a Promise<String>
40   expect(finalText).toBe('George (23 years old)');
41 }, 10000);
```
- TERMINAL:** Shows the command `npm test` and its output:

```
3: Task - test
PASS ./util.test.js
  ✓ should output name and age (8ms)
  ✓ should generate a valid text output
  ✓ should create an element with text and correct class (457ms)

Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 2.097s
```

E2E test: browser

The screenshot displays a web browser window with two tabs: 'about:blank' and 'JS Testing'. The active tab shows a web page at the URL `/home/george/frontend-testing/js-testing-introduction/index.html`. The page content includes a form with 'Name' and 'Age' input fields, an 'Add User' button, and a display area showing 'George (23 years old)'. The browser's developer tools are open on the right, showing the 'Elements' panel with the HTML structure and the 'Styles' panel with CSS rules for the `body` element.

Browser tabs: about:blank, JS Testing

Address bar: `/home/george/frontend-testing/js-testing-introduction/index.html`

Page content:

Name
George

Age
23

Add User

George (23 years old)

Developer Tools - Elements panel:

```
<!doctype html>
<html lang="en">
  <head>...</head>
  <body>...
    <section class="control-panel">
      <div class="input-container">
        <label for="name">Name</label>
        <input type="text" id="name">
      </div>
      <div class="input-container">
        <label for="age">Age</label>
        <input type="number" id="age">
      </div>
      <button id="btnAddUser" class="button">Add User</button>
    </section>
    <hr>
    <section class="user-output">
      <ul class="user-list">
        <li class="user-item">George (23 years old)</li>
      </ul>
    </section>
    <script src="dist/main.js"></script>
  </body>
</html>
```

Developer Tools - Styles panel:

Filter: `element.style`

`body` {

- `font-family: sans-serif;`

`body` {

- `display: block;`
- `margin: 8px;`

Inherited from `html`

`html` {

- `color: -internal-root-color;`

Filter: `color`

- `color: rgb(0, 0, 0)`
- `display: block`
- `font-family`

Async Code Mock

The image shows a web browser window on the left and a code editor on the right. The browser window has a single tab titled 'JS Testing' with the address bar showing the file path `/home/george/frontend-testing/js-te...`. The page content is a button labeled 'Get Post Title!'. The browser's developer console is open, displaying a sequence of log messages: 'Fetching data...' followed by 'DELECTUS AUT AUTEM', repeating five times. Each message is associated with a source location: `http.js:4` for the fetches and `util.js:13` for the log statements. The code editor on the right shows the source code for `index.html`. It includes a DOCTYPE declaration, HTML lang attribute, meta tags for charset, viewport, and compatibility, a link to `styles.css`, and a title 'JS Testing'. The body contains the 'Get Post Title!' button and a script tag that loads `dist/main.js`.

Browser Console Log:

Message	Source
Fetching data...	http.js:4
DELECTUS AUT AUTEM	util.js:13
Fetching data...	http.js:4
DELECTUS AUT AUTEM	util.js:13
Fetching data...	http.js:4
DELECTUS AUT AUTEM	util.js:13
Fetching data...	http.js:4
DELECTUS AUT AUTEM	util.js:13
Fetching data...	http.js:4
DELECTUS AUT AUTEM	util.js:13
Fetching data...	http.js:4
DELECTUS AUT AUTEM	util.js:13

Code Editor Content (index.html):

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <link rel="stylesheet" href="styles.css">
9   <title>JS Testing</title>
10 </head>
11
12 <body>
13   <button>Get Post Title!</button>
14   <script src="dist/main.js"></script>
15 </body>
16
17 </html>
```

Async Code Mock

The screenshot shows a VS Code editor with a project named 'JS-TESTING-INTRODUCTI...'. The Explorer sidebar on the left shows the file structure with folders like 'dist' and 'node_modules', and files like 'app.js', 'http.js', 'index.html', 'package-lock.json', 'package.json', 'styles.css', 'util.js', and 'util.test.js'. The main editor area is split into three panes. The top-left pane shows 'http.js' with a function 'fetchData' that uses 'axios' to fetch data from 'https://jsonplaceholder.typicode.com/todos/1'. The top-right pane shows 'util.js' with functions 'loadTitle' and 'printTitle'. The bottom-left pane shows 'util.test.js' with a Jest test for 'loadTitle'. The bottom-right pane shows the 'TERMINAL' output, indicating that the test passed and the console log 'Fetching data...' was executed.

```
JS http.js > fetchData
1  const axios = require('axios');
2
3  const fetchData = () => {
4    console.log('Fetching data...');
5    return axios
6      .get('https://jsonplaceholder.typicode.com/todos/1')
7      .then(response => {
8        return response.data;
9      });
10 };
11
12 exports.fetchData = fetchData;
13
```

```
JS util.test.js > ...
1  // jest.mock('./http');
2
3  const { loadTitle } = require('./util');
4
5  test('should print an uppercase text', () => {
6    loadTitle().then(title => {
7      expect(title).toBe('DELECTUS AUT AUTEM');
8    });
9  });
10
```

```
JS util.js > ...
1  const { fetchData } = require('./http');
2
3  const loadTitle = () => {
4    return fetchData().then(extractedData => {
5      const title = extractedData.title;
6      const transformedTitle = title.toUpperCase();
7      return transformedTitle;
8    });
9  };
10
11 const printTitle = () => {
12   loadTitle().then(title => {
13     console.log(title);
14     return title;
15   });
16 };
17
18 exports.printTitle = printTitle;
19 exports.loadTitle = loadTitle;
20
```

```
3: Task - test
PASS ./util.test.js
  ✓ should print an uppercase text (6ms)

console.log http.js:4
  Fetching data...
```

Async Code Mock

The screenshot displays a VS Code editor with a Jest test setup for an async axios call. The test mocks the axios fetchData function to return a promise with a title. The test passes, showing the output 'PASS ./util.test.js' and 'should print an uppercase text (2ms)'.

EXPLORER

- JS app.js
- JS http.js
- GROUP 2
- JS http.js __mo... M
- GROUP 3
- <> index.html M
- JS util.js C
- GROUP 4
- JS util.test.js C
- JS-TESTING-INTRODUCTI...
- __mocks__
- JS http.js M
- dist
- JS main.js C
- node_modules
- .gitignore
- JS app.js C
- JS http.js A
- <> index.html M
- package-lock.json C
- package.json C
- styles.css
- JS util.js C
- JS util.test.js C
- OUTLINE
- MAVEN PROJECTS
- NPM SCRIPTS
- package.json C
- start
- test

JS http.js

```
1 const axios = require('axios');
2
3 const fetchData = () => {
4   console.log('Fetching data...');
5   return axios
6     .get('https://jsonplaceholder.typicode.com/todos/1')
7     .then(response => {
8       return response.data;
9     });
10 };
11
12 exports.fetchData = fetchData;
13
```

JS util.js

```
1 const { fetchData } = require('./http');
2
3 const loadTitle = () => {
4   return fetchData().then(extractedData => {
5     const title = extractedData.title;
6     const transformedTitle = title.toUpperCase();
7     return transformedTitle;
8   });
9 };
10
11 const printTitle = () => {
12   loadTitle().then(title => {
13     console.log(title);
14     return title;
15   });
16 };
17
18 exports.printTitle = printTitle;
19 exports.loadTitle = loadTitle;
```

JS util.test.js

```
1 jest.mock('./http');
2
3 const { loadTitle } = require('./util');
4
5 test('should print an uppercase text', () => {
6   loadTitle().then(title => {
7     expect(title).toBe('DELECTUS AUT AUTEM');
8   });
9 });
```

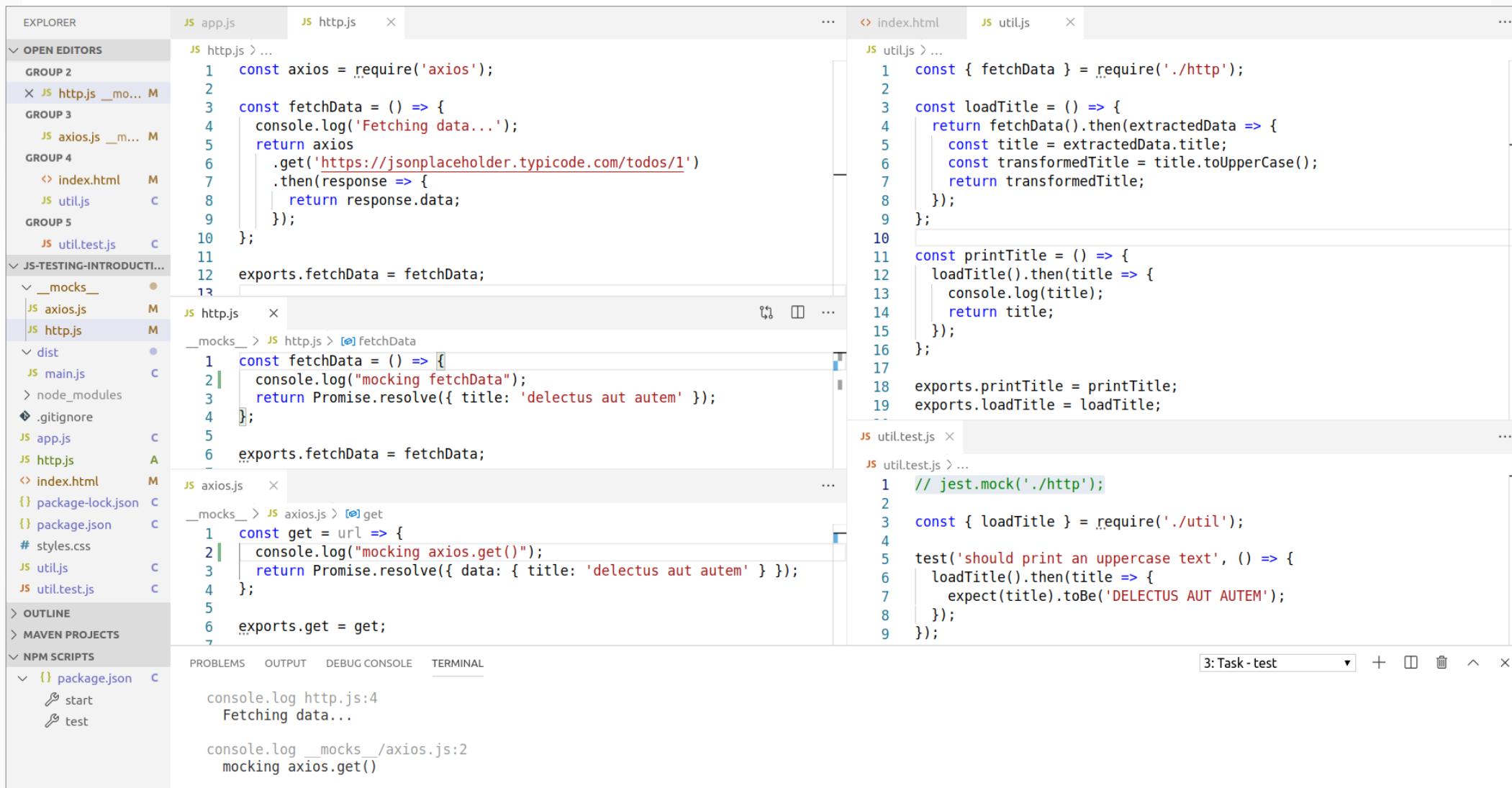
PROBLEMS **OUTPUT** **DEBUG CONSOLE** **TERMINAL**

PASS ./util.test.js

✓ should print an uppercase text (2ms)

console.log __mocks__/http.js:2
mocking fetchData

Async Code Mock



The screenshot displays a VS Code workspace for a project named 'Task-test'. The Explorer on the left shows the file structure, including a 'dist' folder and a 'node_modules' directory. The Editor shows the code for several files:

- http.js**:

```
1 const axios = require('axios');
2
3 const fetchData = () => {
4   console.log('Fetching data...');
5   return axios
6     .get('https://jsonplaceholder.typicode.com/todos/1')
7     .then(response => {
8       return response.data;
9     });
10 };
11
12 exports.fetchData = fetchData;
```
- http.js** (mocks):

```
1 const fetchData = () => {
2   console.log("mocking fetchData");
3   return Promise.resolve({ title: 'delectus aut autem' });
4 };
5
6 exports.fetchData = fetchData;
```
- axios.js** (mocks):

```
1 const get = url => {
2   console.log("mocking axios.get()");
3   return Promise.resolve({ data: { title: 'delectus aut autem' } });
4 };
5
6 exports.get = get;
```
- util.js**:

```
1 const { fetchData } = require('./http');
2
3 const loadTitle = () => {
4   return fetchData().then(extractedData => {
5     const title = extractedData.title;
6     const transformedTitle = title.toUpperCase();
7     return transformedTitle;
8   });
9 };
10
11 const printTitle = () => {
12   loadTitle().then(title => {
13     console.log(title);
14     return title;
15   });
16 };
17
18 exports.printTitle = printTitle;
19 exports.loadTitle = loadTitle;
```
- util.test.js**:

```
1 // jest.mock('./http');
2
3 const { loadTitle } = require('./util');
4
5 test('should print an uppercase text', () => {
6   loadTitle().then(title => {
7     expect(title).toBe('DELECTUS AUT AUTEM');
8   });
9 });
```

The Terminal at the bottom shows the output of running the tests:

```
console.log http.js:4
  Fetching data...

console.log __mocks__/axios.js:2
  mocking axios.get()
```



Πηγές και επιπλέον άρθρα

- Άρθρο παραδείγματος:
<https://academind.com/learn/javascript/javascript-testing-introduction/>
- Κώδικας παραδείγματος:
<https://github.com/academind/js-testing-introduction>
- Introduction to front-end unit testing:
<https://dev.to/christopherkade/introduction-to-front-end-unit-testing-510n>
- Top JS Testing Frameworks in Demand for 2019:
<https://blog.bitsrc.io/top-javascript-testing-frameworks-in-demand-for-2019-90c76e7777e9>
- UI test automation practices:
<https://www.blazemeter.com/blog/top-15-ui-test-automation-best-practices-you-should-follow/>
- Jest documentation:
<https://jestjs.io/docs/en/getting-started>
- Jasmine documentation:
https://jasmine.github.io/pages/docs_home.html
- Puppeteer documentation:
<https://pptr.dev/>