Project Part 2 Additional Feature Guide: Implementing Frontend Code Coverage Threshold

Upon completion of this lab-sheet, you would be able to:

- Implement frontend code coverage threshold for a Node.js project

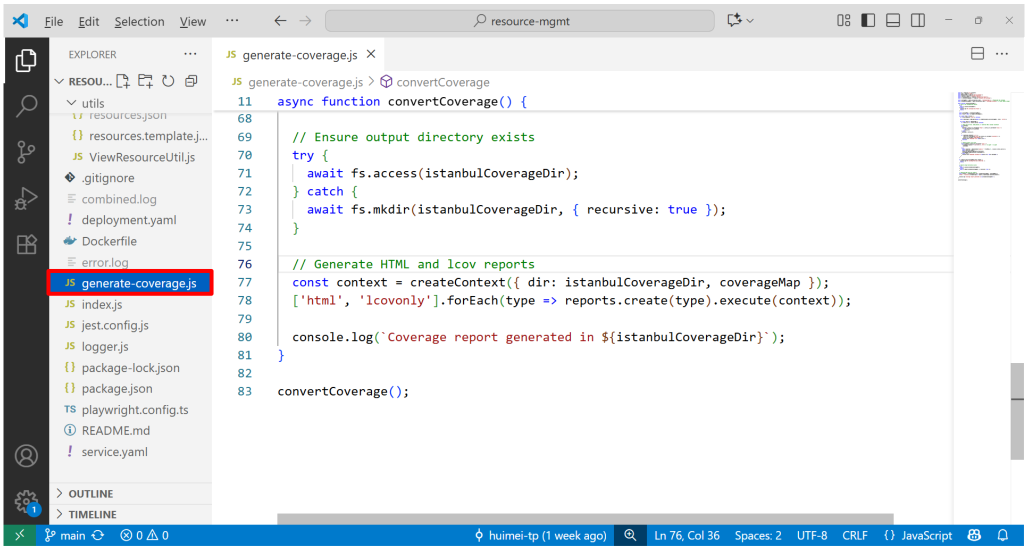
A. Introduction

In Lab-Sheet #6, we learned how to measure code coverage for frontend tests using Playwright. We also explored how Playwright's built-in coverage tools help identify which parts of the frontend code (such as user interactions and UI logic) are executed during testing on a Chromium browser. In this guide, we will extend that process by enforcing a minimum coverage threshold to maintain testing quality standards.

B. Adding a Coverage Threshold Check

The **generate-coverage.js** file processes the raw JavaScript coverage data collected by Playwright and converts it into readable Istanbul reports for frontend analysis. In this section, we will edit the file to include code that enforces threshold.

1. In **Visual Studio Code**, open the **generate-coverage.js** file located in your project folder.



```
11  async function convertCoverage() {
68
69    // Ensure output directory exists
70    try {
71      await fs.access(istanbulCoverageDir);
72    } catch {
73      await fs.mkdir(istanbulCoverageDir, { recursive: true });
74    }
75
76    // Generate HTML and lcov reports
77    const context = createContext({ dir: istanbulCoverageDir, coverageMap });
78    ['html', 'lcovonly'].forEach(type => reports.create(type).execute(context));
79
80    console.log(`Coverage report generated in ${istanbulCoverageDir}`);
81  }
82
83  convertCoverage();
```

2. Locate the following line of code (Line 78) in **generate-coverage.js**:

```
['html', 'lcovonly'].forewatch(type => reports.create(type).execute(context));
```

The above line of code generates the coverage reports. We will include the threshold computation and checking code after the above line of code.

3. Include the **yellow highlighted code** in **generate-coverage.js**, as follows:

```
...
async function convertCoverage() {
  ...
  // Generate HTML and lcov reports
  const context = createContext({ dir: istanbulCoverageDir, coverageMap });
  ['html', 'lcovonly'].forEach(type => reports.create(type).execute(context));

  // Retrieve overall coverage summary data from the coverage map
  const summary = coverageMap.getCoverageSummary().data;

  // Define minimum acceptable coverage thresholds for each metric (in percentage)
  const thresholds = {
    lines: 80,         // Minimum 80% of lines must be covered
    statements: 80,    // Minimum 80% of statements must be covered
    functions: 80,     // Minimum 80% of functions must be covered
    branches: 80       // Minimum 80% of branches must be covered
  };

  // Array to store any metrics that do not meet the defined threshold
  let belowThreshold = [];

  // Loop through each coverage metric (lines, statements, functions, branches)
  for (const [metric, threshold] of Object.entries(thresholds)) {
    const covered = summary[metric].pct; // Get the coverage percentage for this metric

    // Check if the actual coverage is below the threshold
    if (covered < threshold) {
      // Add a message to the belowThreshold array for reporting later
      belowThreshold.push(`${metric}: ${covered}% (below ${threshold}%)`);
    }
  }

  // If any metrics fall below the required threshold
  if (belowThreshold.length > 0) {
    console.error('\nX Coverage threshold NOT met:');

    // Print each failing metric and its coverage percentage
    belowThreshold.forEach(msg => console.error(`  - ${msg}`));

    // Set exit code to 1 to indicate failure (useful for CI/CD pipelines)
    process.exitCode = 1;
  } else {
    // If all thresholds are met, display a success message
    console.log('\n✓ All coverage thresholds met.');
  }

  console.log(`Coverage report generated in ${istanbulCoverageDir}`);
}

convertCoverage();
```

Ctrl+S to save.

4. We will first run the frontend test cases to collect the coverage data. After that, we will run the script to generate the coverage report. Enter the following command in the **Terminal** to run the frontend test cases:

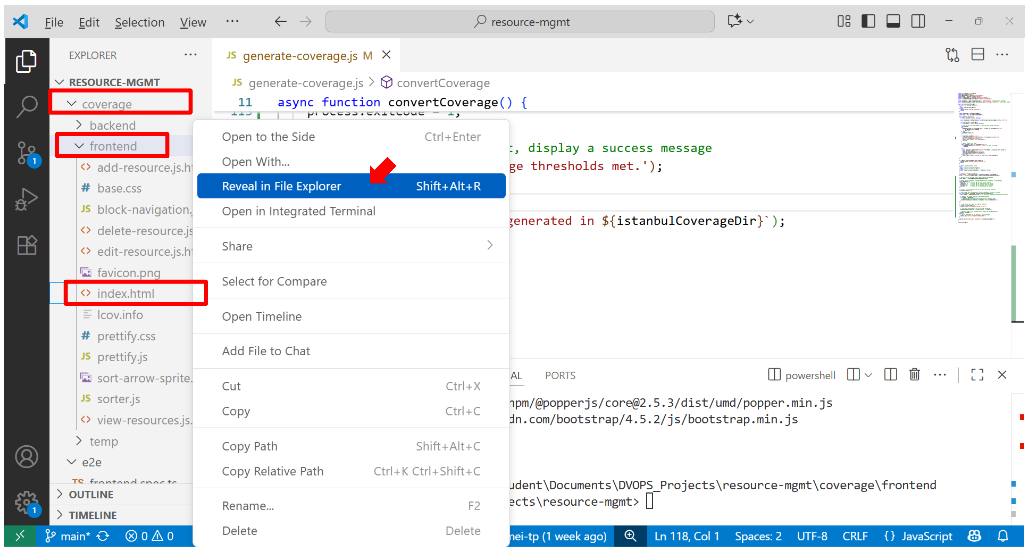
```
npm run test-frontend
```

5. Next, run the following command in **Terminal** to generate the coverage report from the collected data:

```
npm run test-frontend:coverage
```

The reports are generated in the **coverage/frontend** folder. Open the coverage report in your browser to see which parts of the frontend code were tested

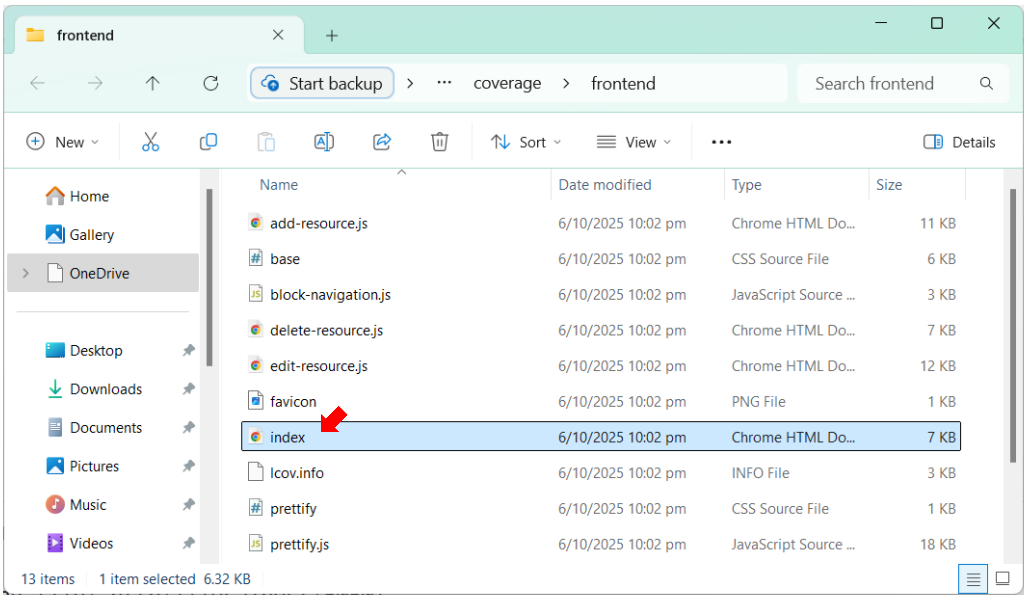
6. To access the HTML report, right click on **index.html** that can be found in **coverage/frontend** → **Reveal in File Explorer**.



```
async function convertCoverage() {
;, display a success message
ge thresholds met.');
generated in ${istanbulCoverageDir}`);
```

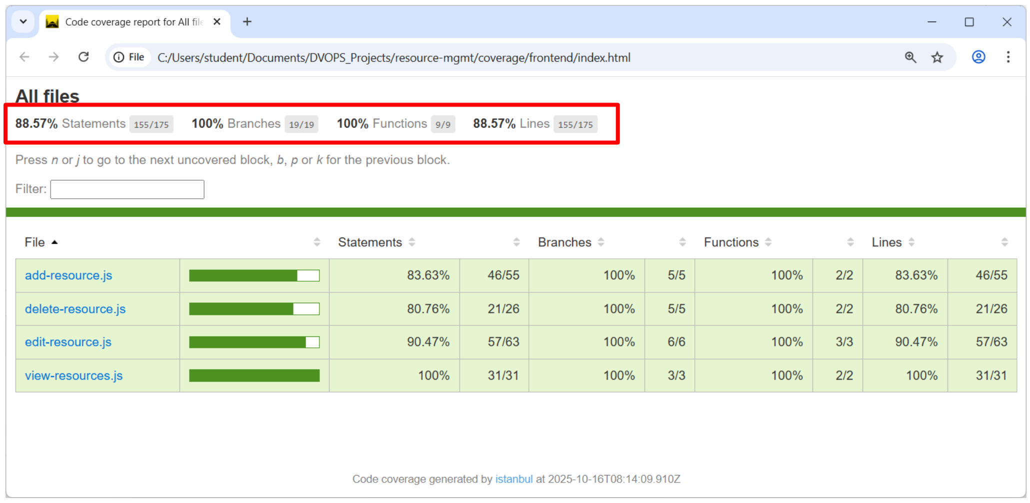
```
npm/@popperjs/core@2.5.3/dist/umd/popper.min.js
dn.com/bootstrap/4.5.2/js/bootstrap.min.js

udent\Documents\DVOPS_Projects\resource-mgmt\coverage\frontend
ects\resource-mgmt>
```

7. In the File Explorer, double click on **index.html** to open in your default web browser.



Name	Date modified	Type	Size
add-resource.js	6/10/2025 10:02 pm	Chrome HTML Do...	11 KB
base	6/10/2025 10:02 pm	CSS Source File	6 KB
block-navigation.js	6/10/2025 10:02 pm	JavaScript Source ...	3 KB
delete-resource.js	6/10/2025 10:02 pm	Chrome HTML Do...	7 KB
edit-resource.js	6/10/2025 10:02 pm	Chrome HTML Do...	12 KB
favicon	6/10/2025 10:02 pm	PNG File	1 KB
index	6/10/2025 10:02 pm	Chrome HTML Do...	7 KB
lcov.info	6/10/2025 10:02 pm	INFO File	3 KB
prettify	6/10/2025 10:02 pm	CSS Source File	1 KB
prettify.js	6/10/2025 10:02 pm	JavaScript Source ...	18 KB

8. The report shows that the coverage for **all metrics is above 80%**, meeting all the set threshold limit, which **tallies** with the printout in **Terminal**.



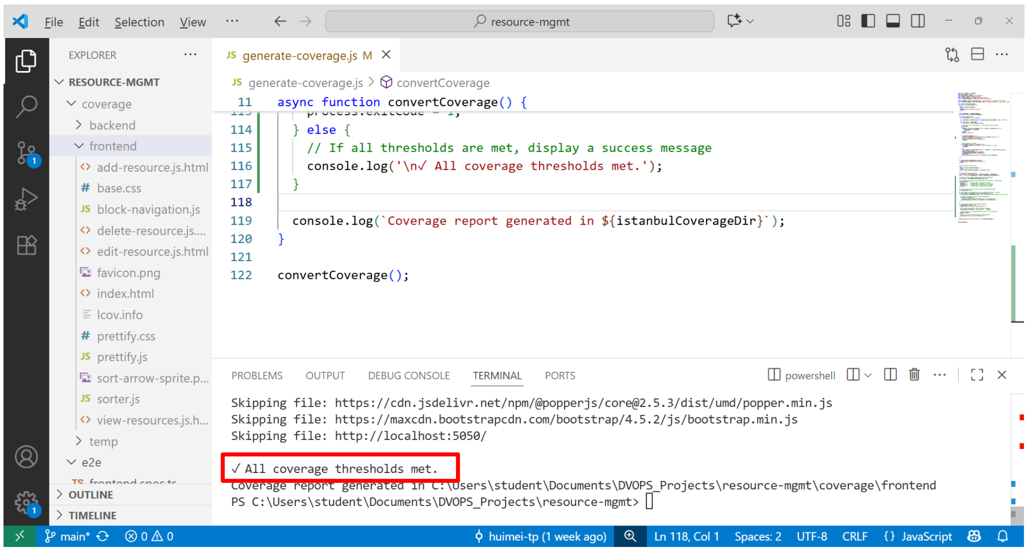
All files

88.57% Statements 155/175 100% Branches 19/19 100% Functions 9/9 88.57% Lines 155/175

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
add-resource.js		83.63%	46/55	100%	5/5	100%	2/2	83.63%	46/55
delete-resource.js		80.76%	21/26	100%	5/5	100%	2/2	80.76%	21/26
edit-resource.js		90.47%	57/63	100%	6/6	100%	3/3	90.47%	57/63
view-resources.js		100%	31/31	100%	3/3	100%	2/2	100%	31/31

Code coverage generated by istanbul at 2025-10-16T08:14:09.910Z



```
async function convertCoverage() {
    process.exitCode = 1;
  } else {
    // If all thresholds are met, display a success message
    console.log('\n√ All coverage thresholds met.');
  }

  console.log(`Coverage report generated in ${istanbulCoverageDir}`);
}

convertCoverage();
```

```
Skipping file: https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js
Skipping file: https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js
Skipping file: http://localhost:5050/

√ All coverage thresholds met.
Coverage report generated in C:\Users\student\Documents\DVOPS_Projects\resource-mgmt\coverage\frontend
PS C:\Users\student\Documents\DVOPS_Projects\resource-mgmt>
```

9. Let's increase the **threhold** to **90%** for all mertrics. Modify **generate-coverage.js**, as follows:

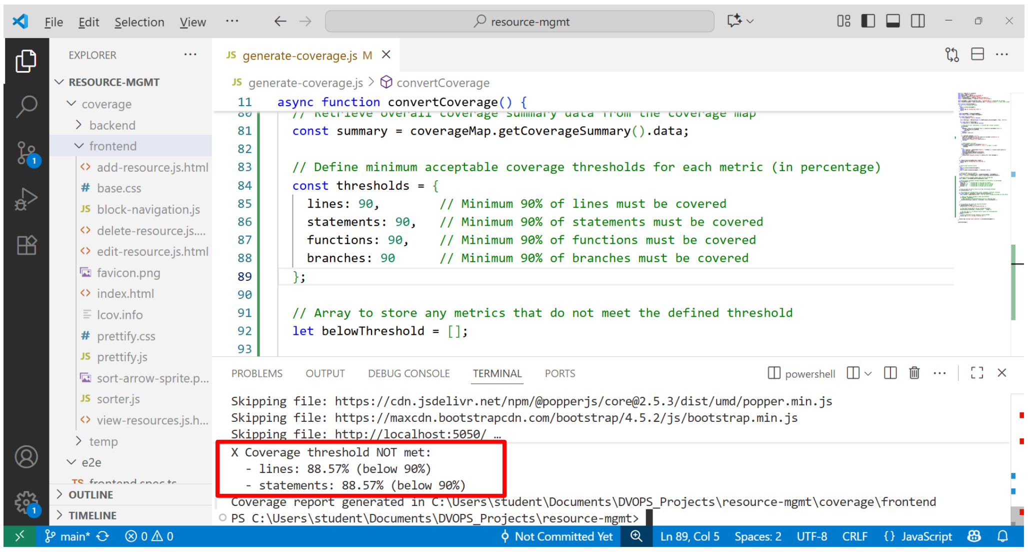
```
...
async function convertCoverage() {
  ...
  // Define minimum acceptable coverage thresholds for each metric (in percentage)
  const thresholds = {
    lines: 90,        // Minimum 90% of lines must be covered
    statements: 90,   // Minimum 90% of statements must be covered
    functions: 90,    // Minimum 90% of functions must be covered
    branches: 90      // Minimum 90% of branches must be covered
  };
  ...
}

convertCoverage();
```

Ctrl+S to save.

10. Run the following command in **Terminal** to generate the coverage report from the collected data:

```
npm run test-frontend:coverage
```

You would see that the lines and statements metrics will now be indicated as not meeting the coverage threshold.



```
async function convertCoverage() {
  // Retrieve overall coverage summary data from the coverage map
  const summary = coverageMap.getCoverageSummary().data;

  // Define minimum acceptable coverage thresholds for each metric (in percentage)
  const thresholds = {
    lines: 90,          // Minimum 90% of lines must be covered
    statements: 90,     // Minimum 90% of statements must be covered
    functions: 90,      // Minimum 90% of functions must be covered
    branches: 90        // Minimum 90% of branches must be covered
  };

  // Array to store any metrics that do not meet the defined threshold
  let belowThreshold = [];
```

```
Skipping file: https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js
Skipping file: https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js
Skipping file: http://localhost:5050/ …
X Coverage threshold NOT met:
  - lines: 88.57% (below 90%)
  - statements: 88.57% (below 90%)
Coverage report generated in C:\Users\student\Documents\DVOPS_Projects\resource-mgmt\coverage\frontend
PS C:\Users\student\Documents\DVOPS_Projects\resource-mgmt>
```

With that, we are done. In your report, describe what you did and explain the reason behind the threshold values you selected. Also, discuss why setting a frontend code coverage threshold is important.

<u>End of Guide</u>