JavaScript Modules

Javascript Plus Session-23

Table of Contents

1 JavaScript Modules

What is Module?

- A file - each script file is a module
- Reusable piece of code that encapsulates implementation details
- Modules are usually standalone files, don't have to be.

Benefits of Modules

- **Compose software:** small packages → complex applications
- **Isolate components:** provides isolation on entire codebase
- **Abstract code:** low level implemented code can be stored inside modules, we only call them without knowing the details.
- **Organized code:** helps for more organized codebase
- **Reuse code:** the same code can be reused across multiple projects.

👉 DRY: do not repeat yourself

Brief History

Approach	Runs on	Loaded	Extension
Script	browsers	async	.js
CommonJS	servers	sync	.js .cjs
AMD module	browsers	async	.js
UMD module	browsers and servers	depends	.js
ECMAScript module	**browsers and servers(!)**	**async**	**.js .mjs**

ES6 Module vs Script

	ES6 Module	Regular script file
namespace pollution	no inside module	global
mode	strict mode	sloppy or loose checking
top-level this	undefined	window
import export	✅YES (hoisted)	❌NO
HTML linking	<script type=”module”>	<script>
download	async	sync
dev env	needs live server	works from local file

How to

- Writing a module
- Using a module
  - from js
  - from html

Writing a Module (ES6 Style)

Declare export

mymodule.js > ...

```
// ——— mymodule.js ———
// named export
export const PI = 3.14;
export const SECONDS_IN_A_DAY = 86400;
export const VERSION = 4.01;
const MINOR_VERSION = 2.26;

export function veryLongNamedFunctionThatDoesSomethingVeryImportant() {
    return 'veryLongNamedFunctionThatDoesSomethingVeryImportant';
}

```

Writing a Module (ES6 Style)

Rename export

export as list

mymodule.js > ...

```
12  // rename export
13  export { SECONDS_IN_A_DAY as SECDAY };
14  export { veryLongNamedFunctionThatDoesSomethingVeryImportant as doSmt };
15
16  // export as list & rename
17  export { MINOR_VERSION,
18      VERSION as VER,
19      veryLongNamedFunctionThatDoesSomethingVeryImportant as doSomething };
20
```

Writing a Module (ES6 Style)

default export

mymodule.js > ...

```
1  // ——— mymodule.js ———
2  // default export (!only one)
3  export default num => {
4      return num * num;
5  };
6  // or ! only one default export is allowed
7  export default 'Module name is mymodule'
8  // don't try to give a name!
9  export default const moduleName = 'value';
```

Using a Module (ES6 Style)

declare import

myApp.js

```
// ——— myApp.js ———
// named import
import { SECONDS_IN_A_DAY, doSmt, MINOR_VERSION } from './mymodule.js';
console.log(SECONDS_IN_A_DAY);

// default import
import myName from './mymodule.js';
console.log(myName);

// namespace import everything from mymodule with an alias
import * as m1 from './mymodule.js';
console.log(m1.MINOR_VERSION);
```

Using a Module (ES6 Style)

declare import

myApp.js

```
// rename import
import { SECONDS_IN_A_DAY as SN_GUN } from './mymodule.js';
console.log(SN_GUN);

// import multi and rename
import { veryLongNamedFunctionThatDoesSomethingVeryImportant as f1, VERSION } from './mymodule.js';

console.log(f1());
console.log(VERSION);
```

what about named and default import on a single line?

Using a Module (ES6 Style)

in html file

index.html > html

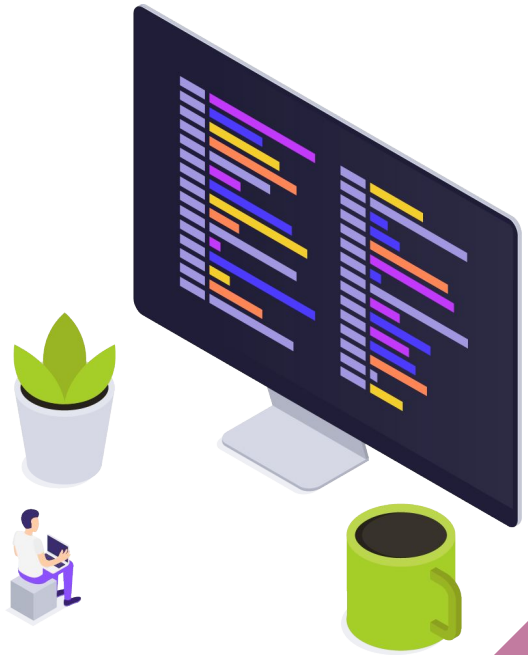
```
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Document</title>
    </head>
    <body>
        <script src="myApp.js" type="module"></script>
    </body>
</html>
```

Introduction to Testing

What is Testing?

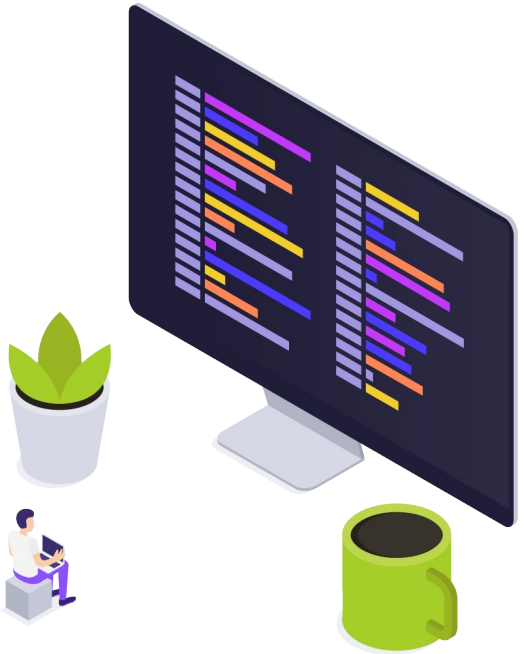
Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do.

- Manual vs Automated testing
- Testing Levels:
  - Unit testing
  - Integration testing
  - End-to-end(system) testing



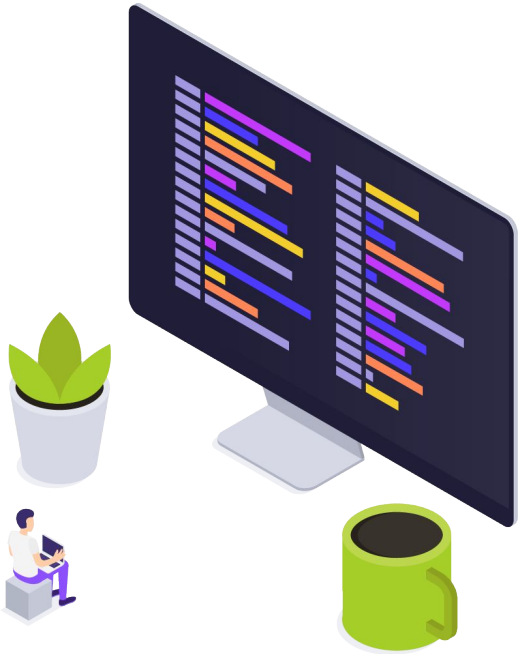
Testing Methods

- Manual
  - till now already what we did
  - code -> preview & check in the browser
  - ! visually ensure output how the users will see
  - * hard to test all combinations



Testing Methods

- Automated
  - a testing framework manages testing process
  - developer writes test code
  - ensures all blocks are tested



Unit Testing

- Testing procedure works
  - by using assertions
  - comparing the output with the expected result
  - giving feedback to the developers

Unit Testing

- We need a framework for testing
  - **Jest**
  - Mocha
  - Chai
  - React Testing Library
    - *testing infrastructure will be delivered by create-react-app, react also uses jest internally.*

Testing Levels

- **Unit tests ****
  - Individual blocks(functions, components)
  - many many xx to xxx unit tests
- **Integration tests**
  - combining units & test them as a whole
- **End-to-end(system) tests**
  - completely test as if user experience the app

Writing Tests - 3 “A”s

- Arrange
  - setup test data, conditions and environment
- Act
  - execute the code to be tested, collect the results
- Assert
  - compare the results with expected values

Writing Tests Steps

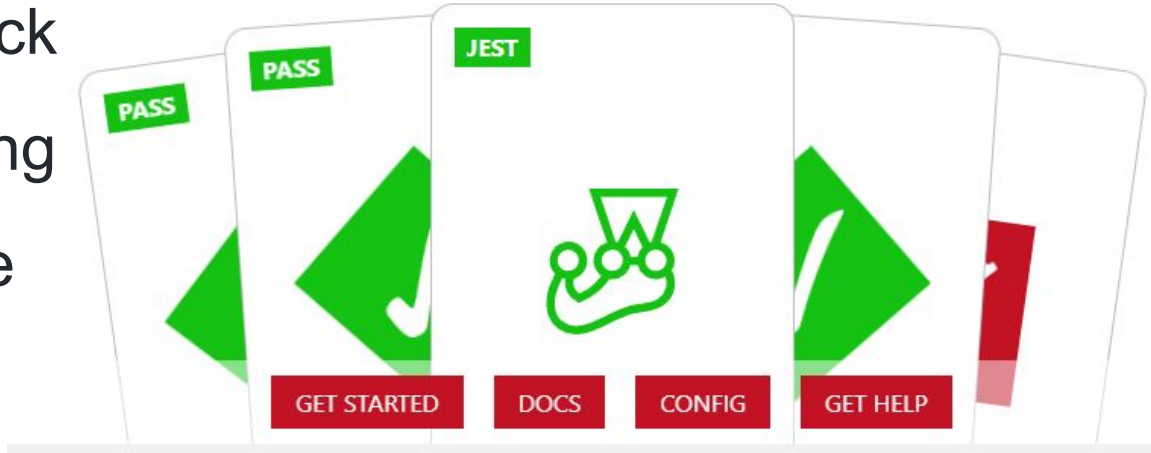
- Setup
- Execution
- Validation
- Cleanup

  revert to the state before the test run

Introduction to Jest

Jest is a delightful JavaScript Testing Framework created by Facebook (meta). It focuses on simplicity.

- Developer Ready Test Runner
- Mocking Support
- Instant Feedback
- Snapshot Testing
- Code Coverage



Intro to Jest

It works with projects using:

- Babel,
- TypeScript,
- Node,
- React,
- Angular,
- Vue and more!

Intro to Jest

Resources

- https://jestjs.io/
- https://en.wikipedia.org/wiki/Jest_(JavaScript_framework)
- https://github.com/facebook/jest#getting-started
- https://www.npmjs.com/package/jest
- https://docs.expo.dev/guides/testing-with-jest/

Let's get our hands dirty!

1

JavaScript Code Compatibility

JS Code Compatibility

How to make our modern code work on older engines that don't understand recent features yet?

Transpilers

Polyfills

JS Transpilers

Transpiler

- special piece of software.
- translates source code to another source code.
- can parse modern code.
- rewrite the modern code using older syntax constructs.

JS Transpilers

- JavaScript before year 2020 didn’t have the “nullish coalescing operator”.

```
height = height ?? 100;
```

JS Transpilers

```
// before running the transpiler
height = height ?? 100;

// after running the transpiler
height = (height !== undefined && height !== null) ? height : 100;
```

JS Transpilers

- ASM
- Babel
- CoffeeScript
- Dart
- GrooScript
- JSIL
- Lua JS
- Opal
- PureScript
- Pyjamas
- Scala
- Sweet
- TypeScript
- Traceur
- Whalesong

JS Pollyfills

New language features:

- Syntax constructs
- Operators
- Built-in functions

```
Math.trunc(n)
```

JS Pollyfills

New language features:

- Syntax constructs
- Operators
- **Built-in functions**

declare the missing function

```
Math.trunc(n)
```

JS Pollyfills

```
if (!Math.trunc) { // if no such function
  // implement it
  Math.trunc = function(number) {
    // Math.ceil and Math.floor exist even in ancient JavaScript engines
    // they are covered later in the tutorial
    return number < 0 ? Math.ceil(number) : Math.floor(number);
  };
}
```

JS Pollyfills

libraries of polyfills

- core js: allows to include only needed features.
- polyfill.io service that provides a script with polyfills.

JS Code Compatibility

resources showing the support for features

- https://kangax.github.io/compat-table/es6/: pure JS.
- https://caniuse.com/: browser-related functions.

THANKS!

Any questions?