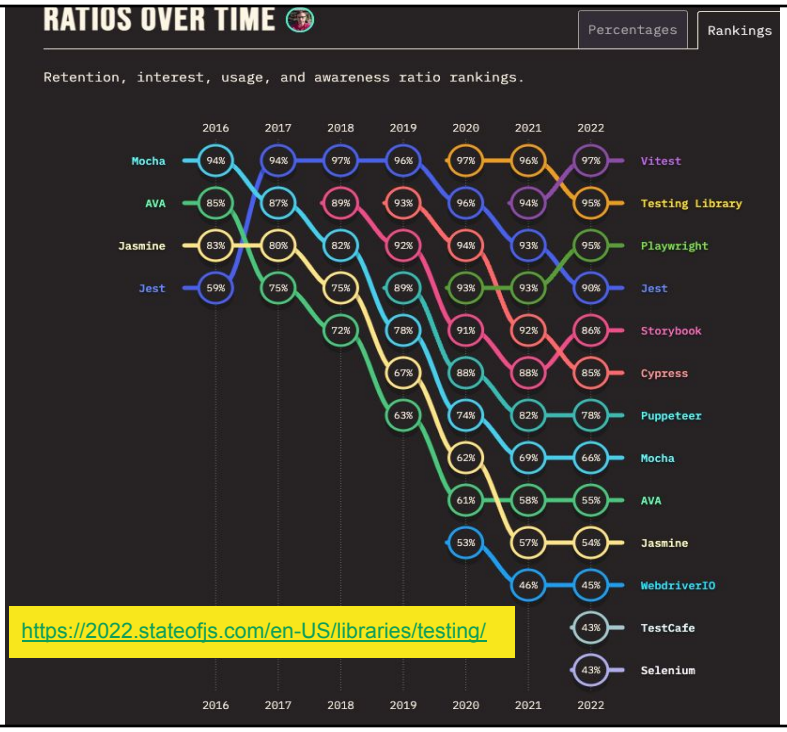
Vitest

Topics

- Testing Framework (Hardware + Software)
- JavaScript Test Libraries
- Vitest Overview
- Using vitest
  - For testing functions
  - For testing Vue Components (code snippets included)
  - For testing React Components

Approach to (Hardware) Testing



https://2022.stateofjs.com/en-US/libraries/testing/

Vitest Overview

- Compatible with test APIs used by many other test platforms (Jest, Mocha, Jasmine, Chai, ...)
- Integrated with vite dev server, test runner does not have to deal of transforming source files (.ts, .vue)
- Use the same configuration file (vite.config.ts) as your application

Testing Framework

- Test Runner
  - Allows developers to describe and organize test cases
  - Run test cases, collect results
  - Analysis and Test Coverage
- Matchers
  - Assertions to verify result of calculation, result of function calls,
  - Utility predicates for writing boolean expressions
- Test Doubles ("Stunt" in movie making)
  - Developer-managed **replacement** for external dependencies
  - Function spies: to monitor function calls
  - Mocks, Stubs: to substitute a real function with a "fake" one
  - Fakes: fake timer, fake canvas, fake network, ...

Scaffold for wing, Sensors, computers....

Electronic transducers, Analog/Digital converters

Cockpit mockup, Pilot dashboard mockup, warning lights, etc..

JS Test Libraries

	Test Runner	Assertions / Matchers	Test Doubles	Coverage
Chai		✅		
Istanbul				✅
Jasmine	✅	✅	✅	
Jest	✅	✅	✅	✅
Karma (browser)	✅			
Mocha	✅			
Sinon			✅	
Vitest	✅	✅	✅	✅

Which One?

Getting Started: Add to Existing Vite-Based Project

- Add vitest as development dependency

```
npm install -D vitest
npm install -D @vitest/coverage-c8       # only when test coverage needed
npm install -D @vitest/coverage-istanbul # pick c8 or istanbul
```

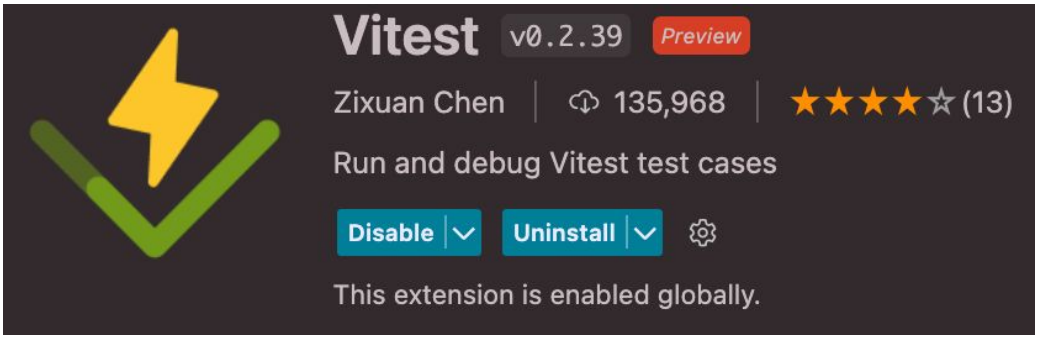
- Add new scripts section in `package.json`

```
{
  “scripts”: {
    “test”: “vitest”,
    “test-coverage”: “vitest run --coverage”
  }
}
```

Prerequisites

- Vite version 3.x (or newer)
- NodeJS version 14.18 (or newer)

VSCode Extension



Vitest Configuration

By default, vitest scans your project directory for the following file names:

- `_____.(spec|test|).(js|cjs|mjs|jsx|ts|cts|mts|tsx)`
- Manual configuration:

```
// vite.config.ts
// import {defineConfig} from “vite”
import { defineConfig } from "vitest/config";
import vue from "@vitejs/plugin-vue";

// https://vitejs.dev/config/
export default defineConfig({
 plugins: [vue()],
 test: {
   includeSource: ["wordle-tests/**/*.test.ts"],
 },
});
```

vitest Hello World

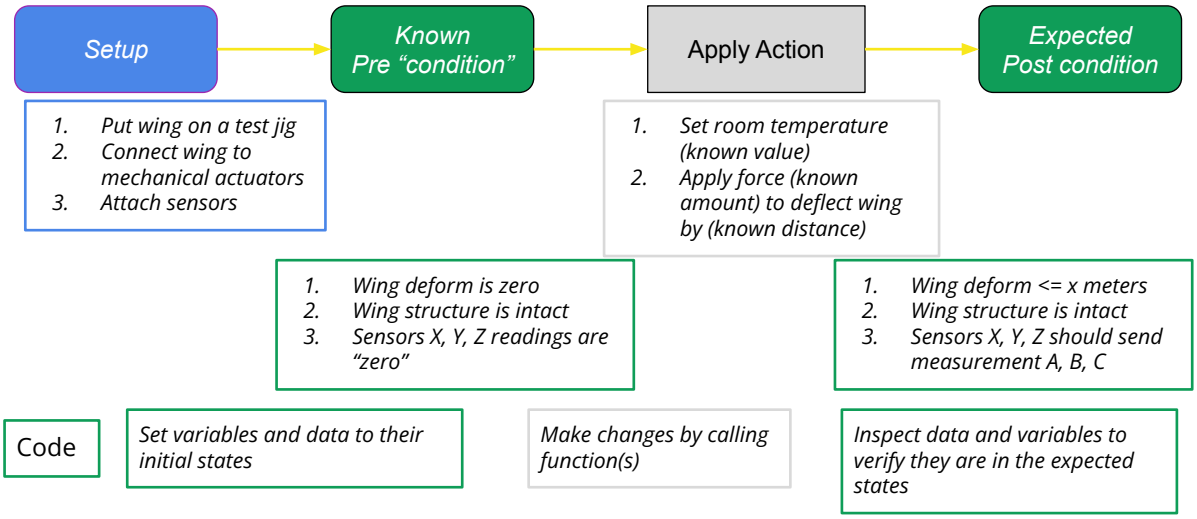
```
// _____/hello.test.ts
import { expect, test, it } from "vitest";

it("checks hello world", () => {
  expect("Hello").not.toEqual("World");
});

          /* test and it are aliases of each other */
test("is failing hello", () => {
  expect("Hello").toEqual("World");
});
```

Online Playground

General Approach



Testing a Function

```
// Function to test in gen.ts
export function generate(n:number): string | null {
  if (n == 0) return null;
  else if (n > 0) return "*".repeat(n)
  else throw "Length can't be negative"
}
```

Online Playground

```
import { assert, expect, test } from 'vitest';
import { generate } from './path/to/test/target/gen';

it('returns null on zero length', () => {
  const s = generate(0);
  expect(s).toBeNull();
});

it('returns repeated string on positive length', () => {
  const s = generate(13);
  expect(s).toBeDefined();
  expect(s!.length).toBe(13);
  expect(s).toContain('*************');
});

it('throws error on negative length', () => {
  expect(() => generate(-3)).toThrowError();
});
```

Test APIs: `it()` ⇔ `test()`

Test function	Description
`it()`	Defines a set of related expect()
`it.skip()`	Skip this test (a convenient way instead of commenting test code)
`it.skipif()`	Conditionally skip this test
`it.runif()`	Opposite of `it.skipif()`
`it.only()`	Run only selected tests in a given suite
`it.concurrent()`	Run this test concurrently with other tests
`it.todo()`	Stub out tests to be implemented later
`it.fails()`	Indicate that an assertion will fail explicitly
`it.each()`	Repeat a test with different values

Assertions using expect()

Assertion function	**Description**
toBe(value)	Assert if equal or share the same reference
toBeCloseTo(value, numDigits)	Assert if a floating number is close enough up to the specified precision
toBeTruthy()	Assert if the value is true when converted to boolean
toBeTypeOf(___)	Assert if it is of a certain type (‘boolean’, ‘object’, ‘string’, etc.)
toContain(*some_string*)	Assert if it contains a specific string

Complete List (many more...)

Running a Test with Multiple Values

```
it.each([8, 14, 21])('Repeated string on positive length', (len) => {
  const s = generate(len);
  expect(s).toBeDefined();
  expect(s!.length).toBe(len);
});
```

```
import { my_prime_func } from “./somefile.ts”

it.each([[8, false], [13, true], [211,true]])('checks for prime', (num, isPrim) => {
  expect(my_prime_func(num)).toBe(isPrim);
});
```

Running a Test with Options

```
it('Repeated string on positive length', () => {
  const s = generate(17);
  expect(s).toBeDefined();
  expect(s!.length).toBe(17);
}, { timeout: 3000,            // time limit (3 secs)
     retry: 3,                 // if it fails, retry at most 3 times
     repeat:10                 // repeat the test 10x
   });
```

Testing Vue Components

Setup

- Add Testing Library as development dependency

```
npm install -D @testing-library/vue      # Version 7.x for Vue3 components
npm install -D @testing-library/vue@5    # Version 5.x for Vue2 components

npm install -D happy-dom                 # Browser DOM emulator
```

- Specify Browser DOM emulation environment in `vite.config.ts`

```
{
  test: {
    globals: true,
    environment: 'happy-dom'
  }
}
```

Using @testing-library/vue

Hello.vue

```
<template>
  <h1>Hello World</h1>
</template>
```

hello.test.ts

```
import {render, screen} from “@testing-library/vue”
import Component from “./path/to/Hello.vue”
import {it} from “vitest”

it(“renders”, () => {
  render(Component)
})

it(“shows ‘Hello World’”, () => {
  render(Component)
  screen.getByText(“Hello World”)
})
```

Testing User Interactions (async + await)

Counter.vue

```
<template>
  <p>Counter value is {{ count }}</p>
  <button @click="addOne">+1</button>
</template>
<script setup lang="ts">
import { ref } from 'vue';

const count = ref(0);

function addOne() {
  count.value++;
}
</script>
```

Online playground

counter.test.ts

```
import { fireEvent, render, screen } from '@testing-library/vue';
import Component from './Counter.vue';
import { it, expect } from 'vitest';

it('Show initial counter value 0', () => {
  render(Component);
  screen.getByText('Counter value is 0');
});

it('Update counter on click', async () => {
  render(Component);
  screen.getByText('Counter value is 0');
  const plusBtn = screen.getByText('+1');
  await fireEvent.click(plusBtn);
  screen.getByText('Counter value is 1');
});
```

Passing Prop Value to Components

```vue
<script setup lang="ts">
type CounterProps = {
  start?: number;
};
import { ref, defineProps, withDefaults } from 'vue';
const props = withDefaults(
  defineProps<CounterProps>(),
  {
    start: 0,
  });
const count = ref(props.start);
</script>

<template>
  <p>Counter value is {{ count }}</p>
</template>
```

```vue
<!-- in your UI -->
<Counter />
<Counter start="17" />
```

```ts
import {render, screen } from '@testing-library/vue';
import Component from './Counter.vue';
import { it, expect } from 'vitest';

it('Uses props initial value', () => {
  render(Component, {
    props: {
      start: 17,
    },
  });
  screen.getByText('Counter value is 17');
});
```