Features

- Vite's config, transformers, resolvers, and plugins.
- Use the same setup from your app to run the tests!
- Smart & instant watch mode, like HMR for tests!
- Component testing for Vue, React, Svelte, Lit and more
- Out-of-the-box TypeScript / JSX support
- ESM first, top level await
- Workers multi-threading via Tinypool
- Benchmarking support with Tinybench
- Filtering, timeouts, concurrent for suite and tests
- Jest-compatible Snapshot
- Chai built-in for assertions + Jest expect compatible APIs
- Tinyspy built-in for mocking
- happy-dom or jsdom for DOM mocking
- Code coverage via c8 or istanbul
- Rust-like in-source testing
- Type Testing via expect-type

Learn how to write your first test by Video

Shared config between test, dev and build

Vite's config, transformers, resolvers, and plugins. Use the same setup from your app to run the tests.

Learn more at Configuring Vitest .

Watch Mode

```bash
$ vitest
```

When you modify your source code or the test files, Vitest smartly searches the module graph and only reruns the related tests, just like how HMR works in Vite!

`vitest` starts in `watch mode` by default in development environment and `run mode` in CI environment (when `process.env.CI` presents) smartly. You can use `vitest watch` or `vitest run` to explicitly specify the desired mode.

Common web idioms out-of-the-box

Out-of-the-box ES Module / TypeScript / JSX support / PostCSS

Threads

Workers multi-threading via Tinypool (a lightweight fork of Piscina), allowing tests to run simultaneously. Threads are enabled by default in Vitest, and can be disabled by passing `--no-threads` in the CLI.

Vitest also isolates each file's environment so env mutations in one file don't affect others. Isolation can be disabled by passing `--no-isolate` to the CLI (trading correctness for run performance).

Test Filtering

Vitest provided many ways to narrow down the tests to run in order to speed up testing so you can focus on development.

Learn more about Test Filtering.

Running tests concurrently

Use `.concurrent` in consecutive tests to run them in parallel.

```ts
import { describe, it } from 'vitest'

// The two tests marked with concurrent will be run in parallel
describe('suite', () => {
  it('serial test', async () => { /* ... */ })
  it.concurrent('concurrent test 1', async ({ expect }) => { /* ... */ })
  it.concurrent('concurrent test 2', async ({ expect }) => { /* ... */ })
})
```

If you use `.concurrent` on a suite, every test in it will be run in parallel.

```ts
import { describe, it } from 'vitest'

// All tests within this suite will be run in parallel
describe.concurrent('suite', () => {
  it('concurrent test 1', async ({ expect }) => { /* ... */ })
  it('concurrent test 2', async ({ expect }) => { /* ... */ })
  it.concurrent('concurrent test 3', async ({ expect }) => { /* ... */ })
})
```

You can also use `.skip` , `.only` , and `.todo` with concurrent suites and tests. Read more in the API Reference.

WARNING

When running concurrent tests, Snapshots and Assertions must use `expect` from the local Test Context to ensure the right test is detected.

Snapshot

Jest-compatible snapshot support.

```ts
import { expect, it } from 'vitest'

it('renders correctly', () => {
  const result = render()
  expect(result).toMatchSnapshot()
})
```

Learn more at Snapshot.

Chai and Jest `expect` compatibility

Chai is built-in for assertions plus Jest `expect` -compatible APIs.

Notice that if you are using third-party libraries that add matchers, setting `test.globals` to `true` will provide better compatibility.

Mocking

Tinyspy is built-in for mocking with `jest` -compatible APIs on `vi` object.

```ts
import { expect, vi } from 'vitest'
```

```
const fn = vi.fn()

fn('hello', 1)

expect(vi.isMockFunction(fn)).toBe(true)
expect(fn.mock.calls[0]).toEqual(['hello', 1])

fn.mockImplementation(arg => arg)

fn('world', 2)

expect(fn.mock.results[1].value).toBe('world')
```

Vitest supports both happy-dom or jsdom for mocking DOM and browser APIs. They don't come with Vitest, you might need to install them:

```bash
$ npm i -D happy-dom
# or
$ npm i -D jsdom
```

After that, change the `environment` option in your config file:

```ts
// vite.config.ts
import { defineConfig } from 'vitest/config'

export default defineConfig({
  test: {
    environment: 'happy-dom', // or 'jsdom', 'node'
  },
})
```

Learn more at Mocking.

Coverage

Vitest supports Native code coverage via `c8` and instrumented code coverage via `istanbul` .

```json
{
  "scripts": {
    "test": "vitest",
    "coverage": "vitest run --coverage"
  }
}
```

Learn more at Coverage.

In-source testing

Vitest also provides a way to run tests within your source code along with the implementation, similar to Rust's module tests.

This makes the tests share the same closure as the implementations and able to test against private states without exporting. Meanwhile, it also brings the feedback loop closer for development.

```ts
// src/index.ts

// the implementation
export function add(...args: number[]) {
  return args.reduce((a, b) => a + b, 0)
}

// in-source test suites
if (import.meta.vitest) {
  const { it, expect } = import.meta.vitest
  it('add', () => {
    expect(add()).toBe(0)
    expect(add(1)).toBe(1)
    expect(add(1, 2, 3)).toBe(6)
  })
}
```

Learn more at In-source testing.

Benchmarking `experimental`

Since Vitest 0.23.0, you can run benchmark tests with `bench` function via Tinybench to compare performance results.

```ts
import { bench, describe } from 'vitest'

describe('sort', () => {
  bench('normal', () => {
    const x = [1, 5, 4, 2, 3]
    x.sort((a, b) => {
      return a - b
    })
  })

  bench('reverse', () => {
    const x = [1, 5, 4, 2, 3]
    x.reverse().sort((a, b) => {
      return a - b
    })
  })
})
```

Type Testing `experimental`

Since Vitest 0.25.0 you can write tests to catch type regressions. Vitest comes with `expect-type` package to provide you with a similar and easy to understand API.

ts

```ts
import { assertType, expectTypeOf } from 'vitest'
import { mount } from './mount.js'

test('my types work properly', () => {
  expectTypeOf(mount).toBeFunction()
  expectTypeOf(mount).parameter(0).toMatchTypeOf<{ name: string }>()

  // @ts-expect-error name is a string
  assertType(mount({ name: 42 }))
})
```

Suggest changes to this page

Last updated: 2/14/2023, 6:05:21 PM

Previous page
Getting Started

Next page
CLI