Fast and Modern Testing with Vitest 🚀

MOT ATHENS

Ministry Of Testing
June 26th 2025

Ema Zyka

Staff Software Engineer / allwyn • LOTTERY SOLUTIONS

Co-organizer / Vue.js Athens meetup

emazyka emazyka

🙋 Heard about Vite?

- Fast dev server + HMR
- Native ESM support
- TS, JSX, and CSS out of the box
- Built-in support for Vue, React, etc.
- Rollup pre-configured for builds

- Production-ready bundling
- Multi-page & library mode
- Optimized static assets
- One config for dev, build, SSR
- Rollup-style plugin system

🙋 Heard about Vitest?

Why Vitest?

- Watch mode
- Async lifecycle hooks
- No bundling — native ESM and on-demand serving
- `jsdom` and `happy-dom` support
- Component testing: Vue, React, Svelte, Lit, etc.

Unified Configuration

- Vitest reads your `vitest.config.ts`, sharing build/dev settings
- Or use `vitest.config.ts` to override test-specific options
- Extend own configs: `mergeConfig(...)`

```ts
// 🧪 tests/hello.test.ts

import { describe, it, expect } from 'vitest';
import { sayHello } from '@/utils/hello'; // ← alias works!

describe('sayHello', () => {
  it('greets correctly', () => {
    expect(sayHello('Vite')).toBe('Hello, Vite!');
  });
});
```

```
// 🎯 override test-specific options

import { defineConfig } from 'vitest/config';
import baseConfig from './vite.config';

export default defineConfig({
  ...baseConfig,
  test: {
    globals: true,
    environment: 'jsdom',
  },
});
```

```ts
// 📦 extend your own configs

import { defineConfig, mergeConfig } from 'vitest/config';
import baseConfig from './vite.config';

export default mergeConfig(
  baseConfig,
  defineConfig({
    test: {
      environment: 'jsdom',
      setupFiles: ['./test/setup.ts'],
    },
  })
);
```

In-Source Testing

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

```
export function add(...) { ... }

if (import.meta.vitest) {
  const { it, expect } = import.meta.vitest
  it('works', () => expect(add(1,2)).toBe(3))
}
```

- Tests live alongside code, get private-state access
- Ideal for prototyping and small utilities

Watch Mode & Smart Re-runs

```
npx vitest --watch
```

- Runs tests in HAMR-like hot fashion, re-running only related tests
- Default in dev; detects CI to switch to “run” mode

```ts
// tests/math.test.ts
import { multiply } from '../utils/math';

it('multiplies correctly', () => {
  expect(multiply(2, 3)).toBe(6);
});
```

While changing `multiply()` in `math.ts`

✅ Re-runs just math.test.ts

🚫 Doesn't re-run unrelated tests

👮 Safeguards CI by automatically switching to `run once` mode

Parallel & Concurrent Testing

```
it.concurrent('a', async () => {...})
describe.concurrent('suite', () => {...})
```

- Launch tests in parallel threads by default
- Leverage `.concurrent` for simultaneous execution within suites

Mocking & Time Control

```
vi.useFakeTimers()
vi.setSystemTime(new Date(2000,1,1,13))
```

- Built-in DOM mocks: `happy-dom` or `jsdom`
- `vi.fn`, `vi.spyOn`, `vi.mock` for jest-like mocks
- Full control over clocks and timers

Snapshot Testing

```
expect(result).toMatchSnapshot()
```

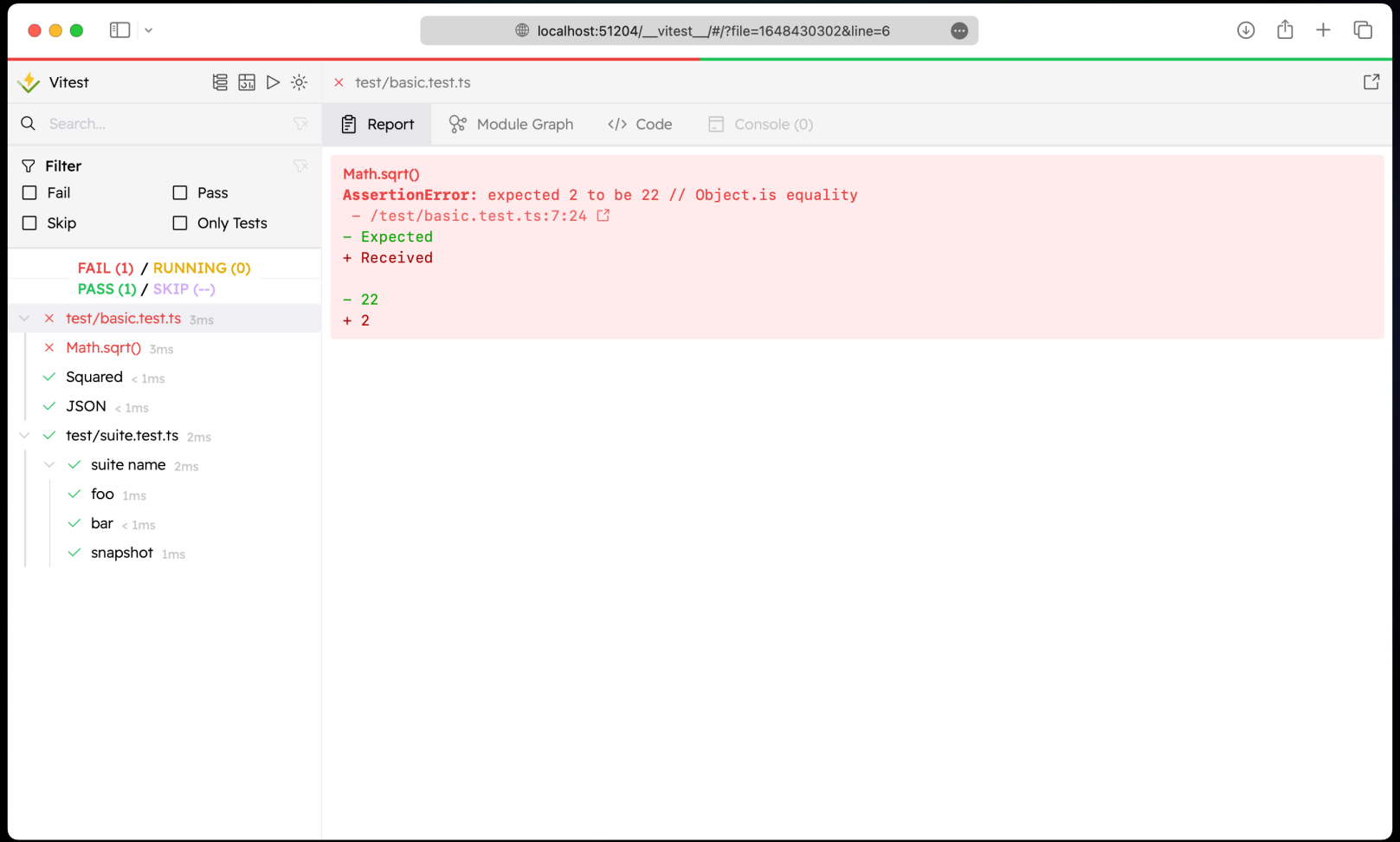
- Jest-compatible snapshots
- Auto handling & integration with watch mode

Vitest UI

Interactive browser-based UI:

```
npm i -D @vitest/ui
vitest --ui
```

- Visualize test suites, assertions, coverage snapshots
- Also available as HTML reporter


localhost:51204/__vitest__/#/?file=1648430302&line=6
Vitest
test/basic.test.ts
Search...
Report
Module Graph
Code
Console (0)
Filter
Fail
Pass
Skip
Only Tests
FAIL (1) / RUNNING (0)
PASS (1) / SKIP (--)
test/basic.test.ts 3ms
Math.sqrt() 3ms
Squared < 1ms
JSON < 1ms
test/suite.test.ts 2ms
suite name 2ms
foo 1ms
bar < 1ms
snapshot 1ms
Math.sqrt()
AssertionError: expected 2 to be 22 // Object.is equality
- /test/basic.test.ts:7:24
- Expected
+ Received
- 22
+ 2

Coverage & Reporting

```
vitest run --coverage
```

- Supports `v8` and `istanbul` instrumentation
- Outputs text & HTML – integrates into CI workflows
- Configurable report folder & ignore rules

```
// 📋 vitest.config.ts

export default {
  test: {
    coverage: {
      reporter: ['text', 'html'],
      exclude: ['src/generated/', 'src/types/'],
      reportsDirectory: './coverage',
    },
  },
};
```

Test Context & Fixtures

```
it('uses fixtures', ({ expect, task }) => { ... })
```

- Inspired by **Playwright**
- `test.extend()` to define fixtures (e.g., DB setup)
- `expect`, `task`, `signal`, hooks within contexts

```ts
const test = vitest.extend({
  db: async ({}, use) => {
    const db = await createTestDatabase();
    await use(db);
    await db.close();
  }
});

test('reads from database', async ({ db, expect }) => {
  const users = await db.getUsers();
  expect(users.length).toBeGreaterThan(0);
});
```

✅ Each test automatically gets a fresh DB object

🔁 Setup and cleanup happen automatically per test

From Jest to Vitest

- 😉 No need for `Babel` or `ts-jest`
- ✍️ Same API surface: `describe`, `it`, `expect`, `vi.mock`
- 🧩 Unified config across dev, build, and test
- 🚀 Faster runs — no bundling overhead

Summary

- Vitest is **fast**, **modern**, and built for Vite
- Rich DX: watch mode, UI, fixtures, coverage
- Easy migration from Jest - low friction

Thank You!

@emazyka

Slides on Github