All slides
Unit testing of a read world typescript system

Nathan Krasney

http://nathankrasney.com/

Section 2

Testing concepts and tools

Introduction

You can skip this chapter , If you the type of guy that love to code first and learn concept later

However, be sure to get back here when i speak of concepts \ tools mentioned in this chapter

What is testing

- Testing is the process of evaluating a software system to ensure it meets the desired requirements and functions correctly
- It involves executing various tests, such as unit tests, integration tests, and end-to-end tests, to uncover defects and ensure the software quality and reliability.
- It the end test has three properties : **test description**, **expected value** and **actual value**. For example
  - The test description can be calling a function that add 1,2.
  - The expected value is 3
  - The actual value is what you get out of the function that implement this add operation

http://nathankrasney.com/

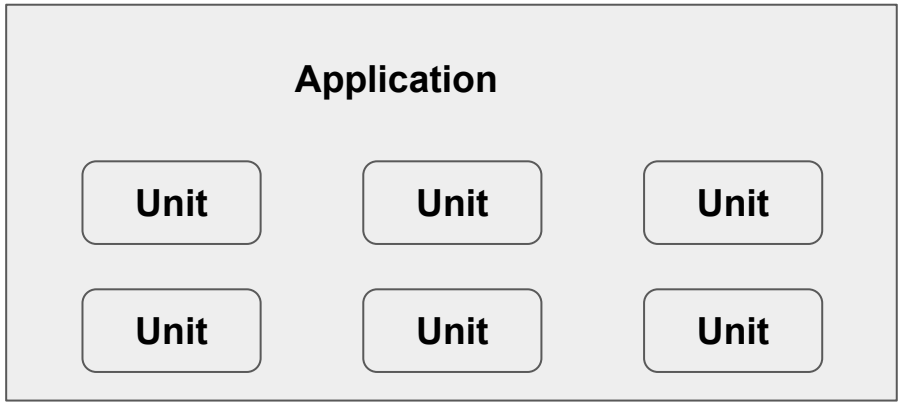
What is unit test vs integration test vs e2e test

- **Unit Test**: Testing individual units or components of the software to ensure they function correctly in isolation.
- **Integration Test**: Testing the interaction between different components or modules of the software to ensure they work together seamlessly.
- **E2E Test**: Testing the entire system as a whole to evaluate its compliance with the specified requirements.

What is a unit in software application

Unit is typically

- Function
- Class
- Component



http://nathankrasney.com/

Side effects

- Modifications to global variables e.g. console , localStorage
- I/O operations : file ,
- Network communication
- Database updates
- UI interactions (DOM)
- Any other interaction with the external world with respect to your application

Properties of good unit test

- **Automated**
- **Deterministic**
- **Repeatable**
- **Fast**
- **Isolated** : from the rest of application , dependencies ,side effects
- **Focused** : on the unit

http://nathankrasney.com/

Manual test vs automatic test

Manual test and automatic test complement each other

Manual test :

- Written in general by QA person and performed by QA person in general when a version is released to the client.
- This consumes time , money an effort each time a test is done
- Manual test are error prone

Automatic test :

- Written in general by developers but performed automatically by a computer using test runners like jest or vitest.
- They can be performed after the night build
- After the test is built the effort to run it is close to zero

http://nathankrasney.com/

Motivation for automatic testing

- Quality of code and product - less bugs
- Provide the developer with confidence for code change and refactoring
- Allow the developer to write better code
- Prevent regression as you write code
- Serve as the best low level documentation - and it is always updated otherwise the tests will fail
- Good tests reduce bugs → less time fixing bugs → improve productivity → ship code changes and products faster
- Save testing time - invoked at button click

http://nathankrasney.com/

Should you test all your code

In theory its nice that your test will cover all your code i.e. coverage of 100%

However, covering all your code require a huge amount of effort

So you need to define the critical part of your project and minimally test them

In my “check your test skills” this critical part is the part that check if you have finished the quiz and compute the wrong and correct answers

Unit testing best practices - chatgpt 1/3

- **Test One Concept at a Time**: Each unit test should focus on testing a single concept or behavior. This helps keep the tests focused and makes it easier to identify and fix issues.
- **Keep Tests Small and Fast**: Unit tests should be small and execute quickly. They should only test a small piece of functionality, such as a single method or function. Fast tests allow for frequent execution, providing rapid feedback during development.
- **Use Descriptive and Readable Test Names**: A test name should clearly describe what is being tested. This helps in understanding the purpose of the test and makes it easier to locate specific tests in a large test suite.
- **Arrange, Act, and Assert (AAA) Pattern**: Structure your tests using the AAA pattern. The "Arrange" phase sets up the necessary preconditions and inputs, the "Act" phase invokes the unit being tested, and the "Assert" phase verifies the expected behavior or outcome.

Unit testing best practices - chatgpt 2/x

- **Test Both Positive and Negative Scenarios**: Ensure that your tests cover both the expected positive scenarios (valid inputs and correct behavior) as well as negative scenarios (invalid inputs, edge cases, and error conditions). This helps identify potential issues and ensures robustness.
- **Use Mocks and Stubs**: Unit tests should focus on testing a specific unit of code in isolation. To achieve this, use mocks or stubs to simulate external dependencies or interactions. This helps control the behavior of external components and enables thorough testing.
- **Test Coverage**: Aim for high test coverage, which refers to the percentage of code that is covered by your tests. While 100% coverage may not always be practical or necessary, strive to achieve coverage that adequately tests critical and complex parts of your codebase.
- **Test Independence and Order**: Ensure that each test is independent and does not rely on the state or behavior of other tests. Tests should be able to run in any order, which makes them easier to maintain and debug.

Unit testing best practices - chatgpt 3/x

- **Regularly Review and Update Tests**: Unit tests should be reviewed regularly to ensure they remain accurate, relevant, and aligned with the current codebase. As the code evolves, update tests to reflect the changes and maintain their effectiveness.
- **Integrate Testing into Your Development Workflow**: Incorporate unit testing into your development process and run tests frequently. Automated testing tools and continuous integration (CI) systems can help execute tests automatically, providing feedback on the health of your codebase.

Unit testing best practices - my take

- Start with the most naturally isolated component
- Test only public methods (if possible)

Unit testing mode

Testing mode	**Host**
Vanilla typescript	Client(browser) \ server
DOM	Client(browser) , jsdom
Mock	Client(browser) \ server
Component	Client(browser) , react testing library

http://nathankrasney.com/

What is vitest

Blazing Fast Unit Test Framework :

- **Vite Powered** : Reuse Vite's config, transformers, resolvers, and plugins - consistent across your app and tests.
- **Jest Compatible : Expect, snapshot, coverage, and more - migrating from Jest is straightforward**
- **Smart & instant watch mode** : Only rerun the related changes, just like HMR for tests!
- **ESM, TypeScript, JSX** : Out-of-box ESM, TypeScript and JSX support powered by esbuild
- **Test runner - automatic testing**

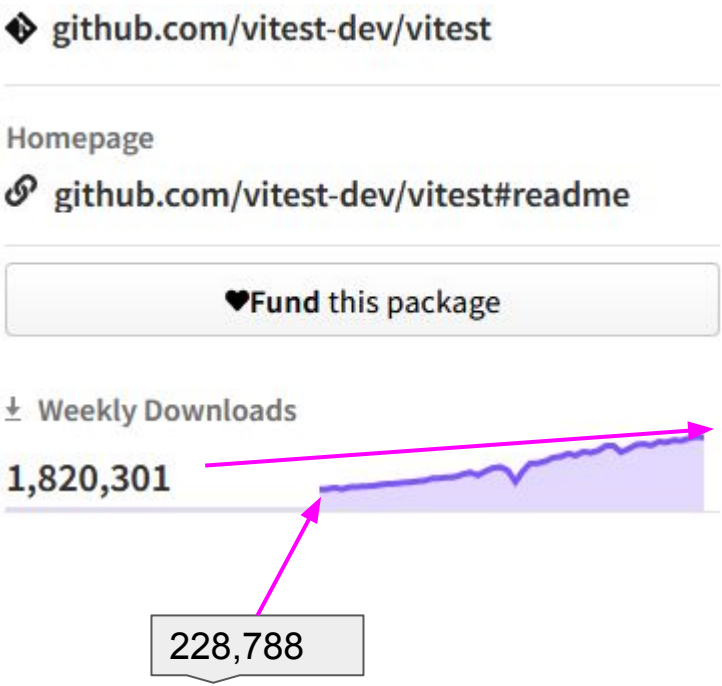
http://nathankrasney.com/

Motivation for vitest 1/x

- **Fast** :
  - Use esbuild (which uses go) to compile ts to js which is 20-30 time faster than the standard ts compile tsc - which is implemented with typescript (check here for vite)
  - Uses multi thread to run the test using service worker via tinypool which uses node.js thread pool using libuv (check this)
- **Used inside vite project** it save you configuration because vite and vitest share the same modules
- **DX**
  - Out of the box support for es module ,typescript , JSX (?)
  - Instant watch mode

Motivation for vitest 2/x

popularity



Almost x8 in one year in weekly download

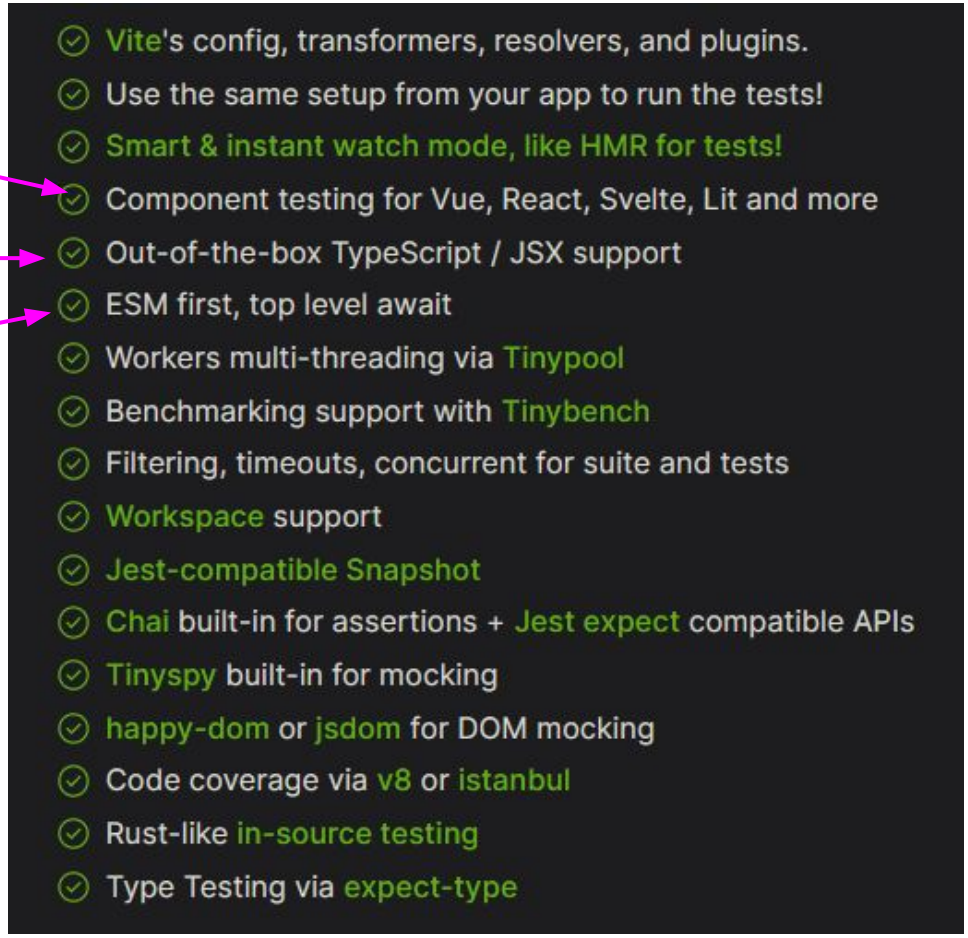
http://nathankrasney.com/

Features of vitest

Integrate e.g. with react testing library , e.g. here

Try to test mini ts project with jest without rollup

Top level await here



http://nathankrasney.com/

What is jest 1/x

Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more!

zero config

Jest aims to work out of the box, config free, on most JavaScript projects.

snapshots

Make tests which keep track of large objects with ease. Snapshots live either alongside your tests, or embedded inline.

isolated

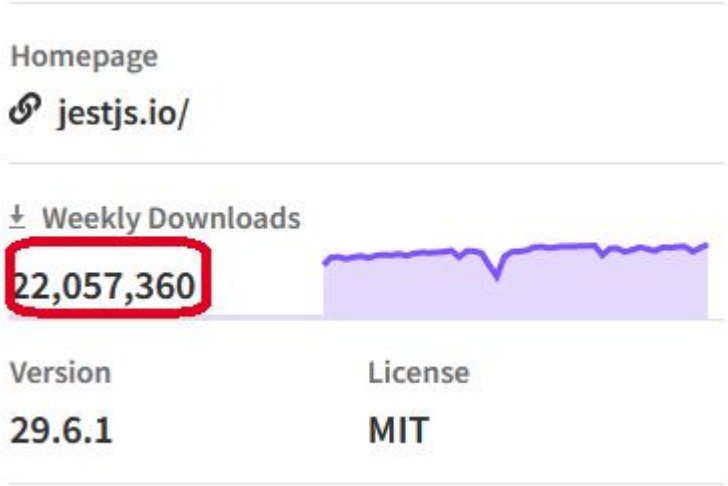
Tests are parallelized by running them in their own processes to maximize performance.

great api

From `it` to `expect` - Jest has the entire toolkit in one place. Well documented, well maintained, well good.

http://nathankrasney.com/

What is jest 2/x



Very popular

Maintained by facebook

http://nathankrasney.com/

Vitest vs jest

According to this :

- Typescript out of the box
- Import out of the box
- Faster
- Default mode is watch mode
- Simple coverage test
- Easy configuration of test coverage results : test, html , json
- Configuration of vite and vitest is in one file
- You can use tests inside the source files (not sure i like it because :
  - of separation of cencerens
  - time to scan source file for test
  - It appears on the build by default (you can tell it not to include it)

http://nathankrasney.com/

Vitest is not a silver bullet

Check this issue for run test performance vs jest

Before moving from jest to vitest you should do a POC first and check run test performance and other DX parameters

Section 3

Task queue manager - the system under unit test

Introduction

To test a system we need to know the system

There few ways to represent the system and i will show you few of them

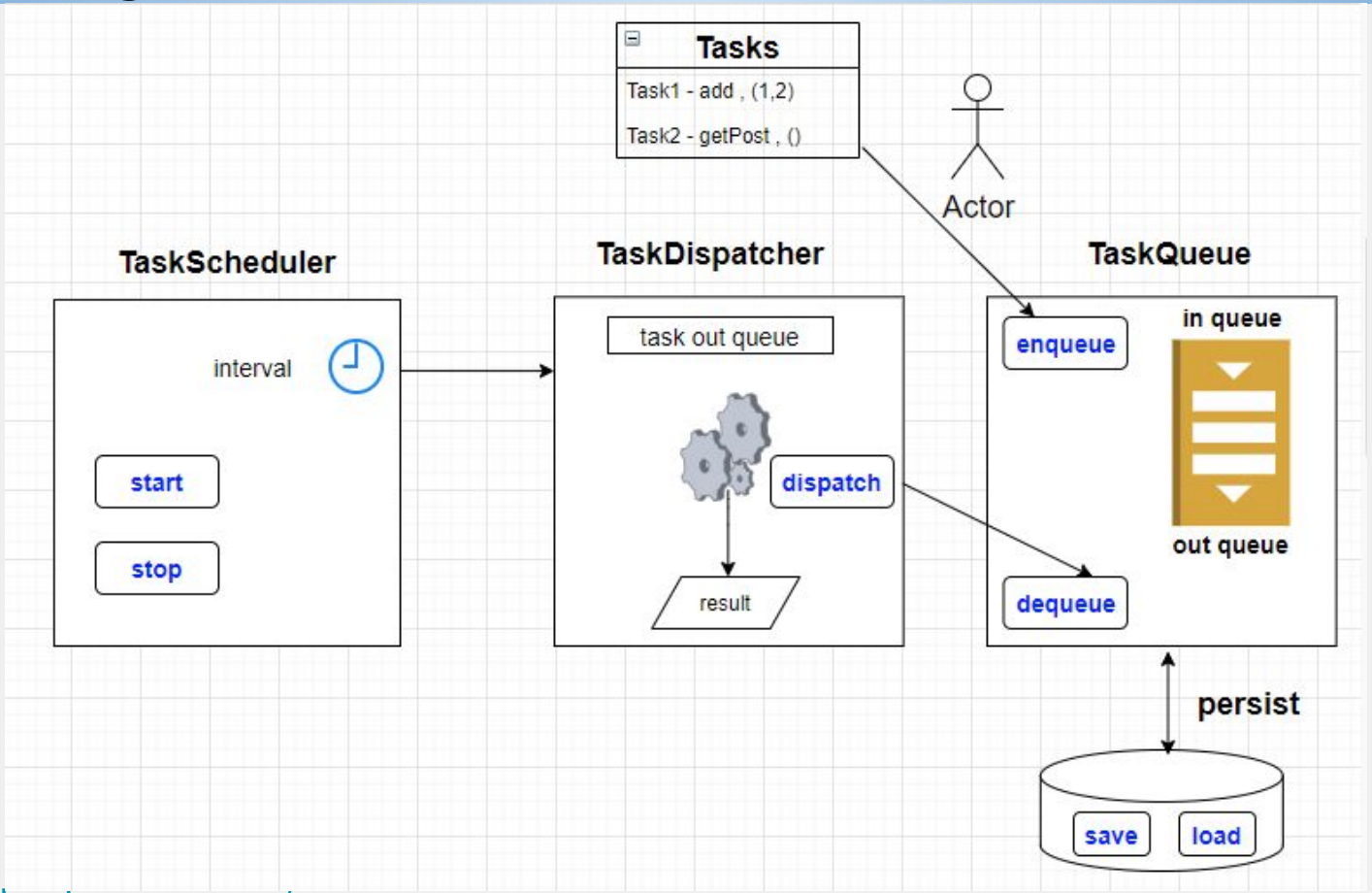
- UI
- Block diagram
- Sequence diagram
- Source code

Introduction continued

But we need also to know

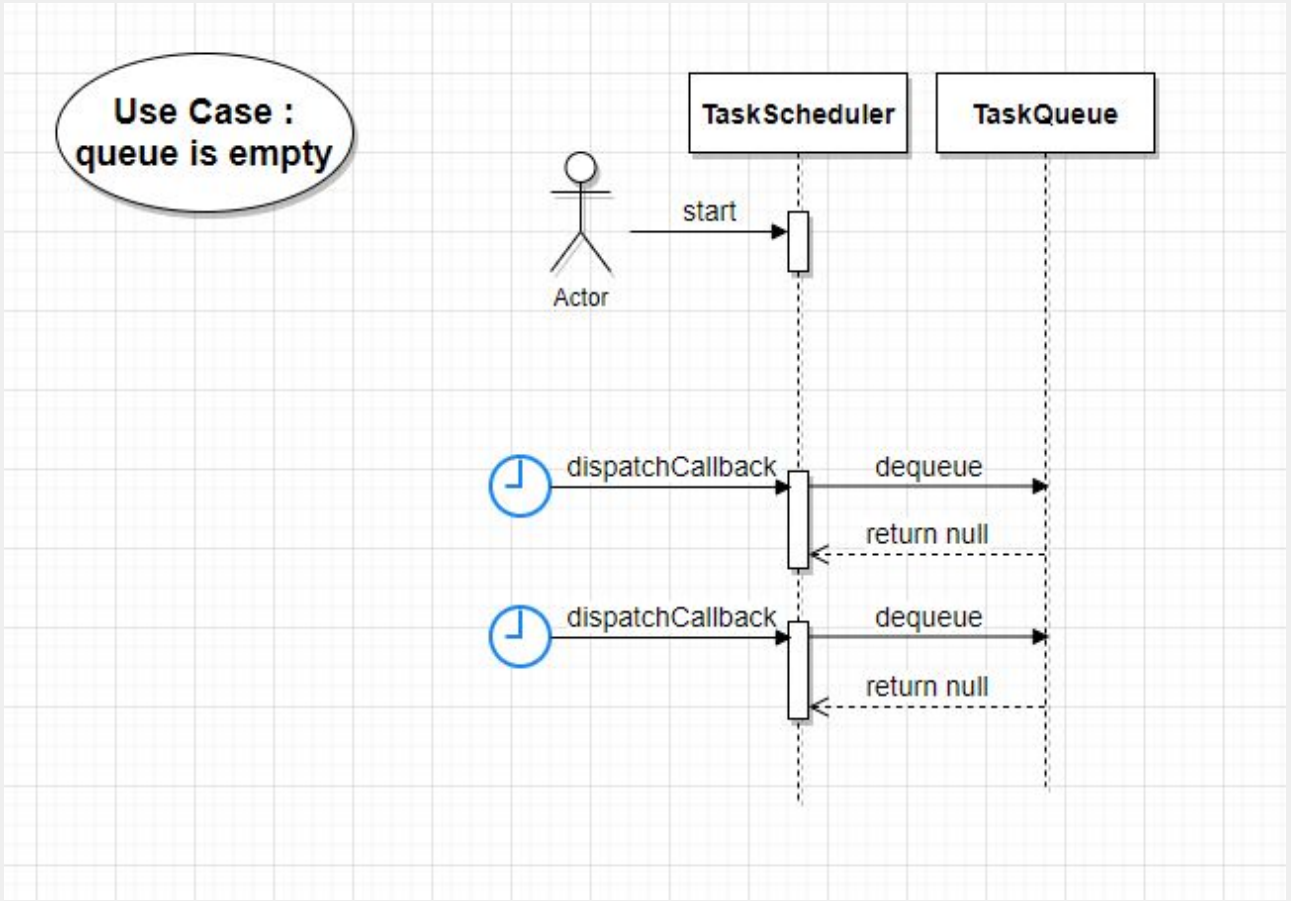
- The motivation for this system
- The specification
- The project structure

Block diagram



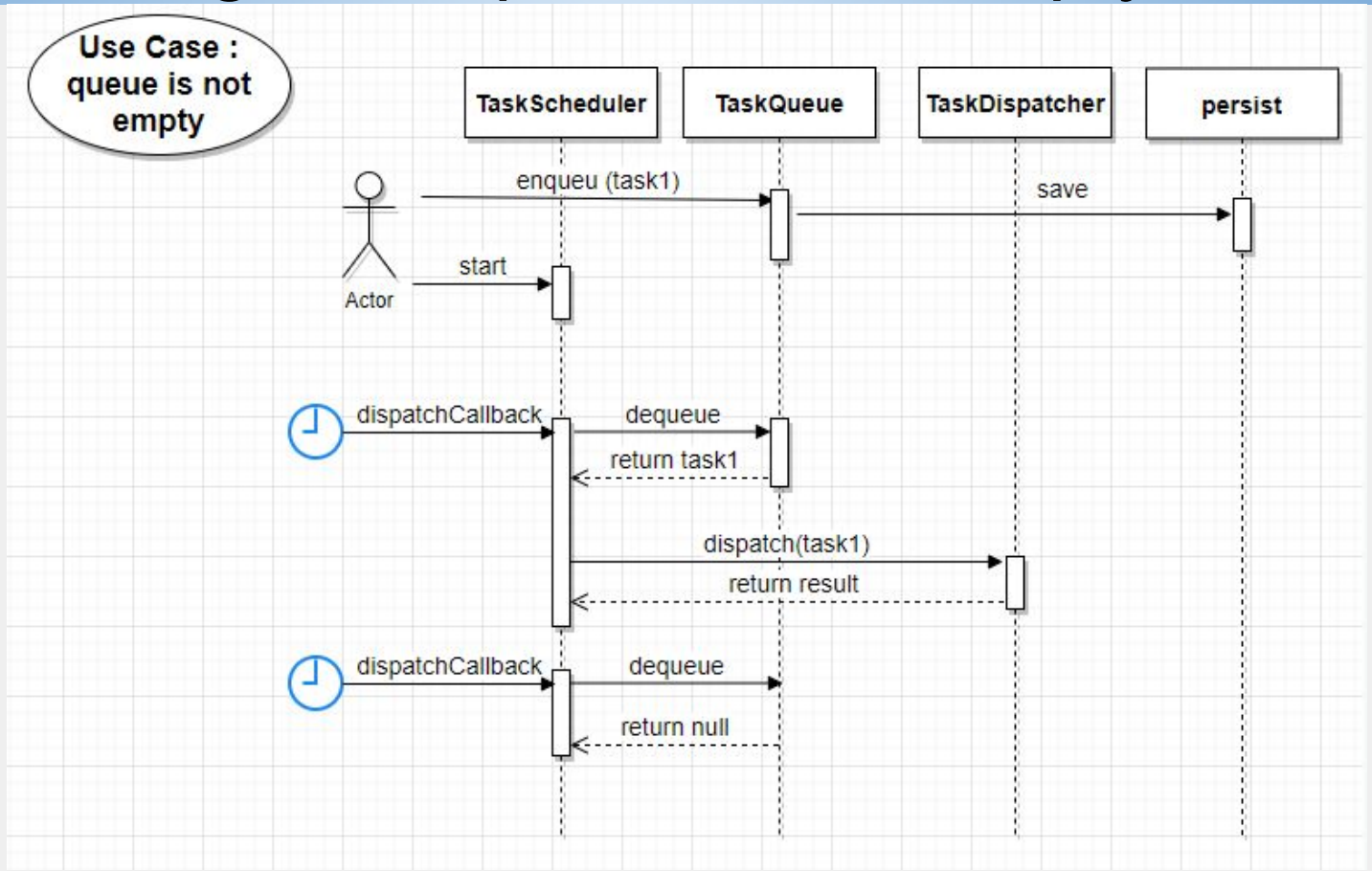
http://nathankrasney.com/

Sequence diagram - queue is empty



http://nathankrasney.com/

Sequence diagram - queue is not empty



http://nathankrasney.com/

Motivation for the system

- I need this for a browser extension which automate linkedin operations : send message , process poll , process post
- Do you know event queue node \ browser ? it has some similarities but far from being the same
- Why Interval

Specification

- Add new task (operation)
- Dispatch tasks in order (FIFO) in series once every period and allow to access the result
- Recover from app down

Section 4

Implement unit tests of pure logic code - TaskQueue

Introduction

- Copy from starter to final
- From simple function to more complex logic
- Which module to choose - sequence diagram
- event queue - queue is empty :
    a. length
    b. dequeue
- event queue - queue is not empty .
    a. code changes : remark save \ load
    b. enqueue
    c. enqueue \ dequeue
    d. length
- Did we cover all code

http://nathankrasney.com/

Section 5

Code coverage \ coverage test

Introduction

- What is code coverage \ coverage test
- Do we need 100% code coverage
- Famous code coverage packages
- How to install it
- How to use it
- Can it reveal a problem in TaskQueue code using code coverage

http://nathankrasney.com/

What is code coverage

- Code coverage is a software metric for measuring the percentage of code that is executed by test cases during software testing.
- It helps identify untested areas in the code, potentially revealing bugs or logic errors.
- High code coverage is desirable, but it does not guarantee the absence of bugs; comprehensive test cases are essential for reliable software.
- Code coverage and coverage test are used with the same meaning

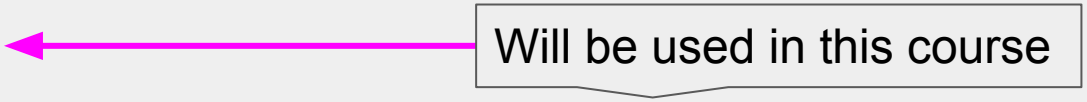
http://nathankrasney.com/

Do we need 100% code coverage

- 100% code coverage is a desirable goal but not always practical or necessary.
- Prioritizing critical areas and error-prone sections for testing is more effective.
- Comprehensive testing and a balanced approach to code coverage are essential for software quality.

Famous code coverage packages

- istanbul
- v8



Section 6

Implement unit tests of pure logic code - TasqDispatcher

Introduction

- Which module to choose : sequence diagram
- Test dispatch ok status
- Test dispatch throw
- Test dispatch ok value
- Filter only this file
- Did we cover all code

Section 7

Introduction to unit testing of code with side effect

Section Intro

- What is a side effect
- What is a mock
- Motivation for mocks
- API
- Modules
- Mock vs sociable test

http://nathankrasney.com/

What is a side effect

Any interaction with the external world with respect to your unit

- Modifications to global variables e.g. console , localStorage
- I/O operations : file ,
- Network communication : axios , fetch
- Database updates
- UI interactions (DOM)

What is a mock

- A "mock" refers to a simulated or fake object that is created to mimic the behavior of a real object or component within a software system.
- Mock objects are typically used in unit testing to isolate and test specific parts of the code.

Motivation for mocks

- **Isolation**: Mocks enable the isolation of units or components being tested from their dependencies, ensuring that tests focus solely on the behavior of the unit under examination.
- **Unavailable Dependencies**: Mocks are useful when certain dependencies are unavailable or inaccessible during testing, allowing developers to simulate their behavior and proceed with testing without relying on their actual availability.
- **Performance Optimization**: By replacing real dependencies with mock objects that provide fast and predictable responses, testing cycles can be optimized, leading to faster and more efficient testing processes.
- **Unit test** : super useful because we want to test our unit isolated from side effect (network,db,file system….) and this can be done with mock

http://nathankrasney.com/

API

- spyOn() - spy
- fn() - replace functionality
- mock() - module
- Jsdom - dom
- Mock timers

Modules

- Persist - jsdom
- Task Queue - use persist
- TaskScheduler - mock timers , use TaskQueue \ TaskDispatcher

Mock vs sociable test

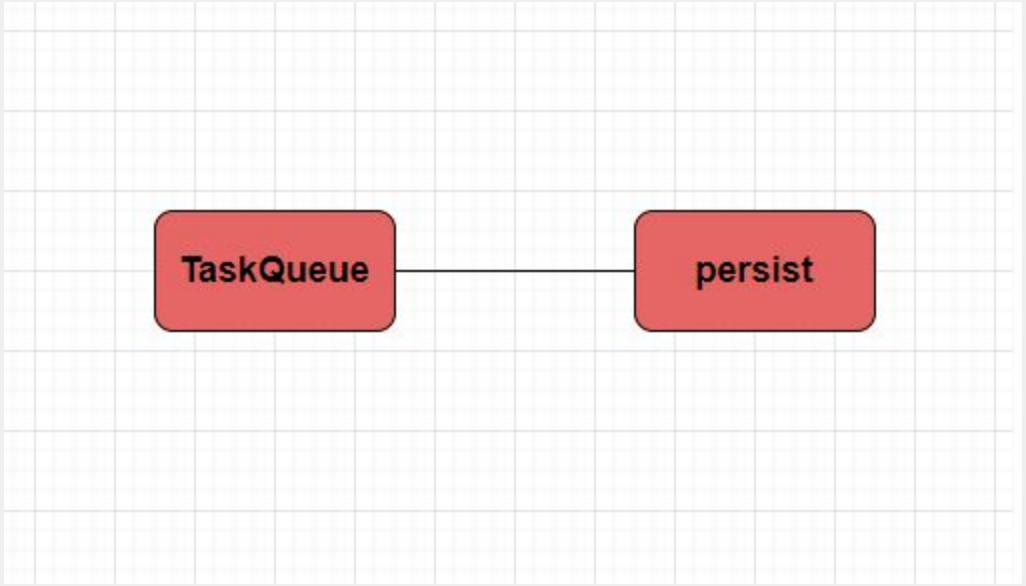
- Mock replace the module we depend on
- Sociable test use the module we depend on (halfway to integration)

http://nathankrasney.com/

Section 9

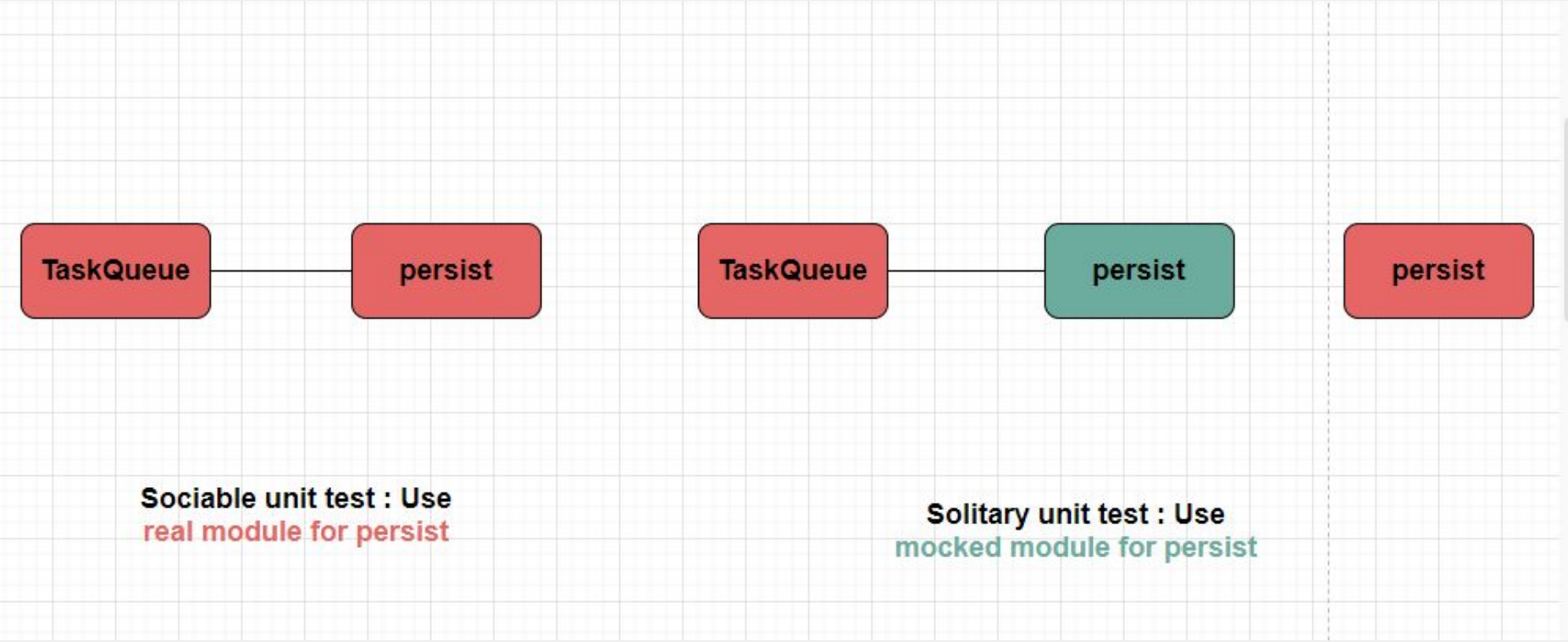
unit test of TaskQueue with persist module interaction

TaskQueue interact with persist



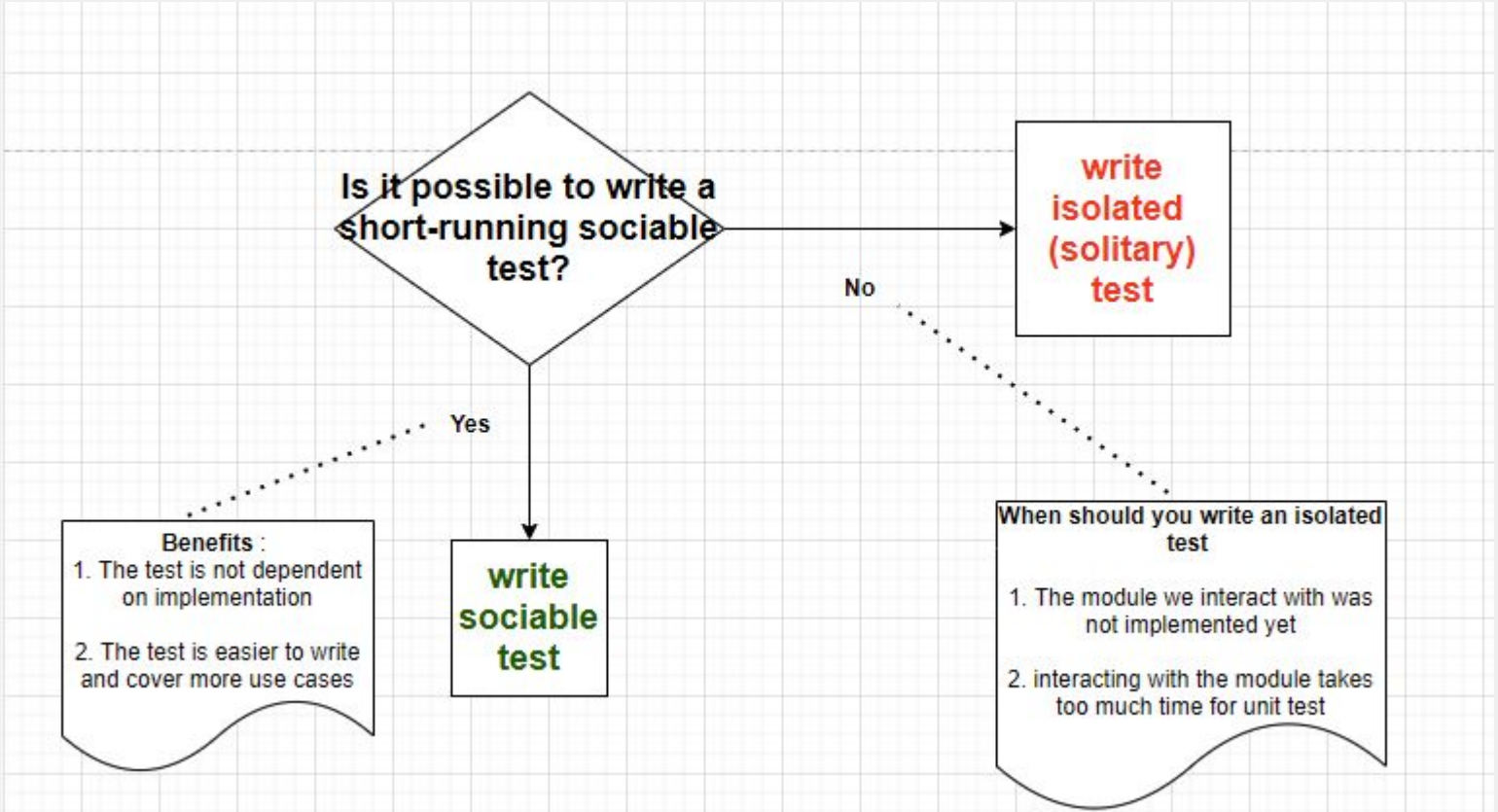
http://nathankrasney.com/

Unit test of module TaskQueue



http://nathankrasney.com/

Should you create isolated or sociable test

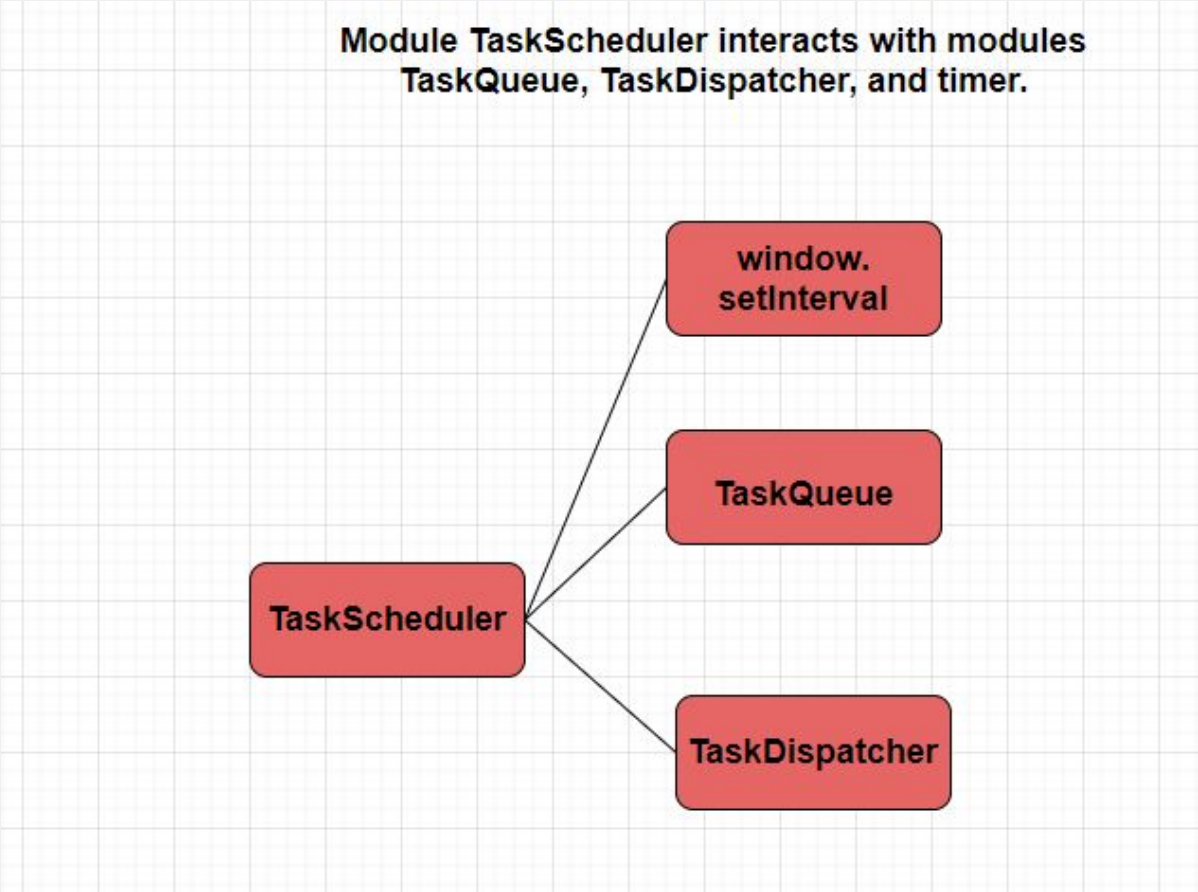


http://nathankrasney.com/

Section 10

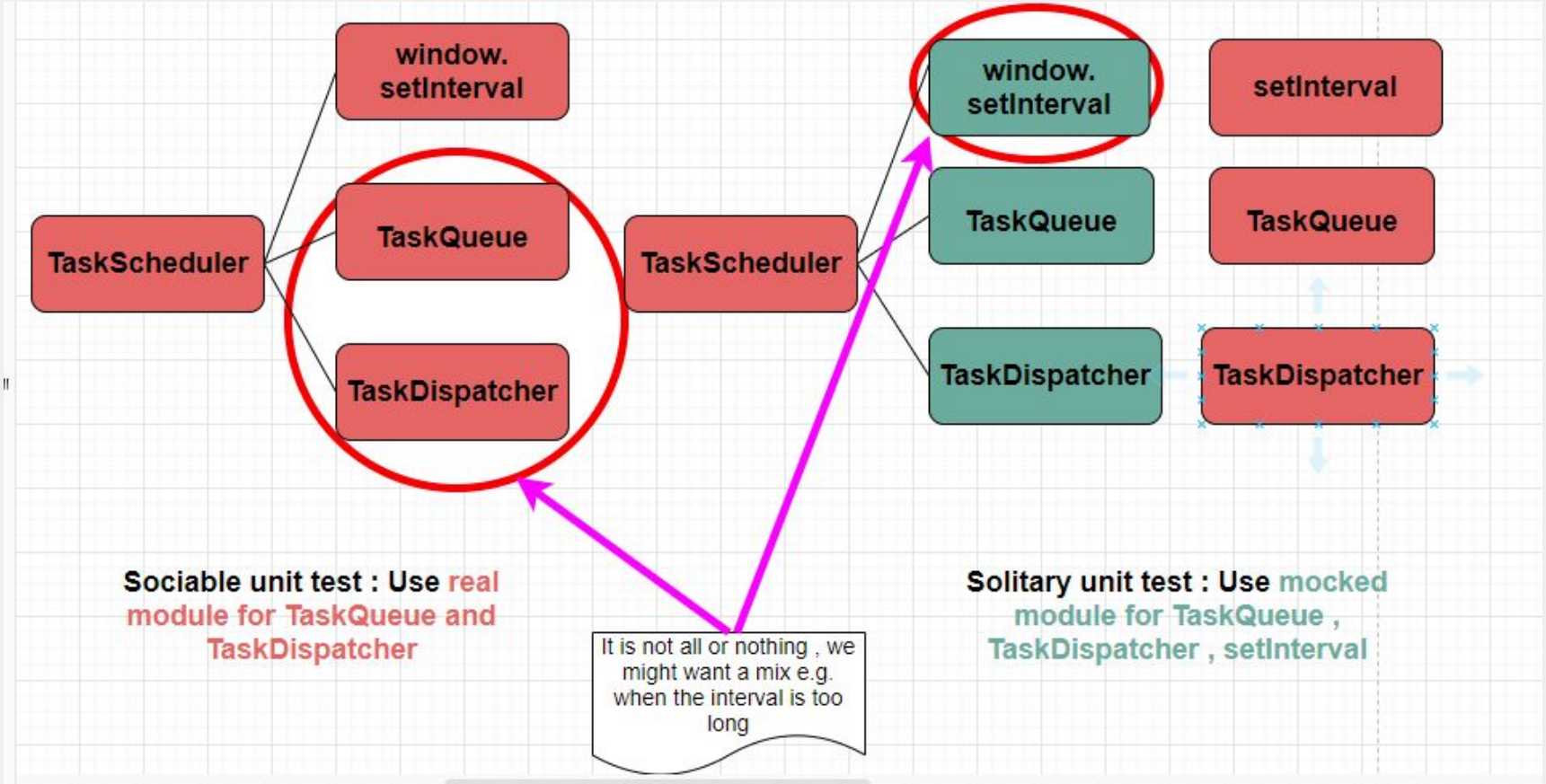
unit test of TaskScheduler with fake timer and module interaction

Module interaction of TaskScheduler



http://nathankrasney.com/

Unit test of TaskScheduler



http://nathankrasney.com/

Section 11

Introduction to frontend unit testing

Where are we now

- At this stage we have first working version of the system - task queue manager. This is logic system with no ui
- It is working with local storage . so can not be used now on the server. But with simple abstraction we can handle this issue.
- We can abstract persist so the system will work in
  - Frontend - Web application : current implementation
  - Frontend - Browser extension : need to change persist implementation
  - Backend - Node server : need to change persist implementation
- We can make more design improvements using refactoring : enum as action type, class instead of function as argument, …
- We can add more features : skip , handle asynchronous dispatch function , …

How to continue

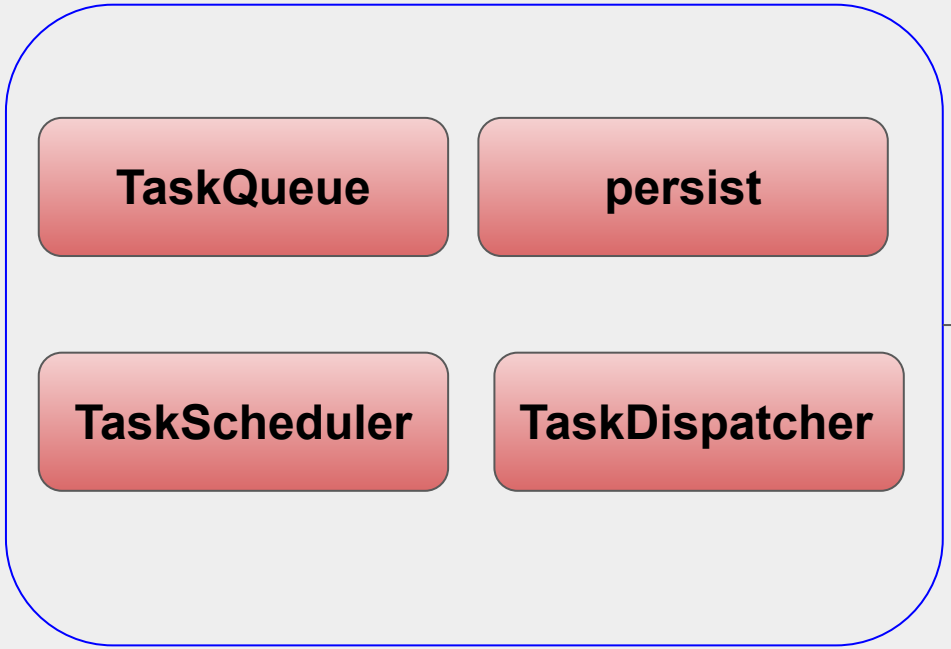
http://nathankrasney.com/

Why UI now

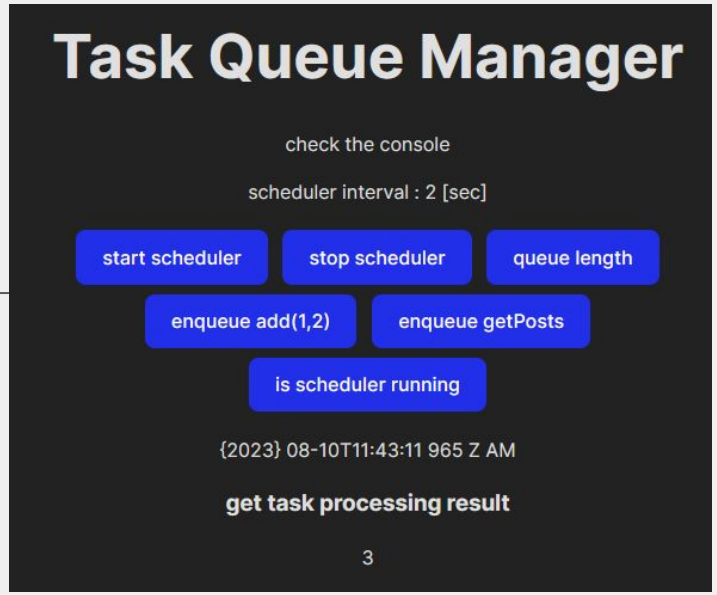
- Our typescript system under lib directory is a logic system
- UI is required
  - This system motivation at the first place is to be used in a browser extension for linkedin automation tool .So in this use case UI is required
  - Even if the system is used only for server side we would like UI to let us play and feel it
  - The UI uses the system without mocks so it may reveal bugs
- Using UI we will get a chance to unit test it

UI and logic

Typescript logic system



UI



http://nathankrasney.com/

UI unit test : jsdom , testing library , react testing library

Typescript logic system

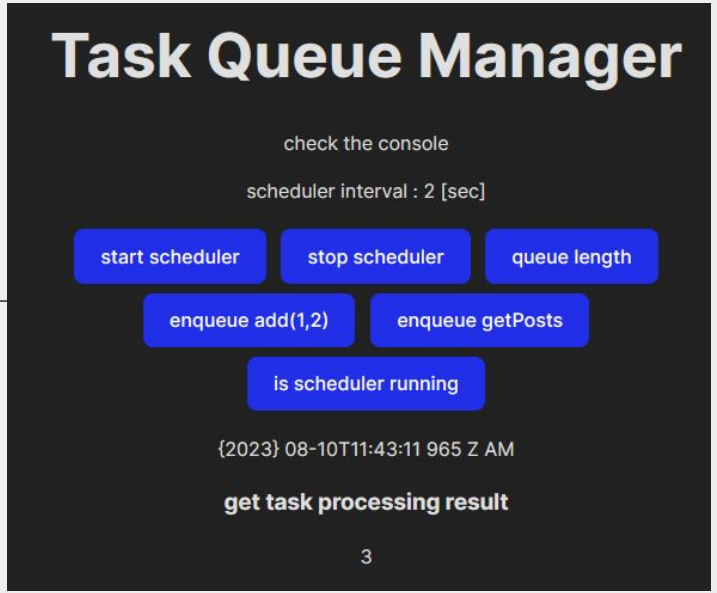
TaskQueue

persist

TaskScheduler

TaskDispatcher

UI



http://nathankrasney.com/

Jsdom , testing library , react testing library

- Jsdom :
  - most simple and straightforward
  - Problematic with find by text and wait for dom element , but possible
  - Used for DOM based application
- Testing library
  - User perspective : you can not access a dom element by element type, you can access by text (as user) and easily
  - Simply API to wait for element in the DOM
  - Used for DOM based application
- React testing library
  - Used for React application

http://nathankrasney.com/

Download updated version

- Download version 0.51 unit-testing-of-a-real-world-ts-system and replace final
- Show how to do this

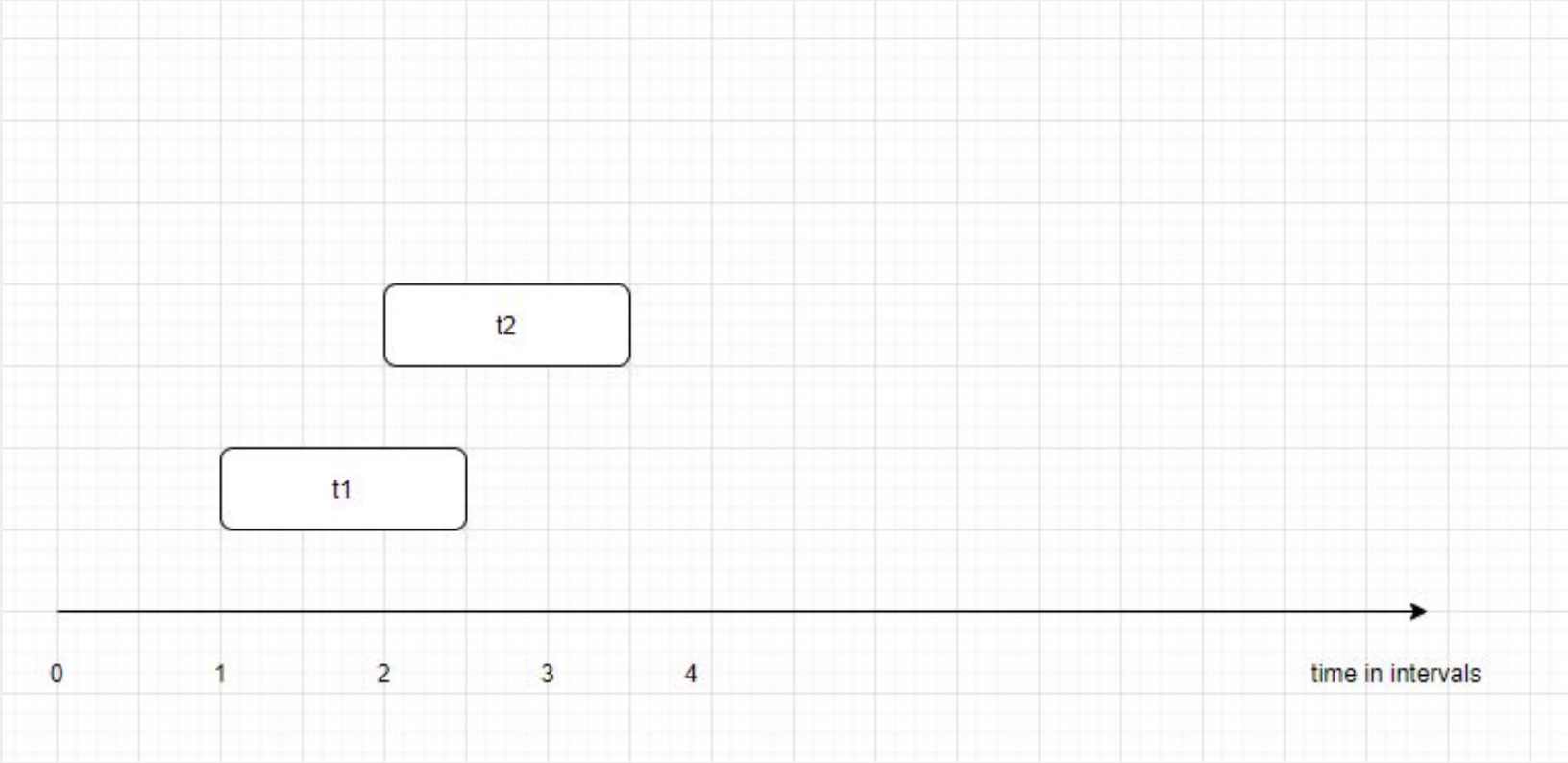
Section 14

Project life cycle : more requirements->code changes \ refactoring->unit test

Section Intro

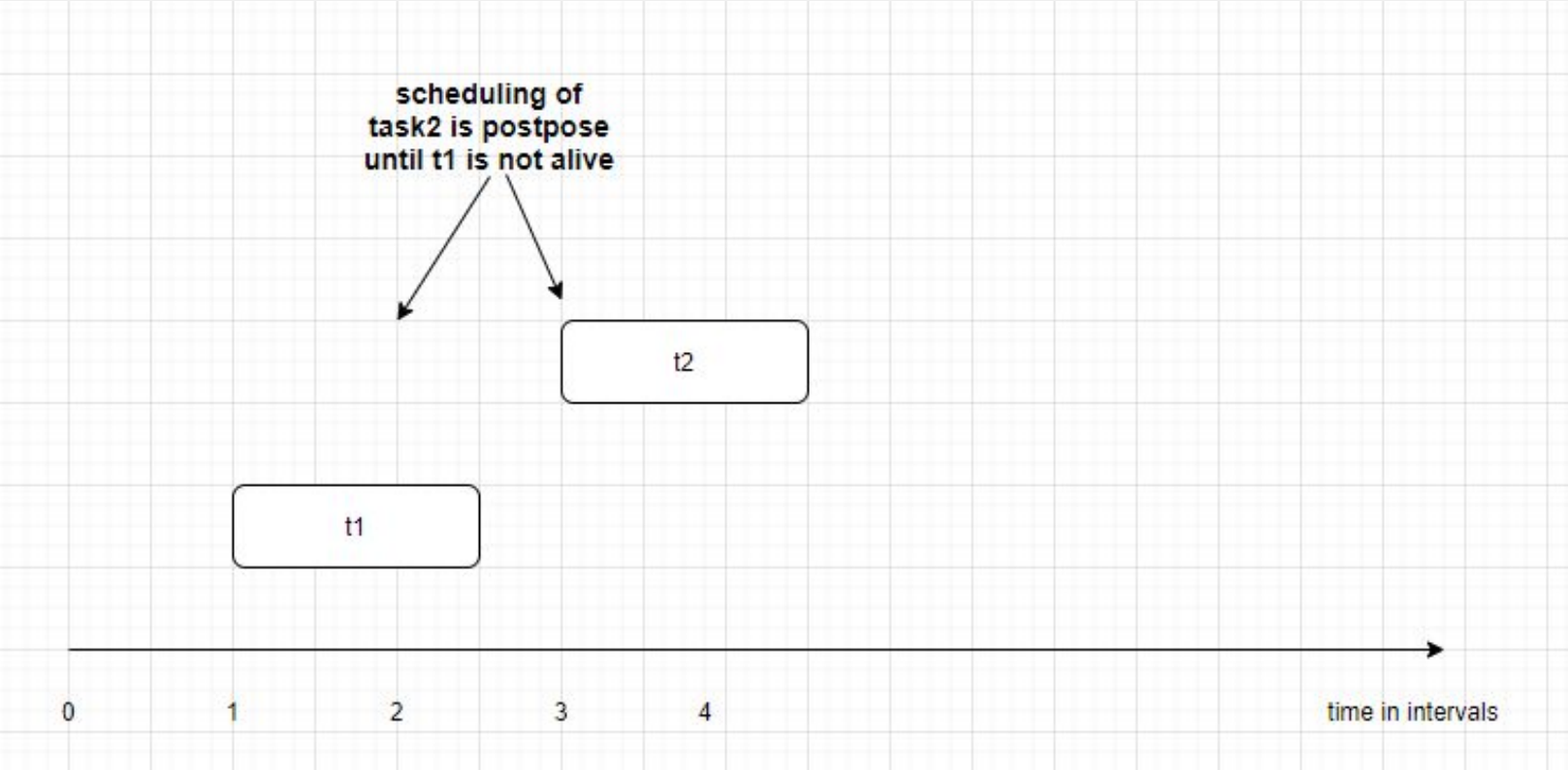
- A software project is dynamic . You start with on set of requirements but along the project life cycle there are always changes
- When requirements are changing also source code is changes and test cases need to be created \ updated
- To be confident that your code works as expected after every change you can use the unit tests. Passing unit test improve your confidence to push to the production stream
- In this chapter we will add new requirements , change source code and verify its quality with existing and new unit tests. This is part of every project life cycle and you have a chance to participate and be part of it

Task overlapping problem



http://nathankrasney.com/

Task overlapping problem solved



http://nathankrasney.com/

Section 15

Advanced typescript for better code

Section intro

- One might think that unit test is only for testing. But unit test allow the developer to be a better developer by producing better code not just less bugs
- We don't have known bugs , but possible bugs may come soon if your code allow it. We want to eliminate future bugs
- You can add code improvement and advanced typescript confidently because we have unit test

Type any

- First introduced in version 0.8 on oct 2012
- In TypeScript, the any type is used to represent a value of any type.
- It essentially allows you to ignore type checking for a particular variable or expression.
- While the any type can be useful in certain situations, it is generally recommended to avoid using it as much as possible.
- The main reason is that using any undermines the benefits of static typing that TypeScript provides
- You can assign anything to a variable type any

Type unknown

- First introduced in version 3.0 on july 2018
- The "unknown" type represents values that are unknown at compile time.
- It is a type-safe counterpart to the "any" type.
- The "unknown" type is designed to ensure that you perform proper type checking and validation before using the value.

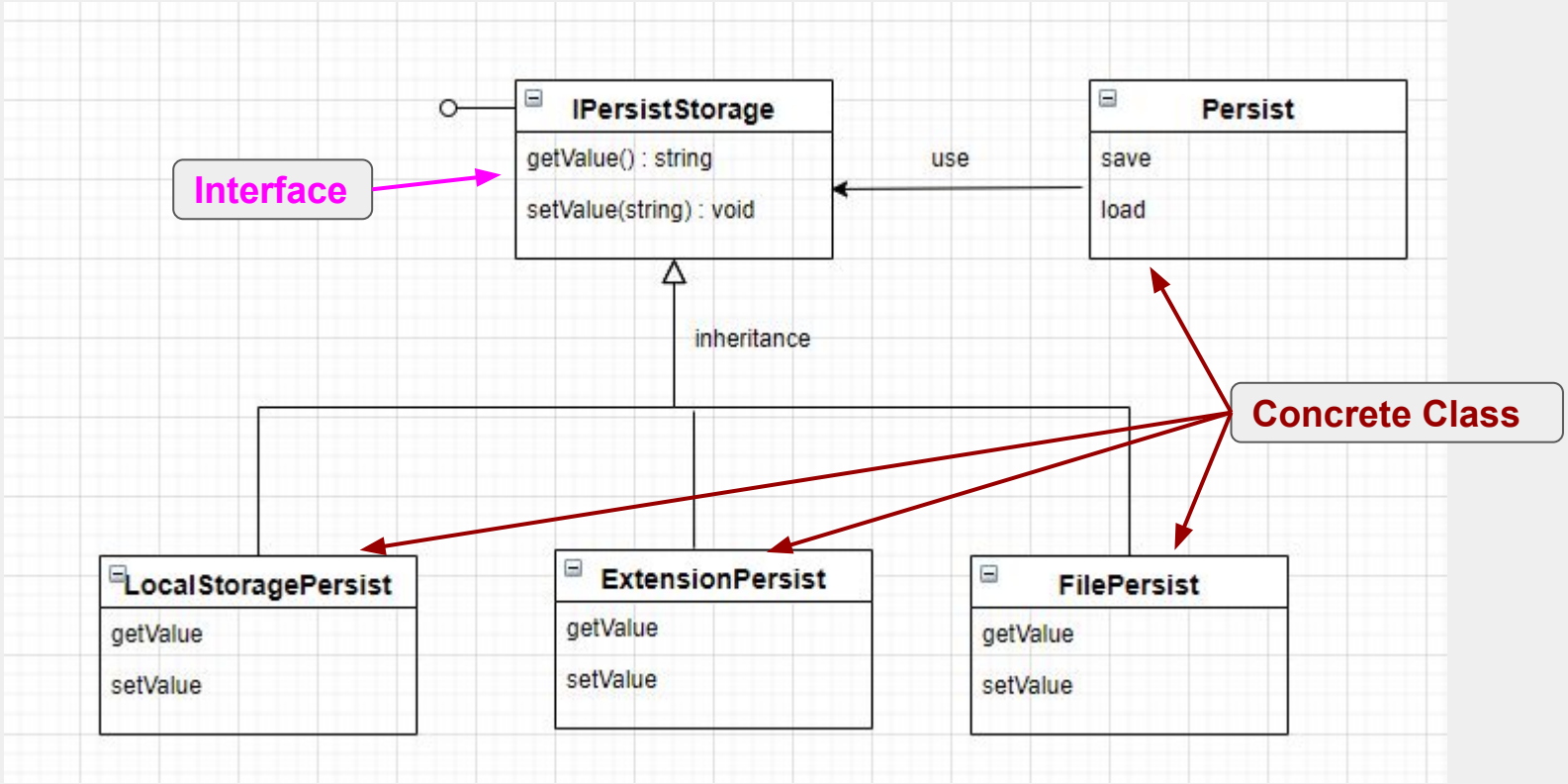
Generic task queue manager system

95% percent of the code is the same. The only difference is the storage. But how to implement this ??

Polymorphism implemented with interface , inheritance and concrete classes

http://nathankrasney.com/

Class diagram for generic persist storage



http://nathankrasney.com/

Section 16
React UI : Unit testing with react testing library - RTL

Add react - version 0.9

- Replaced ui directory
  - directories common (TaskQueueSystemForUi), vanilla , react
  - Important files : vanilla/main.ts , react/main.tsx
- Install : @vitejs/plugin-react react react-dom
- Install on dev : @types/react @types/react-dom
- tsconfig.json : Add "jsx": "react" in compilerOptions
- Update vite.config.ts : react plugin
- Index.html point to react/main.tsx / vanilla/main.ts. One is commented
- Invoke UI with npm run dev and toggle comment
- Small import changes
- Add OnDispatchResult type
- Bug in the ui - stop scheduler not working

The problem

- You want to write maintainable tests for your React components.
- As a part of this goal, you want your tests to avoid including implementation details of your components and rather focus on making your tests give you the confidence for which they are intended.
- As part of this, you want your testbase to be maintainable in the long run so refactors of your components (changes to implementation but not functionality) don't break your tests and slow you and your team down.

http://nathankrasney.com/

The solution

- The React Testing Library is a very light-weight solution for testing React components. It provides light utility functions on top of react-dom and react-dom/test-utils, in a way that encourages better testing practices. Its primary guiding principle is:
- The more your tests resemble the way your software is used, the more confidence they can give you.
- So rather than dealing with instances of rendered React components, your tests will work with actual DOM nodes. The utilities this library provides facilitate querying the DOM in the same way the user would
- This library encourages your applications to be more accessible and allows you to get your tests closer to using your components the way a user will, which allows your tests to give you more confidence that your application will work when a real user uses it.

http://nathankrasney.com/

React testing library setup

- @testing-library/react

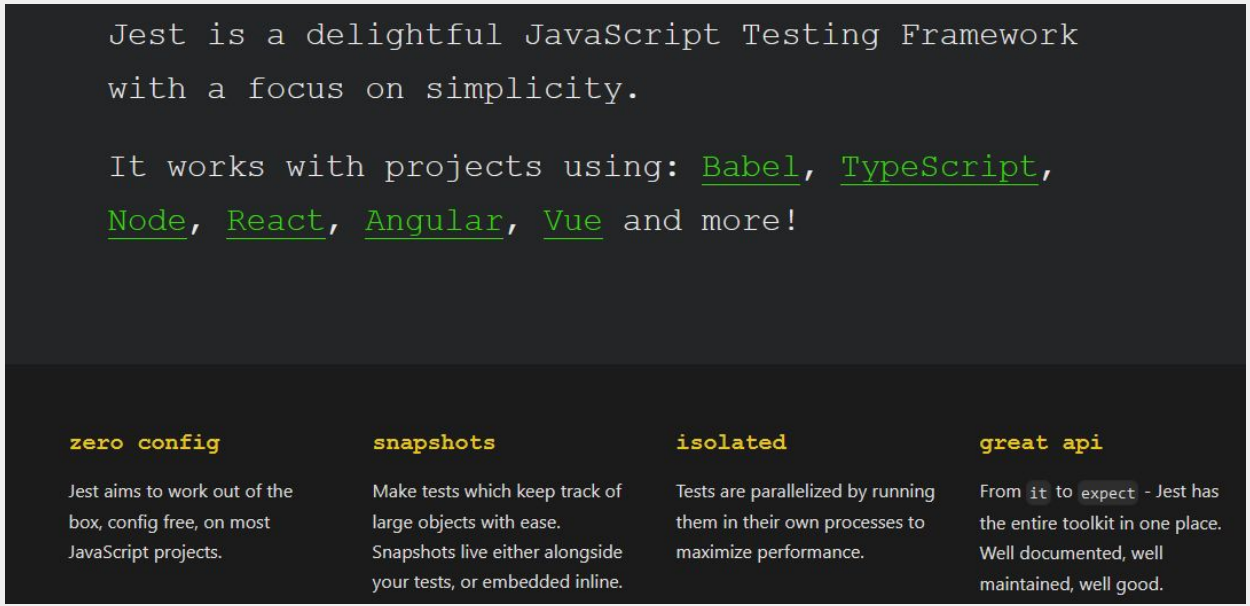
Section 17
Unit tests with jest

Section intro

- Vitest is great but jest is very popular
- Jest
- Jest and vitest api
- Jest setup for our project
- Use vitest and jest on the same project
- Tweak vitest test files to fit jest
- Run tests with jest
- Compare vitest and jest run time

What is jest

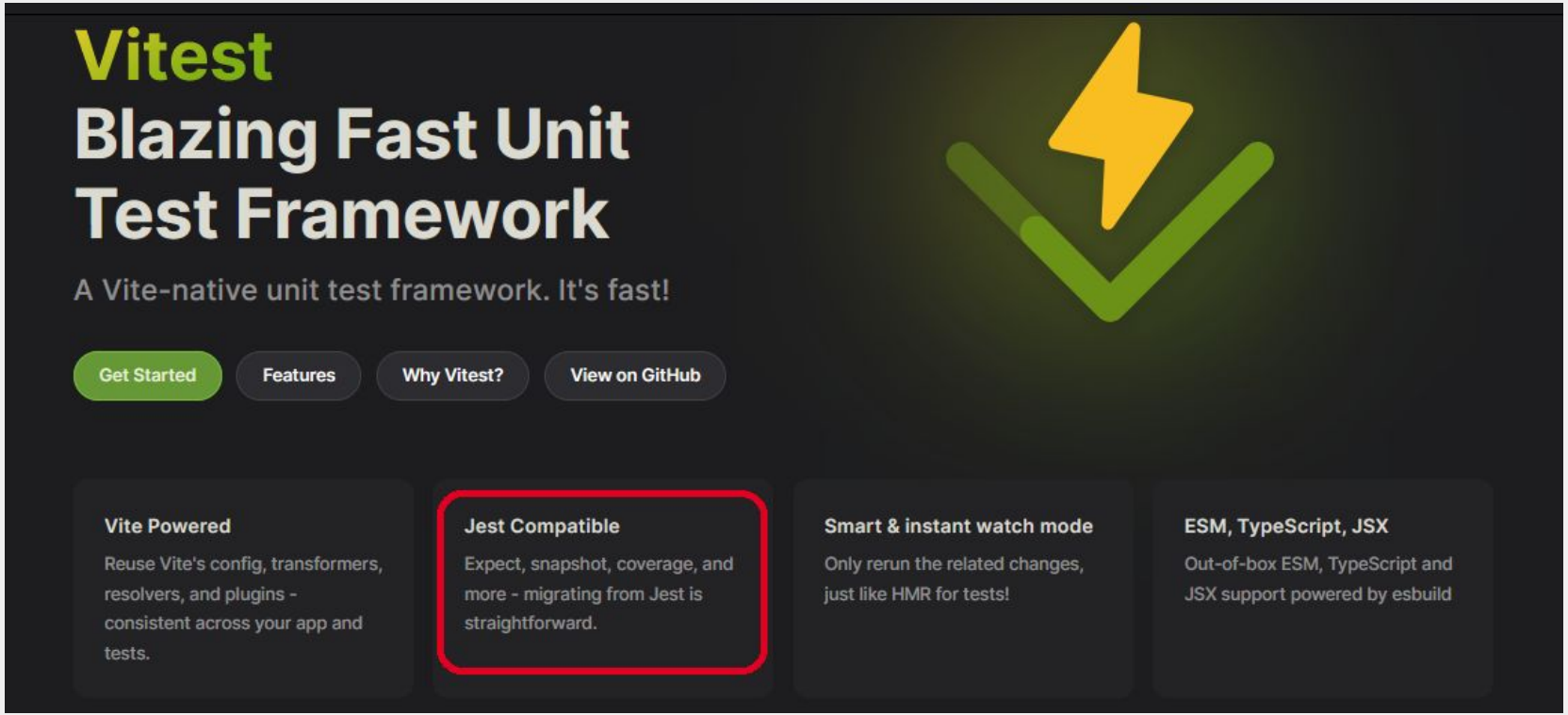
- **Test runner** - test() , ….
- **Utility for mocking** - mock() , fn() , spyOn()
- **Assertion library** - expect() , toBe() , ….



http://nathankrasney.com/

Jest and vitest API

- One of vitest design goals is to use the same API as jest
-



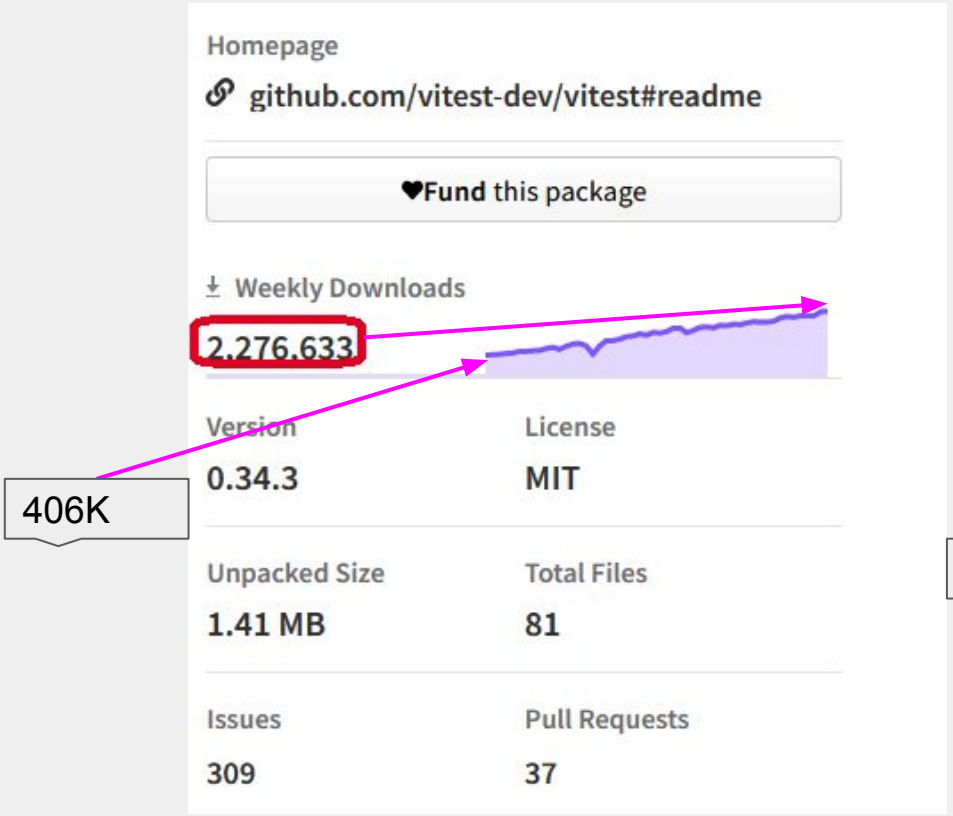
http://nathankrasney.com/

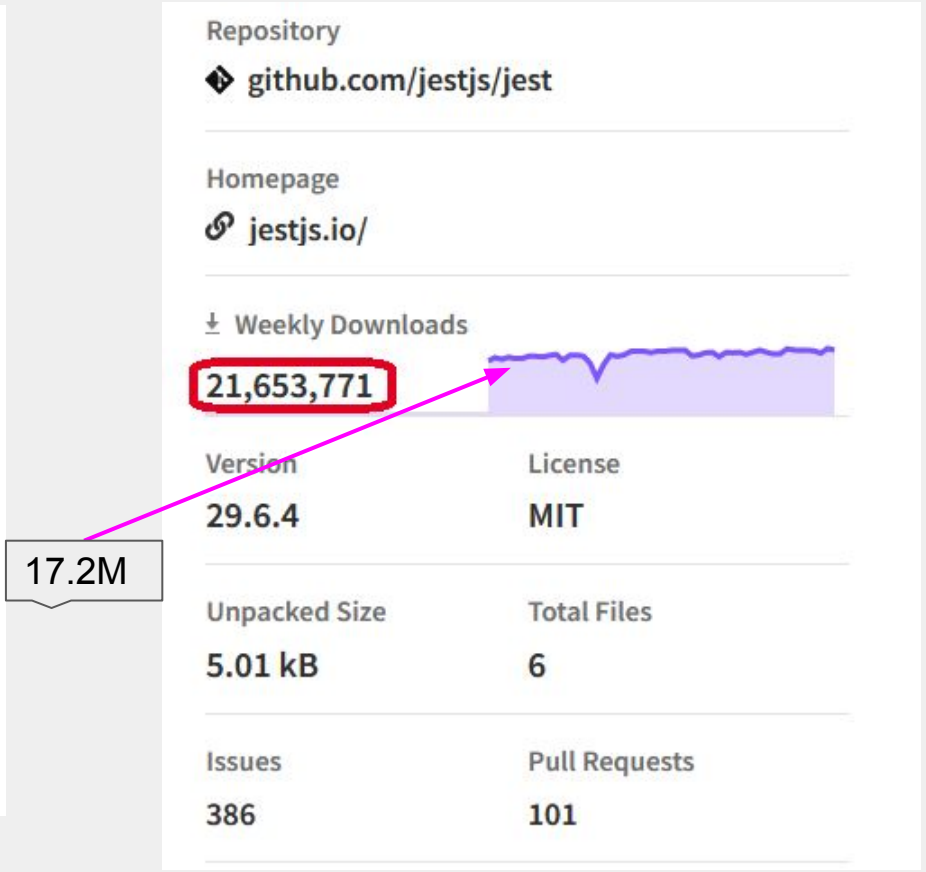
Jest vs vitest

- Popularity : 2.3M vitest , 21.6M → jest
- popularity slope :vitest x5 , jest ~20% → vitest
- Ease of setup and use working with ts and es module → vitest
- Maturity : 0.34 vitest , 29.6 jest → jest
- Test run time : 93, 25.8 , 22,2 , 22.57 vitest , 17,21.9,21.4,21.3→ jest
- Watch mode run time 24.3 ,22.7 vitest 24.1,23,8 →same
- Coverage run time : 17.6 , 16.39 vitest ,22.5,18.6 jest → vitest

http://nathankrasney.com/

vitest vs Jest popularity - sep 2023





http://nathankrasney.com/

vitest vs Jest - run time

```
 Test Files  8 passed (8)
      Tests  71 passed (71)
   Start at  08:52:49
   Duration  93.53s (transform 750ms, setup 22.26s, collect 38.55s, tests 18.86s, environment 91.27s, p
repare 9.74s)
```

vitest First run

```
 Test Files  8 passed (8)
      Tests  71 passed (71)
   Start at  08:57:26
   Duration  17.48s (transform 295ms, setup 1.59s, collect 1.92s, tests 18.39s, environment 5.86s, prep
are 1.46s)
```

vitest Second run

```
Test Suites: 8 passed, 8 total
Tests:       71 passed, 71 total
Snapshots:   0 total
Time:        17.726 s, estimated 19 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Jest first

```
Test Suites: 8 passed, 8 total
Tests:       71 passed, 71 total
Snapshots:   0 total
Time:        13.951 s
Ran all test suites.
```

Jest second

http://nathankrasney.com/

vitest vs Jest - watch mode (add space to persistence.ts)

```
 Test Files  6 passed (6)
      Tests  58 passed (58)
   Start at  09:17:33
   Duration  21.45s
```

vitest

```
Test Suites: 8 passed, 8 total
Tests:       71 passed, 71 total
Snapshots:   0 total
Time:        19.711 s
Ran all test suites related to changed files.
```

jest

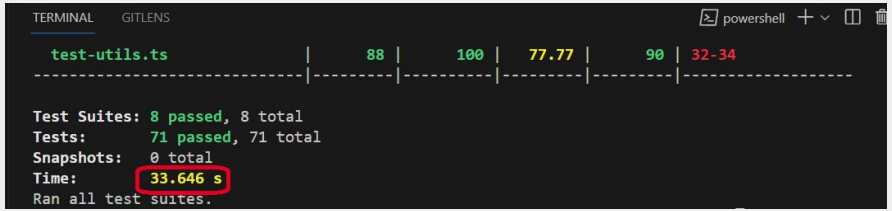
http://nathankrasney.com/

vitest vs Jest - coverage

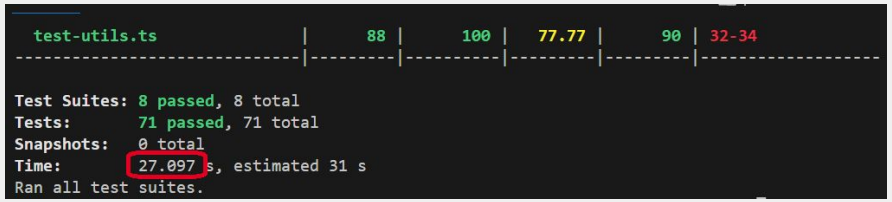
```
 Test Files  8 passed (8)
      Tests  71 passed (71)
   Start at  09:28:43
   Duration  16.69s (transform 1.83s, setup 1.83s, collect 7.50s, tests 18.73s, environment 6.44s, prepa
re 2.43s)

 % Coverage report from istanbul
-------------------------------|---------|----------|---------|---------|-------------------
File                           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
```

Vitest first



Jest first

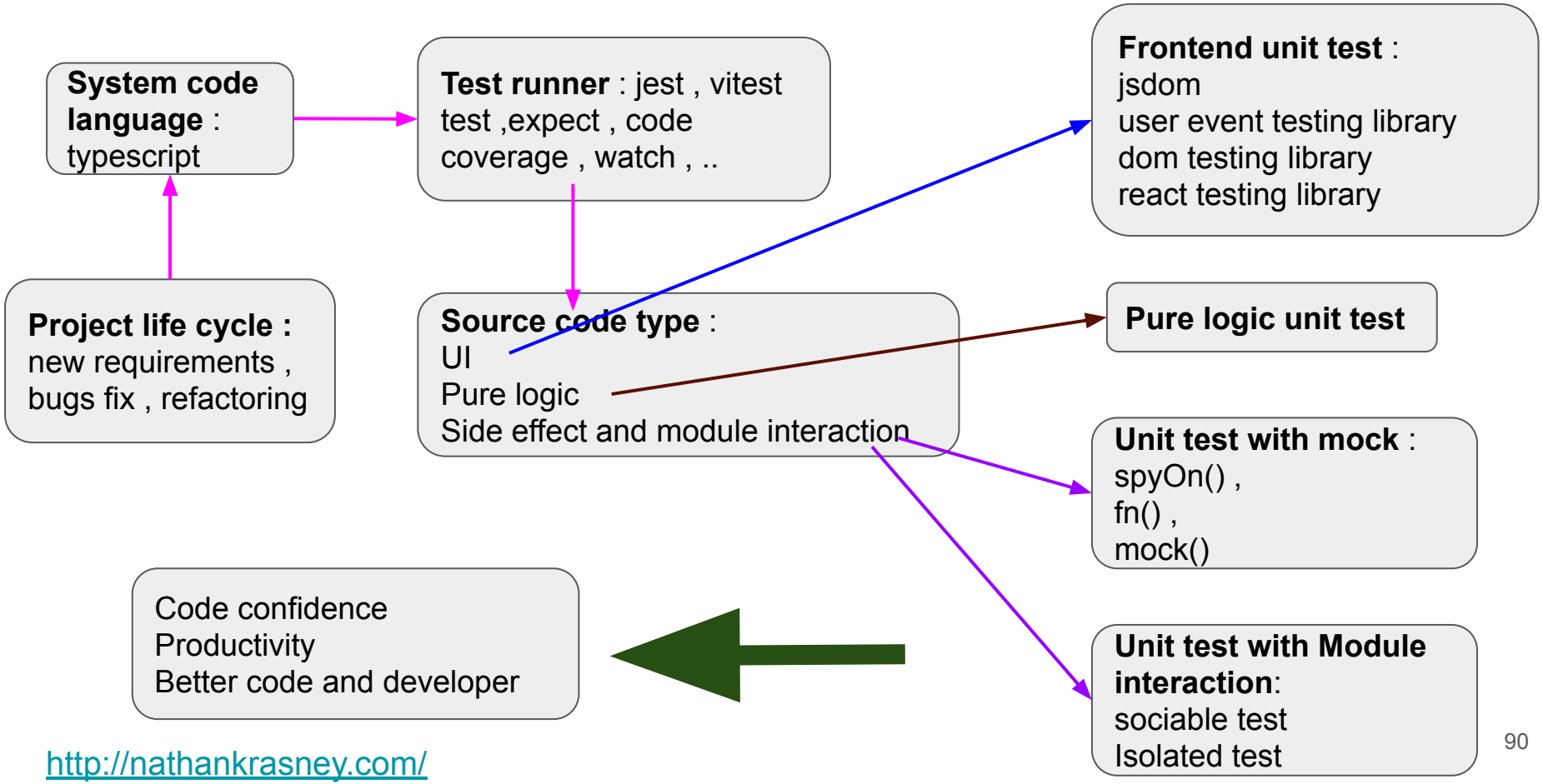


Jest second

http://nathankrasney.com/

Section 18
Where to go from here

Here we are - course summary



http://nathankrasney.com/

Where to go from here - task queue manager

- Generic enum
- UI per dispatch function
- Function as argument problem

Where to go from here - testing

- Integration test - vitest \ jest , jsdom \ testing library
- E2e test : cypress , playwright , puppeteer , …