Enterprise User Interface Development

Steve Kinney — Head of Engineering, Frontend and Developer Tools @ Temporal

This is a course on how to build the infrastructure needed to manage a large code base supported by a team of contributors.

Maybe, you've inherited an older code base.

Maybe, it's been through some "changes in product direction."

Maybe, you're looking to migrate from an older framework to whatever is hot these days.

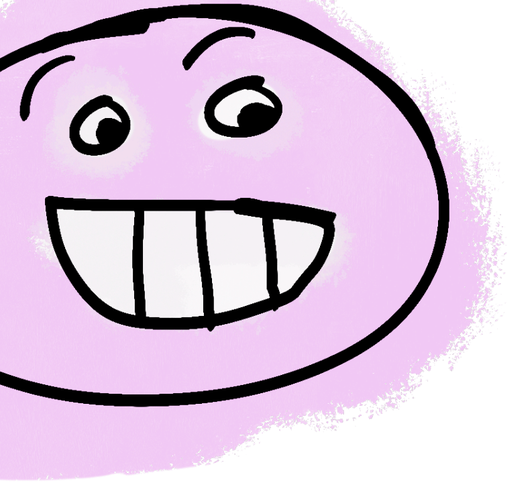
Or, maybe you're breaking ground on a brand new project and you want to get started on the right foot.

This course will show you how to put together an infrastructure that will give you confidence when making changes—big or small.

Oh wait—who even am I?

In which, I do a quick appeal to authority.

- Hi, my name is *Steve*.
- I work on the frontend engineering team at **Temporal**. Originally, I was the **frontend architect** and now I lead a team of engineers.
- Previously, I was the **frontend architect** at **Twilio** and **SendGrid**.
- At some point or another, I've ended up in each of those situations that I just mentioned.



What are we going to cover?

We've got a full agenda, y'all.

- The fundamentals and philosophy of **unit testing** your application.
- Some strategies for **testing your components**.
- How to write browser integration tests with **Playwright**.
- Testing when testing seems hard: **How (and when not) to fake stuff** in your tests (e.g. mocks, stubs, spies, test doubles, etc.).
- I'll try sneak in some tips on to **structure your components and/or application** in a way that makes easy to test and update.
- How to **enforce coding standards** and best practices using the build process.
- How to **build CI/CD processes** using Github Actions.

Some Core Questions

(That may or may not have answers.)

- How do we create **maintainable** code bases? But like, in reality... when they grow.
- Put another way: How to **hold back entropy**?
- What does it even mean to be maintainable?

entropy noun

en·tro·py ˈen-trə-pē

plural **entropies**

1 **thermodynamics** : a measure of the unavailable energy in a closed thermodynamic system that is also usually considered to be a measure of the system's disorder, that is a property of the system's state, and that varies directly with any reversible change in heat in the system and inversely with the temperature of the system

broadly : the degree of disorder or uncertainty in a system

2 **a** : the degradation of the matter and energy in the universe to an ultimate state of inert uniformity

> *Entropy* is the general trend of the universe toward death and disorder.
> – James R. Newman

b : a process of degradation or running down or a trend to disorder

> The deterioration of copy editing and proof-reading, incidentally, is a token of the cultural *entropy* that has overtaken us in the postwar years.
> – John Simon

What does it mean to be maintainable?
(That's like the opposite of *unmaintainable*, right?)

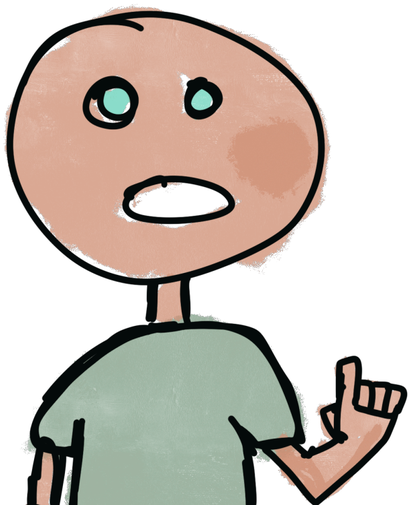
- Well, you should be able to do stuff, right?
- And not just add new stuff on top of it, but like change stuff and **confidently** know you didn't break something, right?
- Maybe you want to be able to **refactor without overwhelming fear**?
- Ideally, you might want to like **update a dependency** at some point.

What about code reviews?

Let’s automate the nitpicking to a robot.

- Code reviews are great, but they **don't always work at scale**.
- Also, **human nature** is a thing.
- It's easy to miss stuff. We all get tired sometimes.
- Sometimes, we feel bad or sometimes we miss stuff because we're tired or there is a lot going on.

It would be nice to be able have some automated systems in place to keep things in a good state.



Components of a Well-Architected UI Application

- **Testing**: Unit, component, integration, end-to-end, smoke tests, health checks.
- **Static analysis**: TypeScript, linting, formatting, etc.
- **Build processes**: Automatically run checks for the items above.
- **Separation of concerns**: Keeping your business logic separate from the view layer.
- **Deployment infrastructure**: Do you even CDN?
- **Design processes**: Are we thinking things out or are we throwing spaghetti at the wall?

Technical Debt and Cognitive Cost

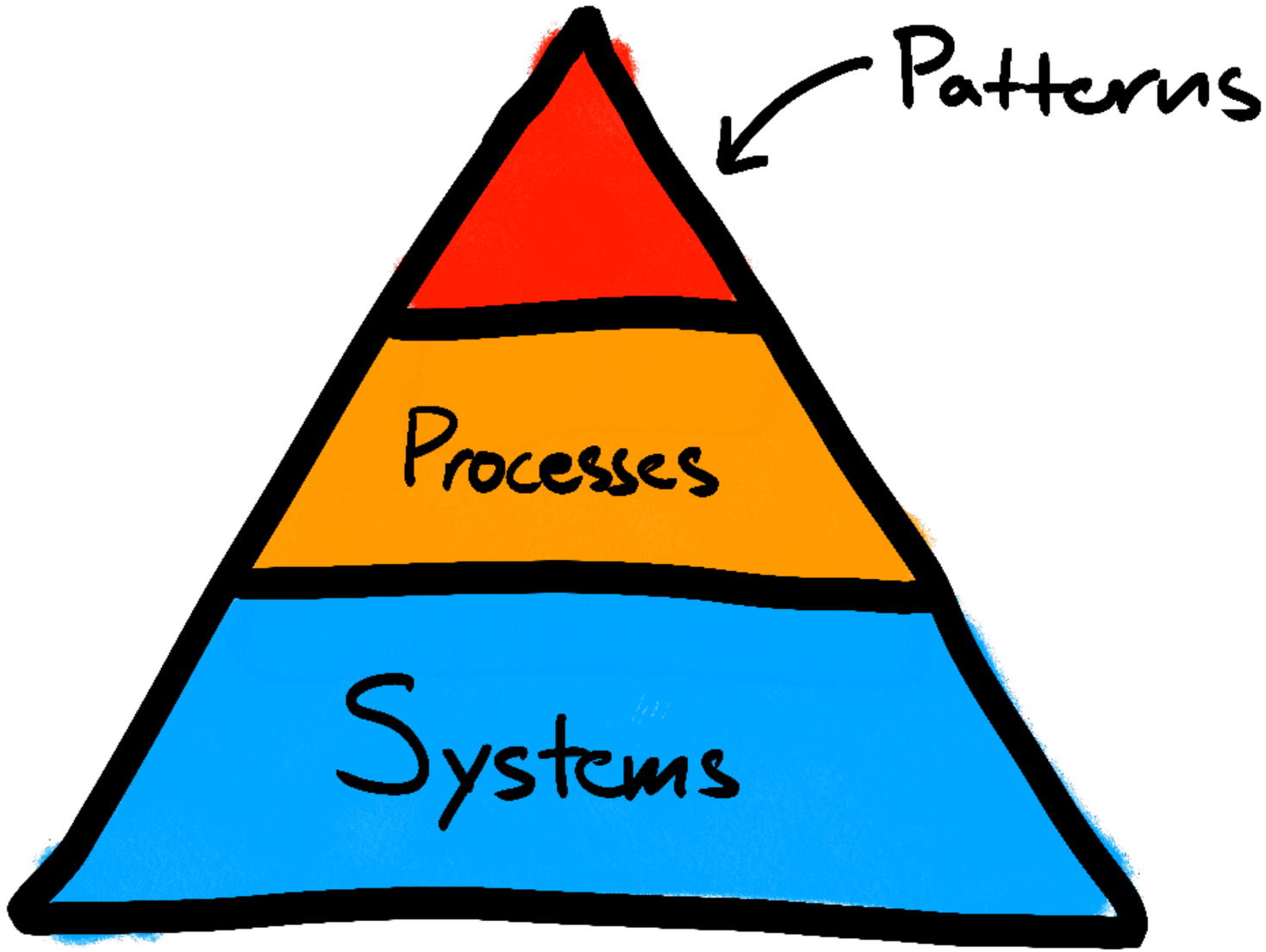
Things can go sideways when it’s hard to understand the code.

- Is there a **common structure**?
- Arguing over style feels pedantic, but it also adds a level of **consistency** that can make you more productive. (That said, you shouldn’t argue about these things.)
- What about **protecting against gotchas**? Or, maybe it's just an element of making sure we're using the patterns we agree upon.

Three Ingredients

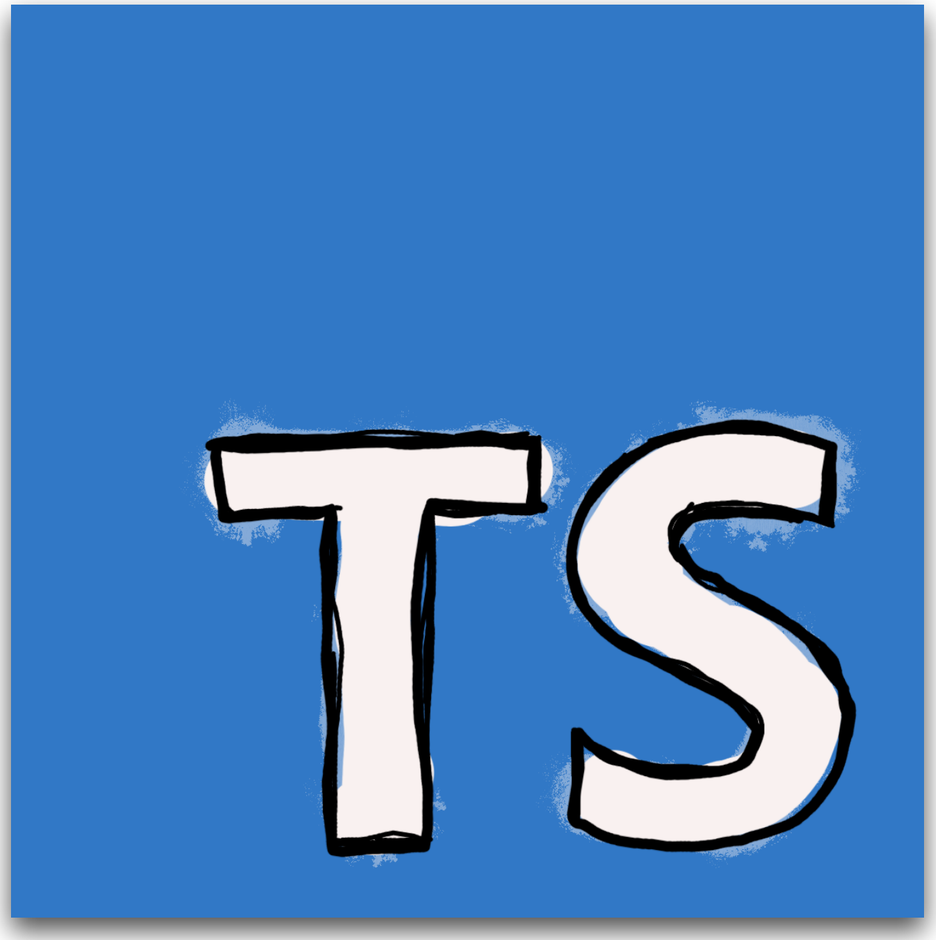
What makes a sustainable architecture?

- **Patterns**: Architecture, state management, abstractions.
- **Processes**: Code reviews, blueprints, design documents.
- **Systems**: Testing infrastructure, static analysis, build systems.



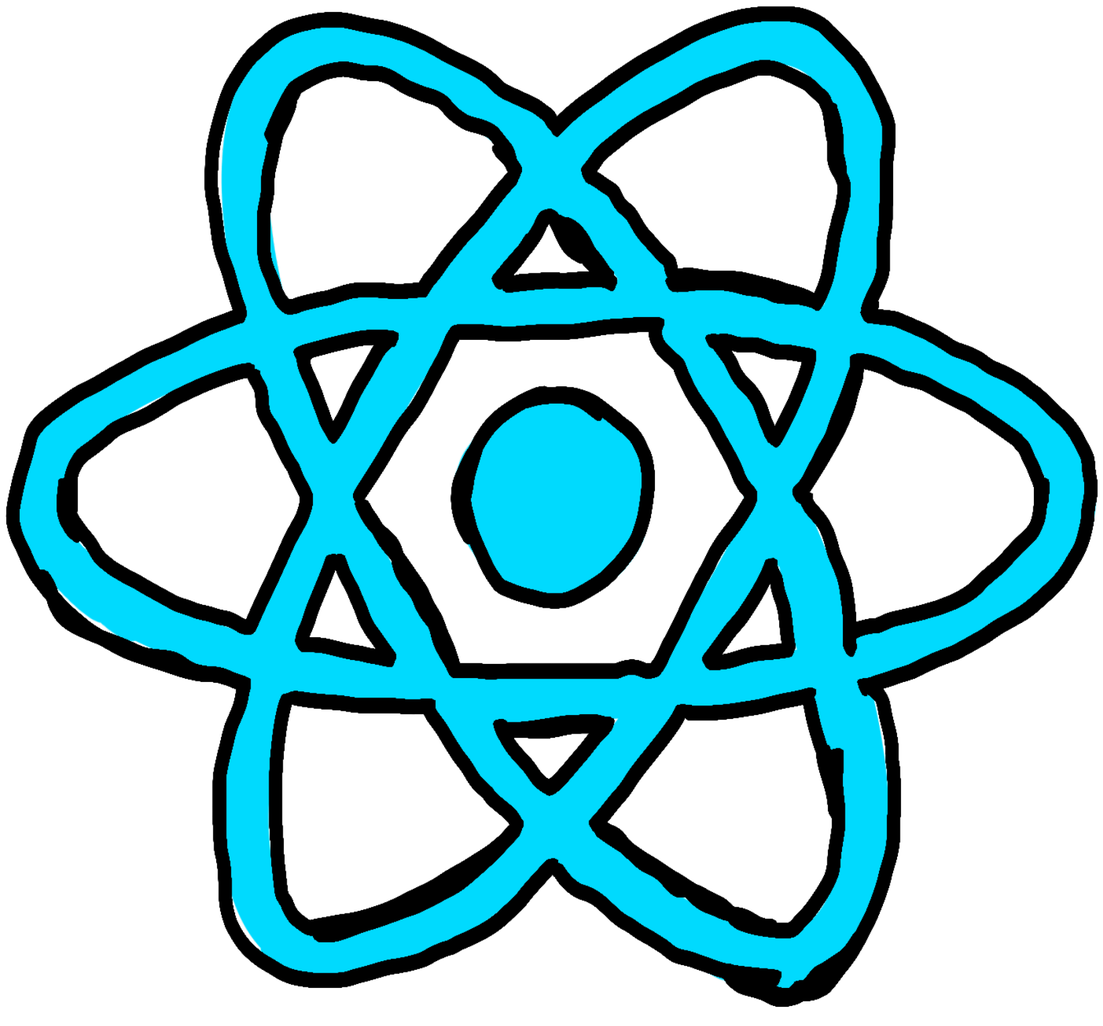
What technologies are we going to use?

I'm going to try to keep this agnostic as possible. (It's not.)



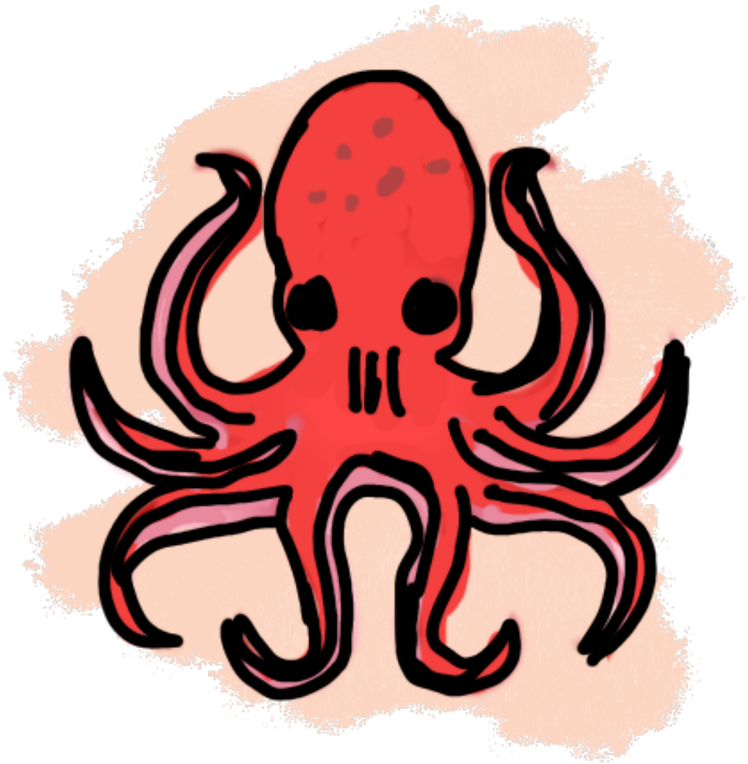
React

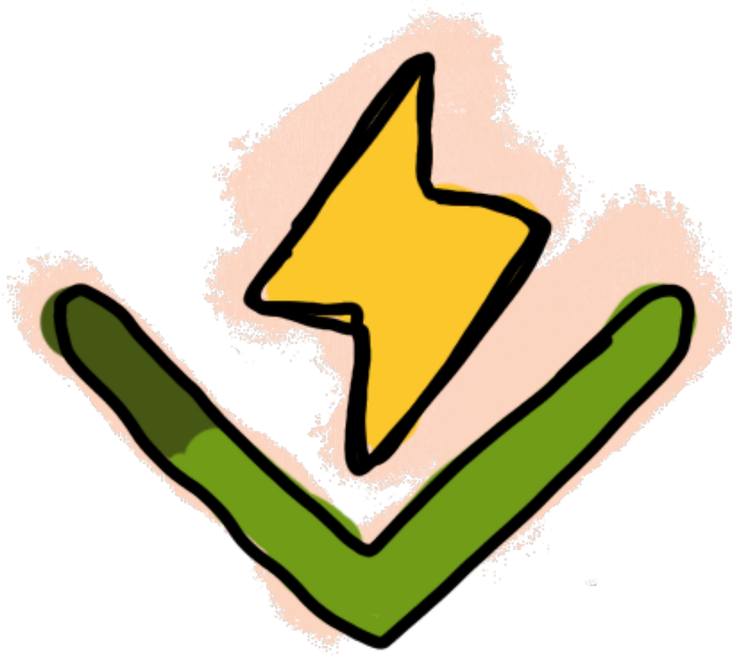
You know it. You may or may not love it. But, you probably know it.

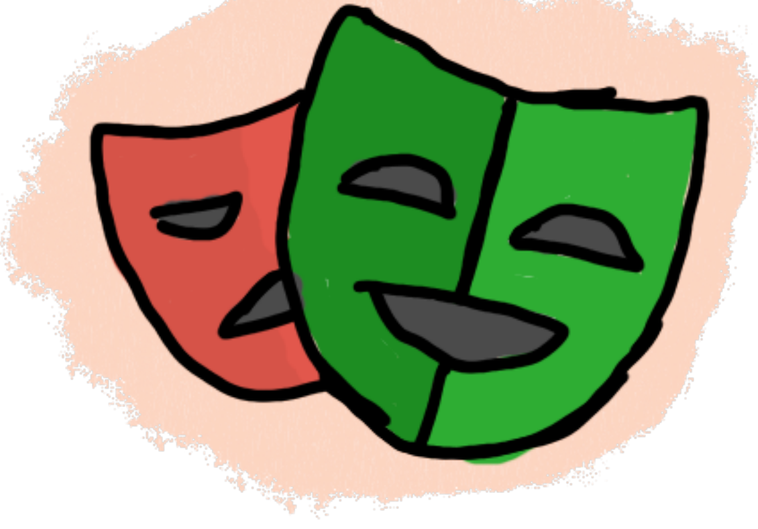


Some Testing Tools

Testing Library, Vitest, Playwright







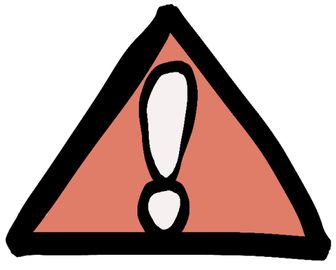
The Ground Rules

It’s important to have standards.

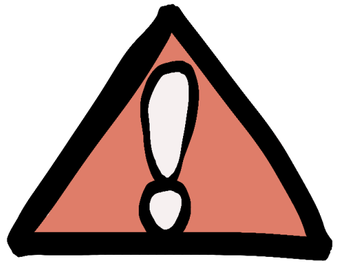
- We’re going to use **TypeScript** for everything all the time and we’re going to use it in **strict mode**.
- We’re going to put our infrastructure into an automated CI process using **Github Actions**.
- We’re going error towards the side of **practicality and reality** over philosophy.

It's hard to talk about maintainability without talking about *testing*.

All of these terms are a bit overloaded, but let's at least level-set on a set of terms during our time together.



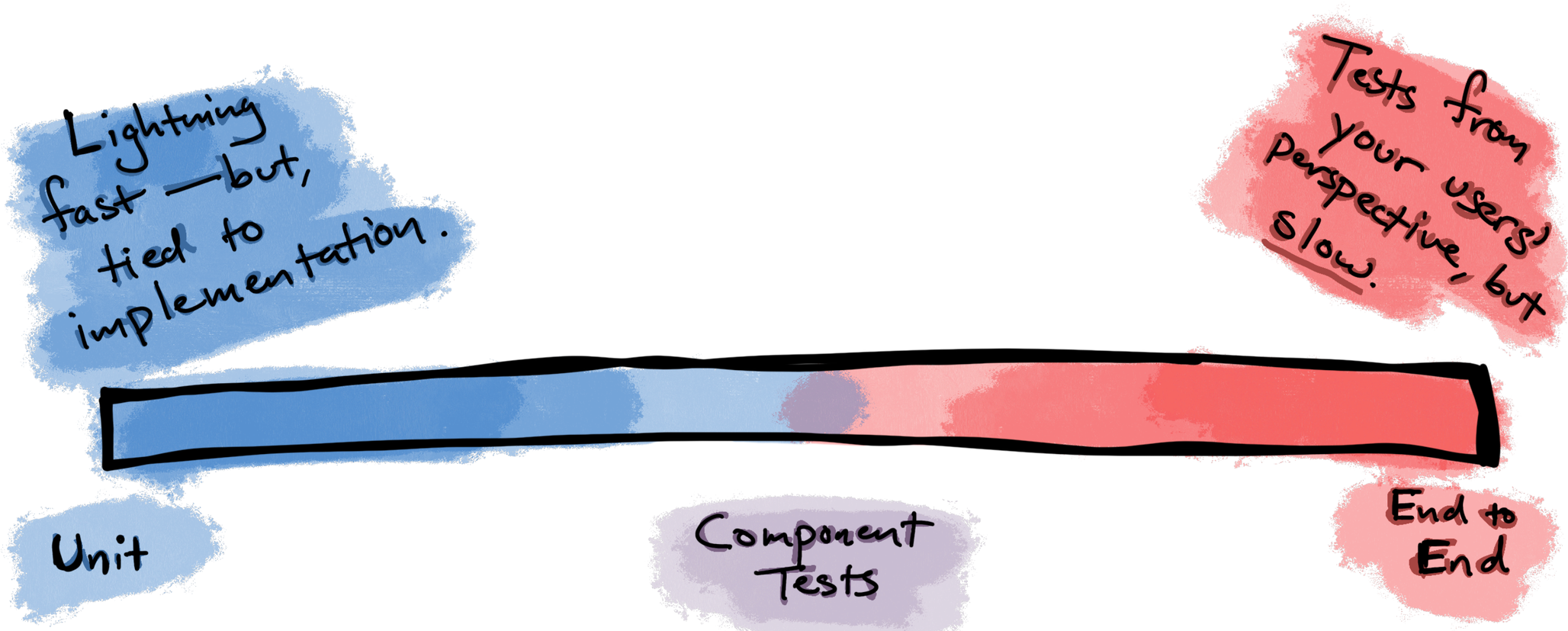
Disclaimer

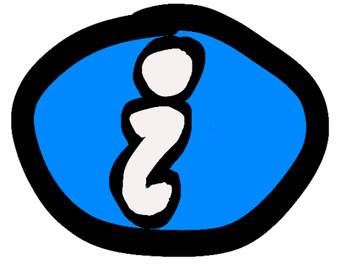


The definitions that follow do not represent the opinions of my employer—or really anyone else, for that matter.

The Spectrum of Testing*

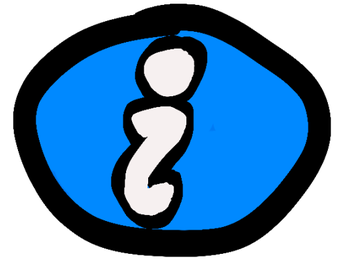
*As made up by yours truly.





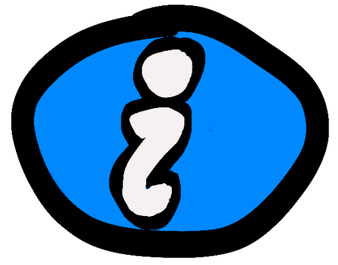
Unit Tests

The smallest possible test you can get away with. It hands some arguments to a function and checks to see if we we got back what we expected. (Pun intended.)



End-to-End Tests

Test the entire application in the Real World™. This is the—mostly—unattainable platonic ideal.

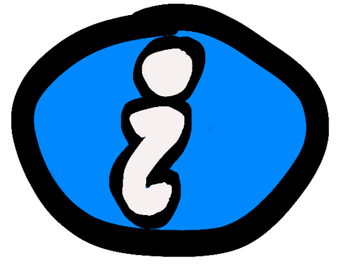


Integration Tests

Ideally, one or more units. But, that’s bit pedantic for our use.

We’re going to save this term for for the type of tests where we spin up a browser and poke at our entire application from the user’s perspective.

We’re just going to pretend like these are end-to-end tests. Don’t tell anyone.

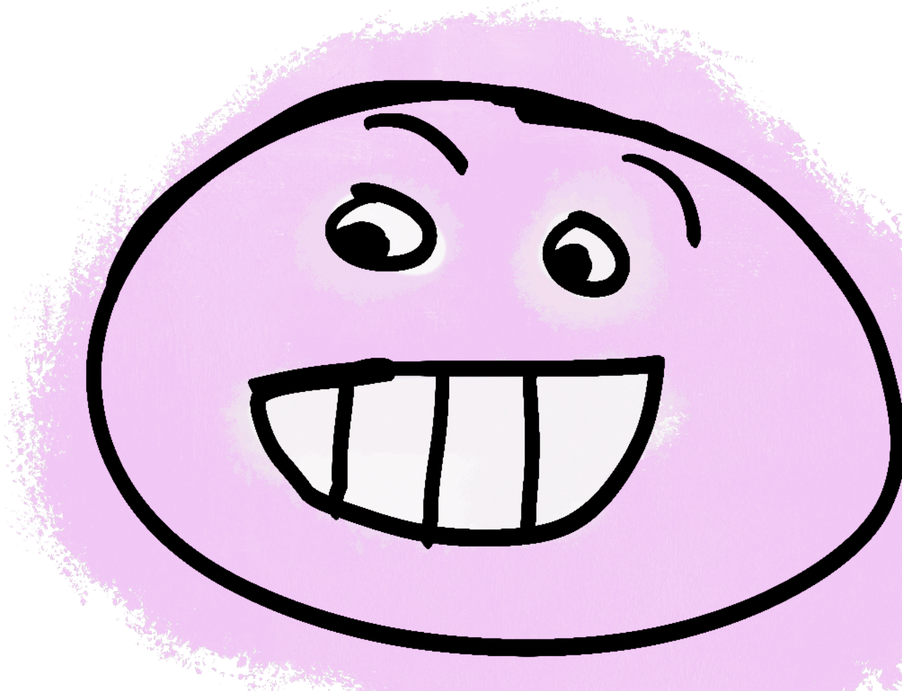


Component Tests

Render a single component from our application and poke at it. These lie somewhere in between our integration tests and unit tests.

They're *definitely* faster than spinning up a browser.

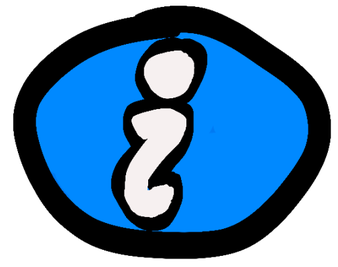
Unit Testing: Your First Layer of Defense



A Super Simple Test

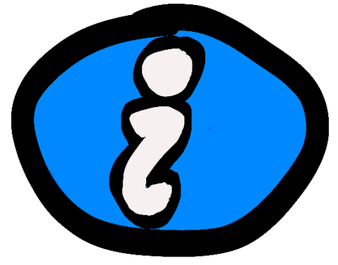
Your Platonic ideal of a unit test.

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

const add = (a: number, b: number) ⇒ a + b;

it('should correct add two numbers', () ⇒ {
  expect(add(2, 4)).toBe(6);
});
```



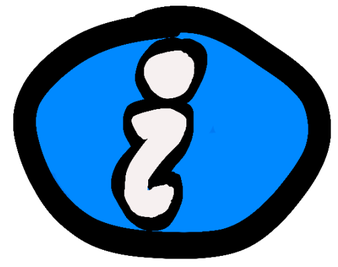
Assertion

A thing that you'd ideally like to be true and should *totally* throw and error if that's *not* the case.



Test

A function that makes your code do things. If everything goes the way you expect, live goes on. Otherwise, it throws an error for your test runner to keep track of and tell you about at the end.



Test Suite

A pretentious way to refer to a JavaScript file that runs your code and intentionally throws an error if things are now how you'd like them to be.

Running Your Tests

Exercise: Running Your Tests

`src/examples/getting-started`

(This totally *isn't* an excuse to see if your setup works.)

Head into `./src/examples/getting-started` and run the following: `npx vitest` and `npm test`.

Truth Bomb:

Tests pass because they don’t fail.

Experiment: Vitest UI and Reporters

Run the following:

- `npx vitest --run --reporter=verbose`
- `npx vitest --run --reporter=dot`
- `npx vitest --ui`

Insist on Having Expectations

This is typically something I only use while debugging.*

```
expect.hasAssertions();
expect.assertions(1);
```

*Or, setting up fake examples for a course.

Meet Me Here:

src/examples/great-expectations/objects.test.ts

Exercise: Great Expectations

`src/examples/great-expectations/exercise.test.ts`

Can you bravely make the tests pass?

Too easy? Try `bonus-exercise.test.ts`.

Experiment: Running Tests, Relatively

Inside of `src/examples/getting-started`:

1. Run `npx vitest --run` and look at which test files run.
2. Run `npx vitest words --run` and look at which test files run.
3. Run `npx vitest related ./math.ts --run` and look at which test files run.
4. Run `npx vitest related ./exponent.ts --run` and look at which test files run.
5. Assuming you don't have any unstaged or uncommitted changes, `run npx vitest --changed HEAD --run` and look at which test files—umm—_didn't_ run.
6. Make a change to `words.ts` (or any other file, really) and then run `npx vitest --changed HEAD --run` and see what tests run.

Meet Me Here:

`src/examples/asynchronicity`

Asymmetric Matching

Getting Started with Github Actions

Automate All the Things!

Experiment: Building Your Assets

- Right now `build-and-test` only tests. Maybe it should build first?
- Could you add a name to the step where we check out the repository? What would that look like?
- *Challenge Mode*: What if wanted to run `npm run build` in parallel with `npm test`?

Component Testing: The Happy Medium?

Meet Me Here:

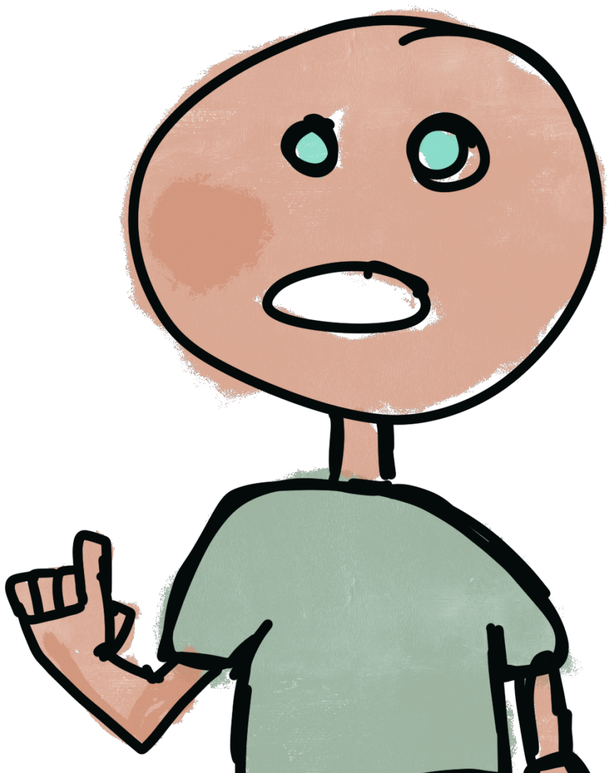
`src/examples/counter/counter.test.tsx`

Exercise: Packing List

`src/examples/packing-list`

- - Verify that there is an input field for adding a new item.
- - Verify that the "Add New Item" button is disabled when the input field is empty.
- - Verify that the "Add New Item" button is enabled when there is content in the input field.
- - Verify that the new item is added to the page when the "Add New Item" button is clicked.

Browser Testing: Worth It, If You Can Afford It



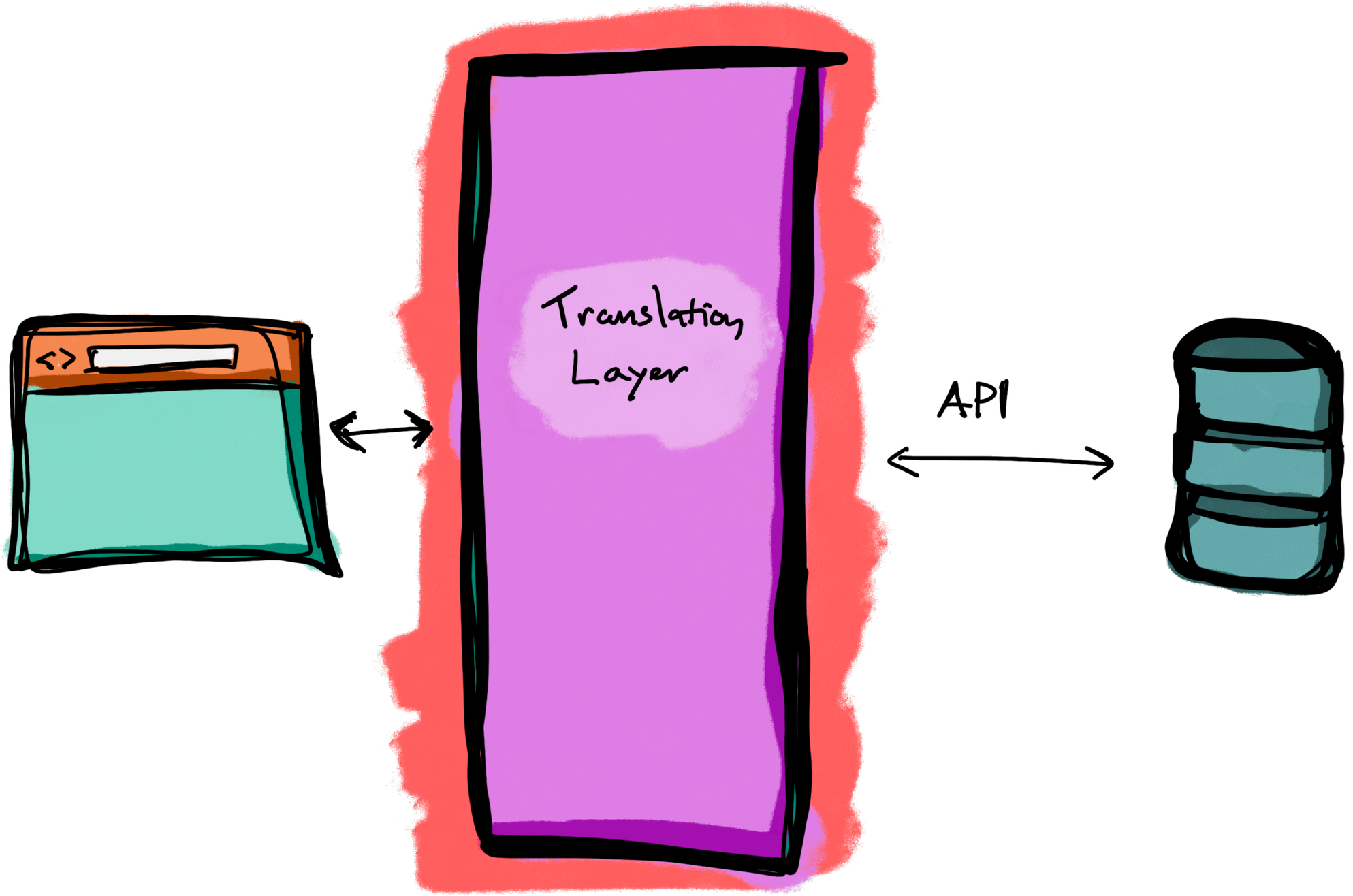
Patterns

Prefer Dependency Injection

```
import someLibrary from 'somewhere';
import someOtherLibrary from 'somewhere-else';

const SomeComponent = () ⇒ {
  const one = someLibrary();
  const two = someOtherLibrary();

  /…
};
```

```
import someLibrary from 'somewhere';
import someOtherLibrary from 'somewhere-else';

const SomeComponent = (a = someLibrary, b = someOtherLibrary) => {
  const one = a();
  const two = b();

  /…
};
```

```tsx
export const SomeComponent = (props) => {
  const [state, dispatch] = useSelector(selectState);

  return <SomeComponentView {...props} {...state} />
}

export const SomeComponentView = (props) => {
  // No logic here!

  /…
};

export default SomeComponent;
```


Translation Layer
API

Questions?