

# TDD IN FRONTEND

by Aleksandr Shinkarev

Back in the late 1990s

Test Driven Development (TDD) is a software development approach where tests are written before the actual code. Yeah, that's it...



Bottom Line

Writing your code according to TDD today is a fun way of working



Is it a magazine or something?

# Why TodDDay?

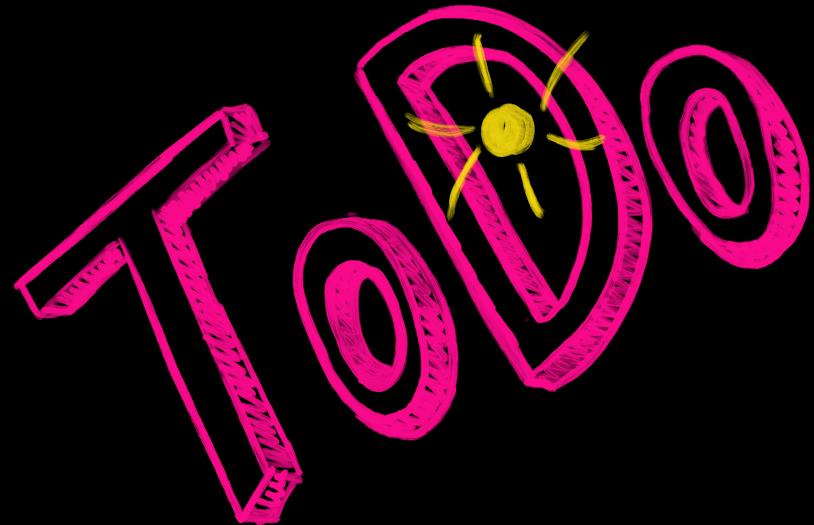
- 1 Maturity
- 2 Technologies
- 3 Automation

## Context

# Your Favorite Web App

Two User Stories

Ugly looking UI



**Not only in-memory!**

Making actual network calls  
when The Empire strikes back.

## User Story Numero Uno

# List of ToDos

**As** a User

**Want to** be able to see my ToDos

**So that** I know what is left to be done



Contract First

# Endpoint to Read

**GET** /api/to-dos-api/to-dos

Response

```
{  
  "toDos": [  
    {  
      "id": 1,  
      "name": "Test"  
    }  
  ]  
}
```

## User Story Deux

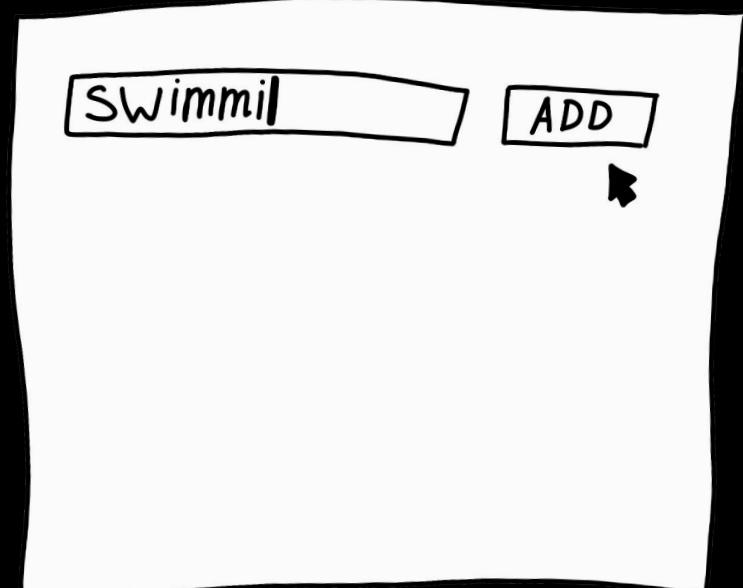
# New ToDo

**As** a User

**Want to** be able to add a new ToDo  
**So that** I can keep track of my duties

### Acceptance Criteria

1. I can add a ToDo with some name
2. When I add it, it should appear  
in the list of my ToDos



Contract First

## Endpoint to Create

**POST** /api/to-dos-api/to-dos

Request Body

```
{  
  "name": "Dishes"  
}
```

Response

```
{  
  "newToDoId": 2  
}
```

## Endpoint to Delete

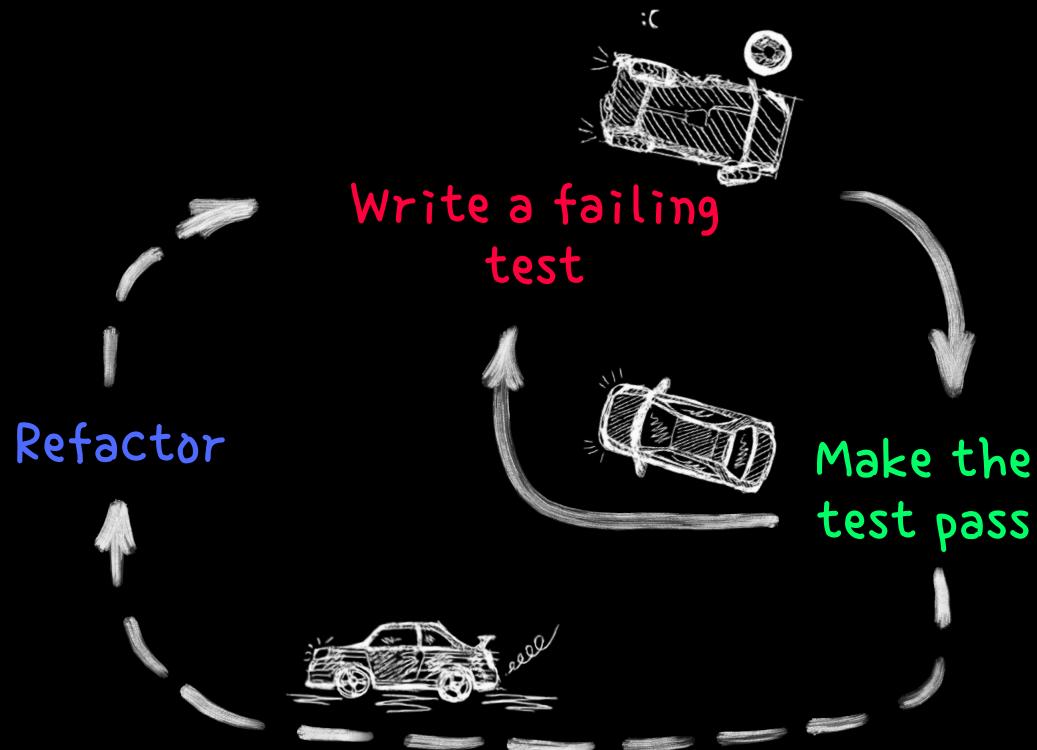
**DELETE** /api/to-dos-api/to-dos?toDoid=2

# TDD Recap

**Red** - fail

**Green** - pass

**Blue** - improve



# Behaviour-Driven Development with Whom?

Bless you!

Example of Gherkin syntax for specifications

**Given** a free online course

**When** watch it and practice

**ThenShould** not buying expensive courses



# Show time!



[https://github.com/Tou  
rmalineCore/to-dos-ui](https://github.com/Tou<br/>rmalineCore/to-dos-ui)



[https://github.com/Tou  
rmalineCore/to-dos-api](https://github.com/Tou<br/>rmalineCore/to-dos-api)

# Swagger

- OpenAPI Specification
- Playground
- Typed Backend Code → OpenAPI Specification → TypeScript Types

Swagger home page



# Cypress is WAT?

- A tool to write and run your web app tests within
- A way to make everything visual
- A sandbox for kids with a not yet ready ~~broken~~ API

Cypress landing shows it all



<https://www.cypress.io/>



# Cypress Component Tests

- You test a component in isolation
- Isolation means no website is running, no backend
- Only the component itself is rendered



Cypress docs about it



<https://docs.cypress.io/guides/component-testing/overview>

One more nice ref from there



<https://www.componentdriven.org/>

Image

[https://en.wikipedia.org/wiki/Spherical\\_cow](https://en.wikipedia.org/wiki/Spherical_cow)

What is  
state?



# State Management

When interact with UI



Update state

When change state



Update UI



# Why MobX?

1 As Simple  
as Possible

2 Clean



<https://mobx.js.org/>

# ES6 Class is MobX Friend #1

```
class TimerState {  
    private _secondsPassed = 0  
  
    constructor() {  
        makeAutoObservable(this)  
    }  
  
    get secondsPassed() {  
        return this._secondsPassed  
    }  
  
    increaseTimer() {  
        this._secondsPassed += 1  
    }  
}
```

The only bit of MobX magic

That is how we read from getters with **no** arguments

This is how we modify state using class methods  
that you **never** read through

MobX example



<https://mobx.js.org/react-integration.html>

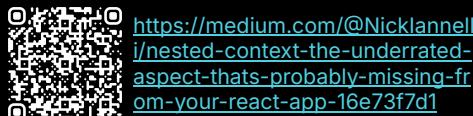
# React Context is the 2nd MobX Bestie

```
const TimerStateContext = createContext<TimerState>()

<TimerStateContext.Provider value={new TimerState()}>
  <TimerView />
</TimerStateContext.Provider>,
```

This is how we inject the state class instance

Nested React Context to get it not  
only for Dependency Injection



# React Context is the 2nd MobX Bestie

MobX magic!

```
const TimerView = observer(() => {
  const timerState = useContext(TimerStateContext)

  return (
    <span>
      Seconds passed: {timerState.secondsPassed}
    </span>
  )
})
```

This is how we access the state

# Cypress E2E Tests

Here you test your end product

Opens your product inside Cypress

You make calls to your real backend (*no or very little mocking*)

You act like a real user of your product (*as much as it makes sense*)

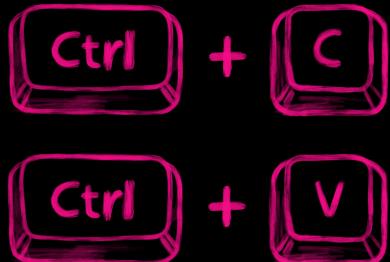
Cypress E2E getting started



<https://docs.cypress.io/guide/s/endpoint-testing/writing-your-first-end-to-end-test>

# Why so complex with all these Content, Container, State?

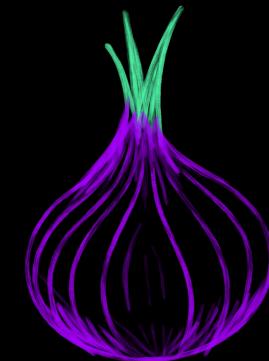
Repeatable

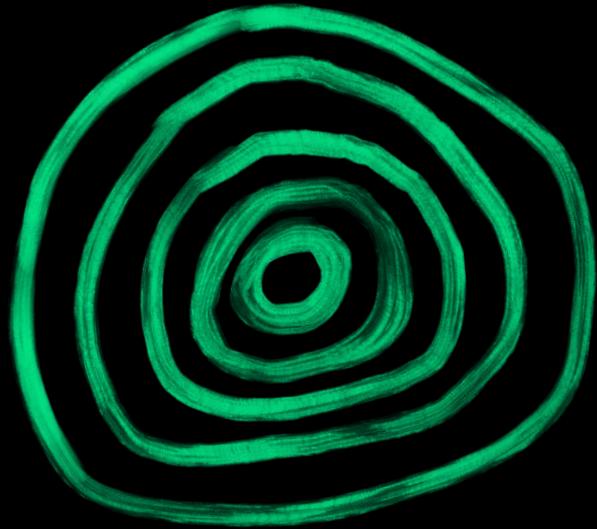


Separation of Concerns



Layers





ToDo App Example



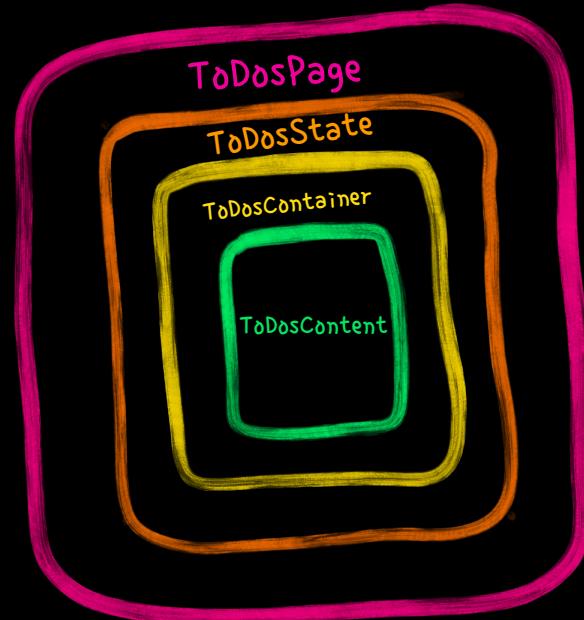
Closer to Real Life

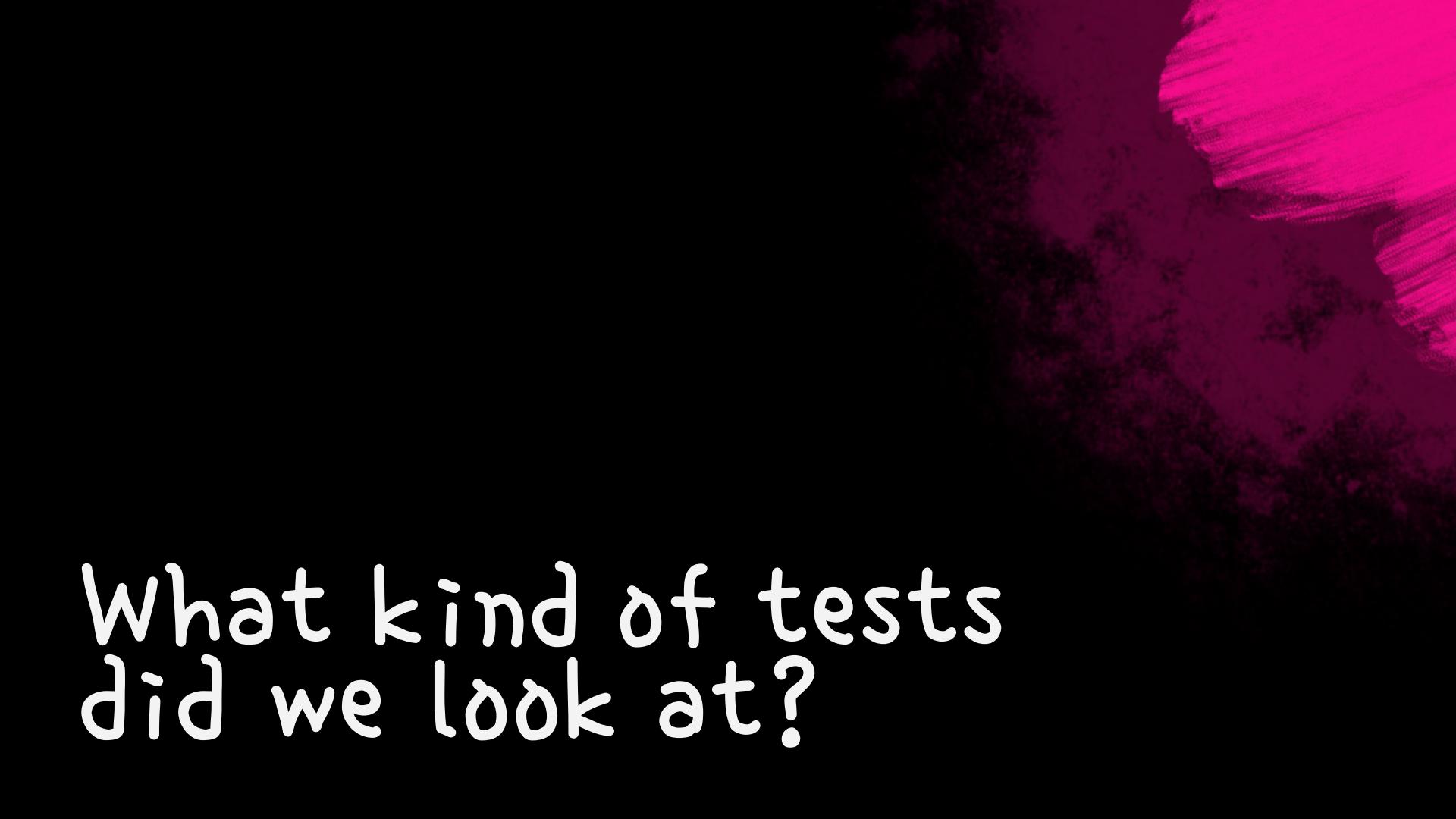
# Components Hierarchy

- Page - orchestrates
- State - data for read and write
- Container - makes network calls
- Content - interaction, **most** tested one



It is questionable, isn't it?





What kind of tests  
did we look at?

# Types of Tests

1 State

2 Component

3 E2E

4 Contract  
using linting by auto-generated  
API types

# Where to Test?

## State

Tricky logic with your state  
(modification and read)

- Toggle selection of items
- Validation of input
- Filtration

## Content

How it is rendered, if functions  
are called or not after certain event

- Is a button disabled at certain state?
- If I see a thing?
- If I click here will this callback be called/not called?

# Where to Test?

## Container

Data loading and data modification via network

- Can I receive data via network and see it?
- Can I save data via network?
- What if I receive an error making a network call?

## E2E

I can execute the user flow of the end product

- Going through the core business flow
- Reveals issues at **any** layer preventing users from getting the main business value
- The biggest ROI (Return of Investment)

# Benefits of Working Like That on Frontend

Testability is baked-in

You don't really test manually

You are independent from API

You can make quite complex pages  
based on these ideas

Code does what you expect from it

You get some dopamine

# Drawbacks

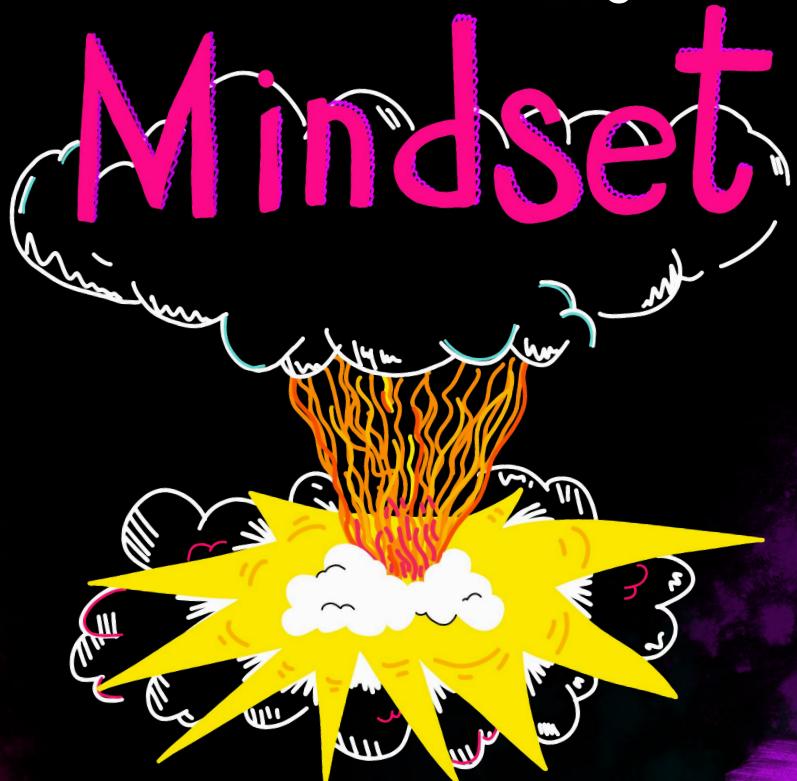
- It is difficult
- You need to carefully decide on your Testing Strategy
- Cypress is a drama queen sometimes



# These are Just Tools

- React
- Cypress
- MobX
- Swagger

But What You Really Need Is



# More References

One more guideline  
from my ex-colleague



Bran van der Meer - Test-Driven Development  
in JS with Acceptance Tests 53m  
[https://www.youtube.com/watch?v=ym62X\\_gvMXs](https://www.youtube.com/watch?v=ym62X_gvMXs)

Uncle Bob's live  
coding on the subject



Robert C. Martin - The Three Laws of TDD  
1h 06m  
<https://www.youtube.com/watch?v=Aolfc5NwRks>

More about modern  
application of TDD itself



Dave Farley - TDD - Test Driven Development  
30 videos playlist  
<https://www.youtube.com/playlist?list=PLwLLcwQInXByqD3a13UPeT4SMhc3rdZ8q>

# Aleksandr Shinkarev

13 years of experience  
CEO at Tourmaline Core

Follow us



Web site  
[tourmalinecore.com](http://tourmalinecore.com)

Design by



Anastasia  
Tupikina



Maria  
Yadryshnikova