# PAINLESS TESTS

- ME: Developer in ivelum (@ivelum in GitHub)
- GitHub, Twitter, StackOverflow: @valentjedi
- EMAIL: valentjedi@gmail.com
- https://github.com/valentjedi/hajs_2017_10_21

# SURVEY

- People here:
- Test gods:
- Tried but left:
- Don't have time:
- Wanna start:

# PAINLESS IS A LIE

BECAUSE TESTS IS A CODE

# YOU HAVE TO MAINTAIN IT.

- LOW COUPLING.
- DRY.
- READABILITY.
- . . . YOUR BUZZ WORDS HERE

# TOP 10 MISTAKES BEGINNERS DO

1. WE TEST CODE INSTEAD OF BEHAVIOUR.
2. HIGH COUPLING WITH PRODUCTION CODE.
3. UNIT TEST IS ABOUT ONE FUNCTION/CLASS.
4. NEVER DELETE TESTS.
5. DOESN'T KNOW YOUR TESTING LIBRARY. . . . . .
6. THE END

# DOING IT BAD

```
import { talks } from './bad';

test('has 0 talks if no talks was added', () => {
  expect(talks).toEqual([]);
});
```

# FAIL

```
$ npm test
[X] has 0 talks if no talks was added

    expect(received).toEqual(expected)
    Expected value to equal:
      []
    Received:
      undefined
```

# FIX

```
const talks = [];
export { talks };
```

```
 PASS  bad/bad.test.js
  v has 0 talks if no talks was added (2ms)
```

# MORE TESTS

```
test('has 1 talk if only one was added', () => {
  talks.push({name: 'Tests Suck'});
  expect(talks.length).toBe(1);
  expect(talks[0]).toEqual({name: 'Tests Suck'});
});
```

```
 PASS  bad/bad.test.js
  v has 0 talks if no talks was added (4ms)
  v has 1 talk if only one was added
```

# MORE TESTS

```
test('has 4 talks if 4 was added', () => {
  talks.push({name: 'Tests Suck even more'});
  talks.push({name: 'Tests Suck even more'});
  talks.push({name: 'Tests Suck even more'});
  talks.push({name: 'Tests Suck even more'});
  expect(talks.length).toBe(4);
});
```

```
[X] has 4 talks if 4 was added
    expect(received).toBe(expected)
    Expected value to be (using ===):
      4
    Received:
      5
```

# beforeEach

```
beforeEach(() => {
  talks.length = 0;
});
```

# NEW TEST

```
test('sorted by talk time automatically', () => {
  talks.push({name: 'Tests Suck', time: '14:00'});
  talks.push({name: 'React Sucks', time: '11:00'});
  talks.push({name: 'Vue Sucks', time: '16:00'});
  talks.push({name: 'JS Sucks', time: '22:00'});
  expect(talks.all()).toEqual([
    {name: 'React Sucks', time: '11:00'},
    {name: 'Tests Suck', time: '14:00'},
    {name: 'Vue Sucks', time: '16:00'},
    {name: 'JS Sucks', time: '22:00'},
  ]);
});
```

```
[X] sorted by talk time automatically
    TypeError: _bad.talks.all is not a function
```

# ALL INTRODUCED

```
talks.all = function() {
  const sorted = this.slice();
  sorted.sort((a, b) => {
    if (a.time < b.time) { return -1; }
    else { return 1; }
  });
  return sorted;
}
```

# FIRST ONE FAILED?

```
x has 0 talks if no talks was added (6ms)
  v has 1 talk if only one was added
  v has 4 talks if 4 was added (1ms)
  v sorted by talk time automatically
  [X] has 0 talks if no talks was added
    expect(received).toEqual(expected)
    Expected value to equal:
      []
    Received:
      []
    Difference:
    Compared values have no visual difference.
```

# WTF???

```
console.log bad/bad.test.js:8
    [ all: [Function] ]
```

# REFACTORING

```
const _talks = [];
const talks = {
  all: () => {
    const sorted = _talks.slice();
    sorted.sort((a, b) => {
      if (a.time < b.time) { return -1; }
      else { return 1; }
    });
    return sorted;
  },
  push: (talk) => _talks.push(talk),
  length: _talks.length,
}
```

```
Tests:       4 failed, 4 total
```

# FAIL FAIL FAIL

```
expect(talks) => expect(talks.all())
expect(talks[0]) => expect(talks.all()[0])
```

```
v has 0 talks if no talks was added (3ms)
  x has 1 talk if only one was added (2ms)
  x has 4 talks if 4 was added
  x sorted by talk time automatically (4ms)
Tests:       3 failed, 1 passed, 4 total
```

# WHY?

```
// bad.js
length: () => _talks.length,
clear: () => _talks.length = 0,

// bad.test.js
talks.length => talks.lenght()
beforeEach(() => {
  talks.clear();
});

Tests:       4 passed, 4 total
```

# NEW RULE? OH NO!

```
test('talks should have unique names', () => {
  talks.push({name: 'Tests Suck', time: '14:00'});
  talks.push({name: 'React Sucks', time: '11:00'});
  talks.push({name: 'JS Sucks', time: '22:00'});
  const addDup = () => talks.push({name: 'Tests Suck', time
  expect(talks.all()).toEqual([
    {name: 'React Sucks', time: '11:00'},
    {name: 'Tests Suck', time: '14:00'},
    {name: 'JS Sucks', time: '22:00'},
  ]);
  expect(addDup).toThrowError(/dup/);
});
```

# PUSH IMPROVED

```
push: (talk) => {
  if (!_talks.filter(submitted => submitted.name === talk.n
    _talks.push(talk)
  } else {
    throw new Error('duplicate talk submitted, go find othe
  }
},
```

# OH NO! TESTS SUCK!

```
[X] has 4 talks if 4 was added
    duplicate talk submitted, go find other
```

```
talks.push({name: 'Tests Suck even more'});
talks.push({name: 'Tests Suck even more1'});
talks.push({name: 'Tests Suck even more2'});
talks.push({name: 'Tests Suck even more3'});
```

# SCREW IT I GIVE UP!

```
push => submit
Tests:       4 failed, 1 passed
```

# DOING IT GOOD

```
import { talks } from './good';

it('shows nothing if no talks was added', () => {
  expect(talks).toEqual([]);
});
```

# SHOW, DON'T COUNT

```
it('shows 1 talk if 1 was added', () => {
  talks.push({name: 'Testing Rocks'});
  expect(talks).toEqual([{name: 'Testing Rocks'}]);
});
```

# SHOW, DON'T COUNT

```
it('shows 4 talks if 4 was added', () => {
  talks.push({name: 'Testing Rocks'});
  talks.push({name: 'Testing Rocks'});
  talks.push({name: 'Testing Rocks'});
  talks.push({name: 'Testing Rocks'});
  expect(talks).toEqual([
    {name: 'Testing Rocks'},
    {name: 'Testing Rocks'},
    {name: 'Testing Rocks'},
    {name: 'Testing Rocks'},
  ]);
});
```

# SORTING KINDA SAME

```
it('sorts talks by time', () => {
  talks.push({name: 'Testing Rocks', time: '11:00'});
  talks.push({name: 'React Rocks', time: '15:00'});
  talks.push({name: 'Vue Rocks', time: '13:00'});
  talks.push({name: 'JS is AWESOME', time: '16:00'});
  expect(talks.all()).toEqual([
    {name: 'Testing Rocks', time: '11:00'},
    {name: 'Vue Rocks', time: '13:00'},
    {name: 'React Rocks', time: '15:00'},
    {name: 'JS is AWESOME', time: '16:00'},
  ]);
});
```

# DROP FIRST THREE TESTS

- They was helpful stairs to get us started.
- They don't provide any value anymore.

# HORAY!

```
Tests:       1 passed, 1 total
```

- Let's do some refactoring!

# REMEMBER THIS?

```
const talks = {
  all: () => {
    const sorted = _talks.slice();
    sorted.sort((a, b) => {
      if (a.time < b.time) { return -1; }
      else { return 1; }
    });
    return sorted;
  },
  submit: (talk) => _talks.push(talk)   ,
};
```

# EXTRACT ALL PUSHES TO TEST HELPER

```
function addTalk(talk) {
  talks.submit(talk);
}
```

# AND USE IT!

```
it('allows only unique talks', () => {
  addTalk({name: 'Testing Rocks', time: '11:00'});
  addTalk({name: 'React Rocks', time: '15:00'});
  addTalk({name: 'Vue Rocks', time: '13:00'});
  addTalk({name: 'JS is AWESOME', time: '16:00'});
  const addDup = () => addTalk({name: 'Testing Rocks'});
  expect(addDup).toThrowError(/dup/);
  expect(talks.all()).toEqual([
    {name: 'Testing Rocks', time: '11:00'},
    {name: 'Vue Rocks', time: '13:00'},
    {name: 'React Rocks', time: '15:00'},
    {name: 'JS is AWESOME', time: '16:00'},
  ]);
});
```

# THAT NEW RULE AGAIN

```
[X] allows only unique talks
    expect(function).toThrowError(regexp)
    Expected the function to throw an error matching:
      /dup/
    But it didn't throw anything.
```

# COOL, LET'S IMPLEMENT IT

```
submit: (talk) => {
  if (!_talks.filter(submitted => submitted.name === talk.n
   _talks.push(talk)
  } else {
    throw new Error("You've submitted duplicate talk. Choos
  }
},
```

# WTF? OH, SURE!

```
Tests   1 failed
[X] allows only unique talks
    You've submitted duplicate talk. Choose another theme
```

```
beforeEach(() => {
  talks.length = 0;
});
```

# LIKE A BOSS!

```
Tests:       2 passed, 2 total
```

And by the way, push $=>$ submit refactoring requires only 1 change now

# PAINLESS TESTS (FIN)

- ME: Developer in ivelum (@ivelum in GitHub)
- GitHub, Twitter, StackOverflow: @valentjedi
- EMAIL: valentjedi@gmail.com
- https://github.com/valentjedi/hajs_2017_10_21