

PAINLESS JAVASCRIPT TESTING WITH CODECEPTJS



by Michael Bodnarchuk

2019

ABOUT ME

- Michael Bodnarchuk **@davert**
- Web developer from Kyiv, Ukraine
- Lead developer of **CodeceptJS**
- Also author of Codeception, Robo and others
- Tech **Consultant**, CTO at SDCLabs

MY VISION

- Tests should be simple to write and understand
- Tests have their priority. Don't write tests for everything
- Tests should follow business values
- Testing should be joyful

WHAT'S WRONG WITH END 2 END TESTS? IN JAVASCRIPT



```
product.element.all(by.xpath(cons.xpathproductRate())).then(function (pr
var i = products.length;
(function loop() {
    product.sleep(1000);
    var product = cons.xpathproductRate(i);
    product.element(by.xpath(product)).click().then(function () {
        main.waitForElementAndClick(product, cons.linkRemoveproduct)
        main.waitForElementAndClick(product, cons.radiobtnRemoveproduct)
        main.waitForElementAndClick(product, cons.btnRemoveproduct)
        i = i - 1;
        if (i > 0) {
            loop();
        }
    });
});
});
});
});
}() );
```

SOLUTION

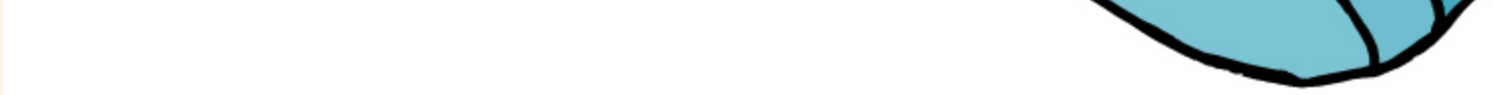
```
When I remove all products from a list
```

- Use Cucumber & BDD!

BEHIND THE SCENE

```
When('I remove all products from a list', function() {  
  
    product.element.all(by.xpath(cons.xpathproductRate())).then(function () {  
        var i = products.length;  
        (function loop() {  
            product.sleep(1000);  
            var product = cons.xpathproductRate(i);  
            product.element(by.xpath(product)).click().then(function () {  
                main.waitForElementAndClick(product, cons.linkRemoveproduct);  
                main.waitForElementAndClick(product, cons.radiobtnRemoveproduct);  
                main.waitForElementAndClick(product, cons.btnRemoveproduct);  
                i = i - 1;  
                if (i > 0) {  
                    loop();  
                }  
            });  
        });  
    });  
});
```





OVERENGINEERING

- Focused on browser control
- Focused on tools: Protractor/Cypress/webdriverio
- Asynchronicity / Promises

FOCUS ON TEST SCENARIO!

I am on page `"/login"`

I fill field "Username" with "davert"

I click "Login"

I see "Welcome, daver"

USUALLY A TEST CONSISTS OF ACTIONS:

- open a page
- click
- see text / element
- fill field
- ...

KEEP CODE SIMPLE

- Make a test follow user scenario
- Use highly flexible predefined actions
- Follow line-by-line structure
- Don't care of how it will be executed

A TEST SHOULD LOOK LIKE THIS

```
Scenario('Create a new store', (I, SettingsPage) => {
  SettingsPage.open();
  const storeName = faker.lorem.slug();
  I.dontSee(storeName, '.settings'); // Assert
  I.click('Add', '.settings'); // Click 1
  I.fillField('Store Name', storeName); // Fill fi
  I.fillField('Email', faker.internet.email());
  I.fillField('Telephone', faker.phone.phoneNumberFormat());
  I.selectInDropdown('Status', 'Active'); // Use cus
  I.click('Create'); // Auto-retry flaky
  I.waitInUrl('/settings/setup/stores'); // Explici
  I.see(storeName, '.settings'); // Assert
}).tag('stores');
```

HOW IT IS EXECUTED?

- Is that WebDriver?
- Is that Cypress?
- Is that TestCafe?
- Is that Puppeteer?

ANY OF THOSE!

CODECEPTJS



codecept.io

CODECEPTJS

- **end to end** testing framework
- helpers for popular testing backend
- high-level **unified APIs** for all backends
- **~50K** weekly installations

codecept.io

SAMPLE TEST

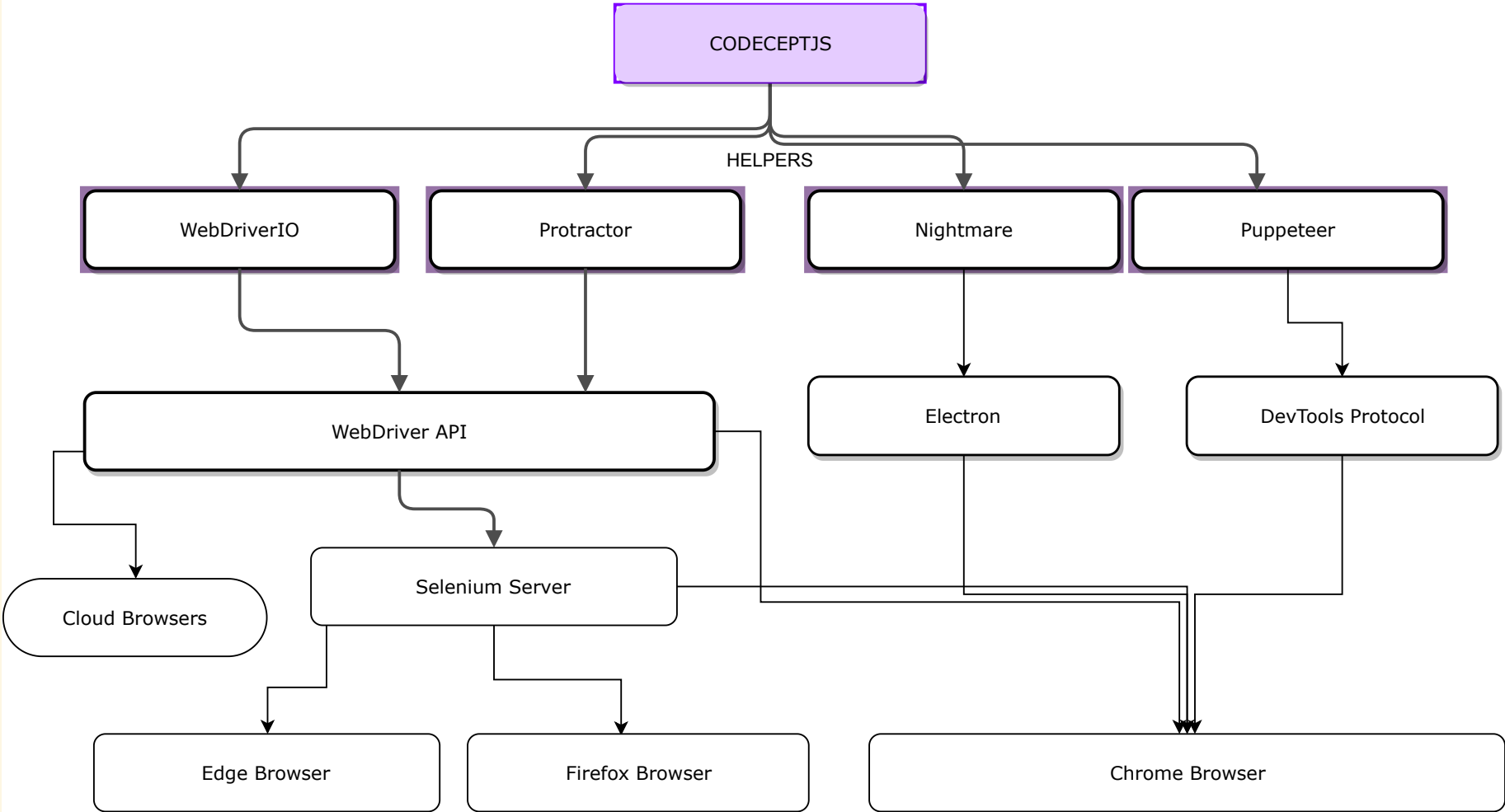
```
Scenario('todomvc', (I) => {
  I.amOnPage('http://todomvc.com/examples/react/');
  I.waitForElement('.new-todo');
  I.dontSeeElement('.todo-count');
  I.fillField('What needs to be done?', 'Write a guide');
  I.pressKey('Enter');
  I.see('Write a guide', '.todo-list');
  I.see('1 item left', '.todo-count');
  I.fillField('What needs to be done?', 'Write a test');
  I.pressKey('Enter');
  I.see('Write a test', '.todo-list');
  I.see('2 items left', '.todo-count');
  I.fillField('What needs to be done?', 'Write a code');
  I.pressKey('Enter');
  I.see('Write a code', '.todo-list');
  I.see('3 items left', '.todo-count');
});
```

```
const checkinDate = moment().add(3, 'months');

Feature('Todo');

Scenario('todomvc', (I) => {
  I.amOnPage('http://todomvc.com/examples/react/');
  I.waitForElement('.new-todo');
  I.dontSeeElement('.todo-count');
  I.fillField('What needs to be done?', 'Write a guide');
  I.pressKey('Enter');
  I.see('Write a guide', '.todo-list');
  I.see('1 item left', '.todo-count');
  I.fillField('What needs to be done?', 'Write a test');
  I.pressKey('Enter');
  I.see('Write a test', '.todo-list');
  I.see('2 items left', '.todo-count');
  I.fillField('What needs to be done?', 'Write a code');
  I.pressKey('Enter');
  I.see('Write a code', '.todo-list');
  I.see('3 items left', '.todo-count');
});
```

ARCHITECTURE



META-FRAMEWORK

- Tests can be run by Selenium, Puppeteer, TestCafe, Appium, Protractor,...
- Tests use the same interface
- If a new tool emerges it's easy to add it
- If an edgecase is hit, it's easy to migrate tests

PROBLEMS TO SOLVE

- Flaky Actions
- Flaky Locators
- Complex Locators
- Debugging
- Reporting
- Parallel Execution
- Data Management

SOLUTION ➔ FLAKY ACTIONS

- Use `autoRetry` plugin
- Explicitly retry flaky steps

```
I.retry(2).click('Link');
```

- Use `wait*` commands

```
I.waitForElement('.modal');
```

FLAKY LOCATORS

- Focus on semantic locators
 - placeholders
 - links
 - buttons
 - form element names
- Use CSS/XPath for stable parts
- Locator builder

SIMPLIFY COMPLEX LOCATORS

```
locate('//table')  
  .find('a')  
  .withText('Edit')  
  .as('edit button')
```

```
// transformed to XPath
```

READABILITY

- Focus on user actions
- Hide implementation
- Use semantic elements
- Scenarios are easy to follow

READABILITY EXAMPLE

```
I.amOnPage('/');  
I.click('Sign in', 'header');  
I.see('Sign in', 'h1');  
I.fillField('Username or email address', 'something@totest.com');  
I.fillField('Password', '123456');  
I.click('Sign in');  
I.see('Welcome!');
```

READABILITY & REUSABILITY

```
Scenario('publish an article', async (I, loginPage) => {
  const user = await I.have('user', { name: 'davert' });
  loginPage.login(davert);
  I.see('User logged in', loginPage.messageBox);
  I.click('New', '.articles');
  within('.new-article', () => {
    I.fillField('title', 'My new article');
    I.fillField('body', 'Very important message');
    I.click('Publish');
  });
  I.see('Article was published', '.message');
});
```


BDD: +1 ABSTRACTION LAYER

Scenario:

Given I have product with \$600 price in my cart

And I have product with \$1000 price

When I go to checkout process

Then I should see that total number of products is 2

And my order amount is \$1600

CODECEPTJS HAS BUILT IN CUCUMBER

- To write business scenarios
- To automate business scenarios as tests
- To combine regression tests and business scenarios

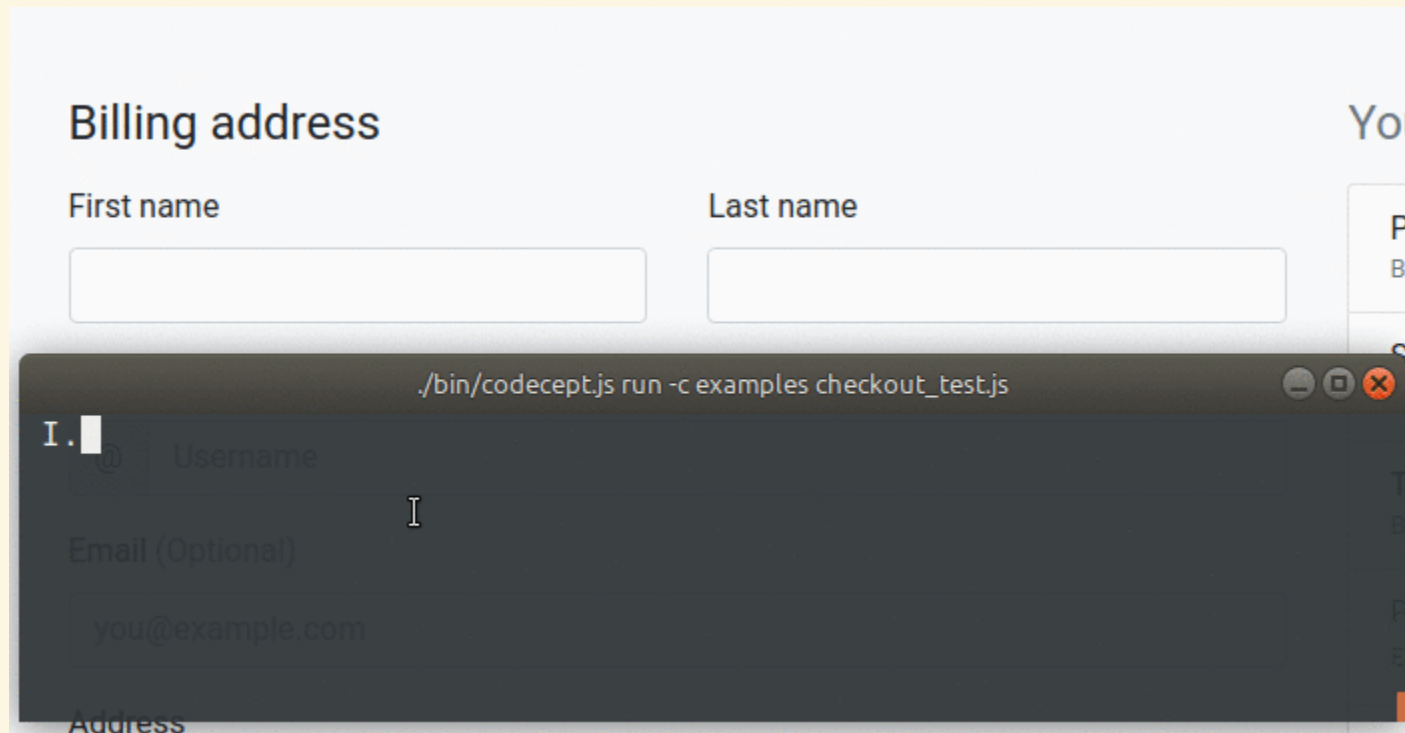
PROMISES

```
I.amOnPage('/validation_code');  
I.see('Your Validation Code:');  
let code = await I.grabValueFrom('#code')
```

- Browser commands are asynchronous
- Global promise recorder
- **await** is used when execution should be interrupted

LIVE DEVELOPMENT

```
I.amOnPage ( '/checkout' );  
pause ( );
```



DEBUGGING

- Pause execution whenever you want
- Automatically saved screenshot on fail

```
After (pause) ;
```

DATA MANAGEMENT

Generate and store all data via **REST** or **GraphQL**

```
const product = await I.have('product');  
const review = await I.have('review', { product_id: product.id });
```

PARALLEL EXECUTION

```
npx codeceptjs run-workers 3
```

REPORTING

The screenshot displays the Allure reporting interface. On the left is a dark sidebar with navigation options: Overview, Categories, Suites, Graphs, Timeline, Behaviors, and Packages. The main area is titled 'Suites' and contains a table of test results. The table has columns for 'order', 'name', 'duration', and 'status'. A filter bar shows counts for different statuses: 4 failed, 0 passed, 3 skipped, 0 pending, and 0 unknown. The test results are grouped by suite: 'GitHub' (2 failed, 3 passed) and 'Yahoo test' (1 failed). The failed test '#1 Nightmare basic test' is highlighted in yellow. To the right, a detailed view of this test is shown, including its name 'Nightmare basic test', status 'Failed', and a description: 'element (#main) still not present on page after 2 sec waiting for selector "#main" failed: timeout 2000ms exceeded'. Below this, the test's categories, severity, and duration are listed. The 'Execution' section shows a list of test steps with their durations, including 'I am on page "http://yahoo.com"', 'I fill field "p", "github nightmare"', 'I click "Search Web"', and 'I wait for element "#main", 2'. A 'Last Screenshot' is also shown, which is a thumbnail of the browser page during the test failure.

order	name	duration	status
Filter by status: 4 0 3 0 0			
▼ GitHub: 2 3			
5	register	3s 973ms	Failed
2	search @grop	5s 280ms	Passed
3	signin	5s 503ms	Passed
4	signin2	5s 692ms	Passed
1	Visit Home Page @retry	3s 737ms	Failed
▶ Testing Begins: 1			
▼ Yahoo test: 1			
1	Nightmare basic test	9s 035ms	Failed

Failed Nightmare basic test

Overview History Retries

element (#main) still not present on page after 2 sec waiting for selector "#main" failed: timeout 2000ms exceeded

Categories: Product defects
Severity: normal
Duration: 9s 035ms

Execution

▼ Test body

- ✓ I am on page "http://yahoo.com" 5s 534ms
- ✓ I fill field "p", "github nightmare" 566ms
- ✓ I click "Search Web" 506ms
- > I wait for element "#main", 2 2s 001ms

▼ Last Screenshot 373.7 KB

Allure report integrated

WRITING A TEST

▶ 0:00 / 2:37



CONCLUSIONS

- CodeceptJS is a meta framework
- CodeceptJS solves real testing problems
- CodeceptJS simplifies end to end testing

QUESTIONS

- **Michael Bodnarchuk @davert**
- CodeceptJS: codecept.io