Introduction to TestCafe

End-to-end web testing

Jennifer Proust
@proustibat

March
2019

What you'll learn

>_ Getting Started

What is TestCafe, how to install it and creating your first test.

>_ Using TestCafe

Writing a real test using a sample project: this will show you how to query elements, observe page state, interact with elements, use assertions.
Organizing tests with the page object pattern, using hooks

>_ Running Tests

Browsers support, headless mode, device emulation, concurrency

>_ Continuous Integration

Using CircleCI to run tests and see results

>_ Features

Some features you should know it exist: intercepting HTTP requests, screenshots and videos, reporters, TestCafe studio.

Getting started

What is TestCafe?

- A node.js tool to automate end-to-end web testing
- Write tests in JS or TypeScript, run them and view results
- Free and open source
- v1.0.1 since February 2019

Requirements and installation

- Be sure Node and NPM are installed (you can also use yarn)
- No webdriver needed
- Installation with just one command:

```
yarn add --dev testcafe
```

TestCafé

Test code structure

- Simplest file test looks like that:

```
fixture `Getting Started`
  .page `https://blog.xebia.fr`;

test('My first test', async t => {
  // Test code
});
```

- TestController is passed to each test function

TestCafé

Test code structure

- You run it with the command:
  `testcafe chrome test.js`

```
Running tests in:
- Chrome 72.0.3626 / Mac OS X 10.13.6

Getting Started
✓ My first test


1 passed (2s)
```

Using TestCafe

The project we gonna test

- A React app with Material-ui:
  - Home page with enter button
  - Posts page with a list of links
  - Article page with content of a post
  - Add page to post a new article

- Sources on Github

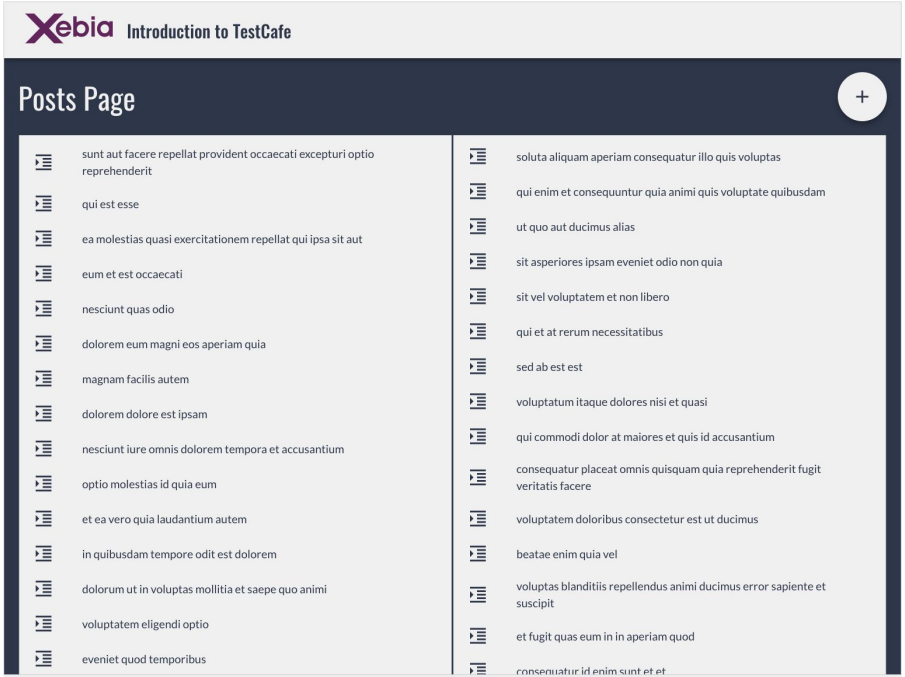
- Deployed on xke-introduction-testcafe.surge.sh

TestCafé

The home page to test

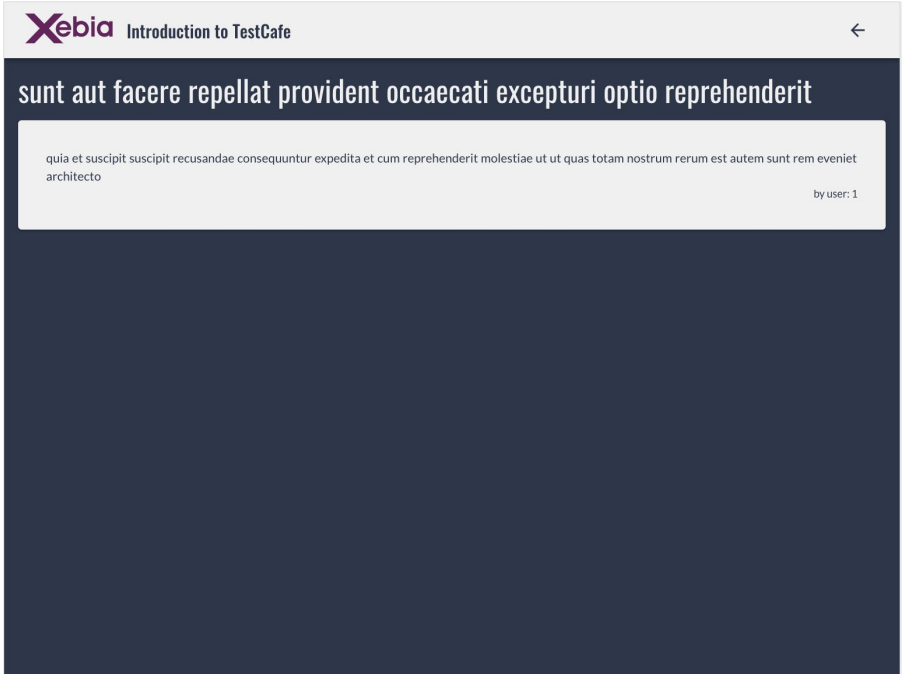
TestCafé

The posts page



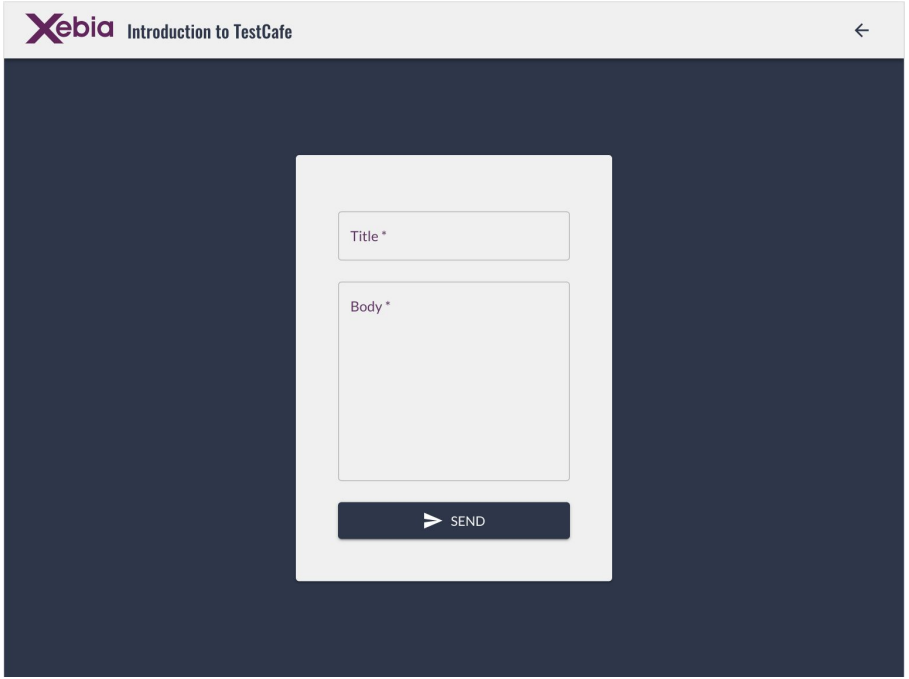
TestCafé

An article page



TestCafé

The form page to add an article



The tests we want

- As a user, when I'm on the home page, I can enter the app and see articles links. If I click on a link, I can see the article page.
- As a user, from the home, I can access to the add page with a form in order to post a new article. I can post my article.
- Enable live mode to watch for changes:
  `testcafe chrome e2e/**/* -L`

TestCafé

Let's writing our first scenario

```
fixture `Navigation`.page `http://localhost:3000`;

test('Access to an article from the home page', async t => {
 // Test code
});
```

* For now we need to run our project locally


```
Live mode is enabled.
TestCafe now watches source files and reruns
the tests once the changes are saved.

You can use the following keys in the terminal:
'Ctrl+S' - stops the test run;
'Ctrl+R' - restarts the test run;
'Ctrl+W' - enables/disables watching files;
'Ctrl+C' - quits live mode and closes the browsers.

Watching the following files:

/Users/proustibat/workspace/xebia/xke-introduction-testcafe/e2e/index.js
 Running tests in:
 - Chrome 72.0.3626 / Mac OS X 10.13.6

 Navigation
 ✓ Access to a specific article from the home page

 1 passed (0s)
```

TestCafé

Querying elements with Selector

- It's a function that identifies a webpage element in the test. The selector API provides methods and properties to select elements on the page and get their state.

- TestCafe uses standard CSS selectors to locate elements. It's like when you use document.querySelector in JS.

```
Selector('#search-subscriptions-input');
Selector('.anteriority-modal');
Selector('ul').find('label').parent('div.someClass');
Selector('p').parent(1).nth(3).find('.ctx').withText('yo!');
```

TestCafé

Selectors in our tests

```
import { Selector } from 'testcafe';

fixture `Navigation`.page `http://localhost:3000`;

test('Access to an article from the home page', async t => {
 const startBtn = Selector('a').nth(1);
});
```

Actions

- Test API provides a set of actions that enable you to interact with the webpage.
- Actions are implemented as methods in the test controller object and can be chained.
- Multiples actions are available: Click, Right Click, Double Click, Drag Element, Hover, Take Screenshot, Navigate, Press Key, Select Text, Type Text, Upload, Resize Window

TestCafé

Actions

```
import { Selector } from 'testcafe';

fixture `Navigation`.page `http://localhost:3000`;

test('Access to a specific article from the home page', async
t => {
 const startBtn = Selector('a').nth(1);
 await t.click(startBtn);
});
```

TestCafé

Observing page state

- TestCafe allows you to observe the page state:
  - Selector used to get direct access to DOM elements
  - ClientFunction used to obtain arbitrary data from the client side.
- The selector API provides methods and properties to select elements on the page and get theirs state.
- For example you can access the text content of a node element with `innerText`

TestCafé

Observing page state

```
import { Selector } from 'testcafe';

fixture `Navigation`.page `http://localhost:3000`;

test('Access to an article from the home page', async t => {
 const startBtn = await Selector('a').nth(1);
 await t.click(startBtn);
 const title = await Selector('h4');
 const titleText = await title.innerText;
 console.log(titleText); // Posts Page
});
```

* Note these methods and property getters are asynchronous

Assertions

- A functional test should also check the result of actions performed

- The following assertion methods are available: Deep Equal, Not Deep Equal, Ok, Not Ok, Contains, Not Contains, Type of, Not Type of, Greater than, Greater than or Equal to, Less than, Less than or Equal to, Within, Not Within, Match, Not Match

TestCafé

Assertions

```
import { Selector } from 'testcafe';

fixture `Navigation`.page `http://localhost:3000`;

test('Access to an article from the home page', async t => {
 const startBtn = await Selector('a').nth(1);
 await t.click(startBtn);
 const title = await Selector('h4');
 const titleText = await title.innerText;
 await t.expect(titleText).eql('Posts Page');
});
```

TestCafé

Our complete first test

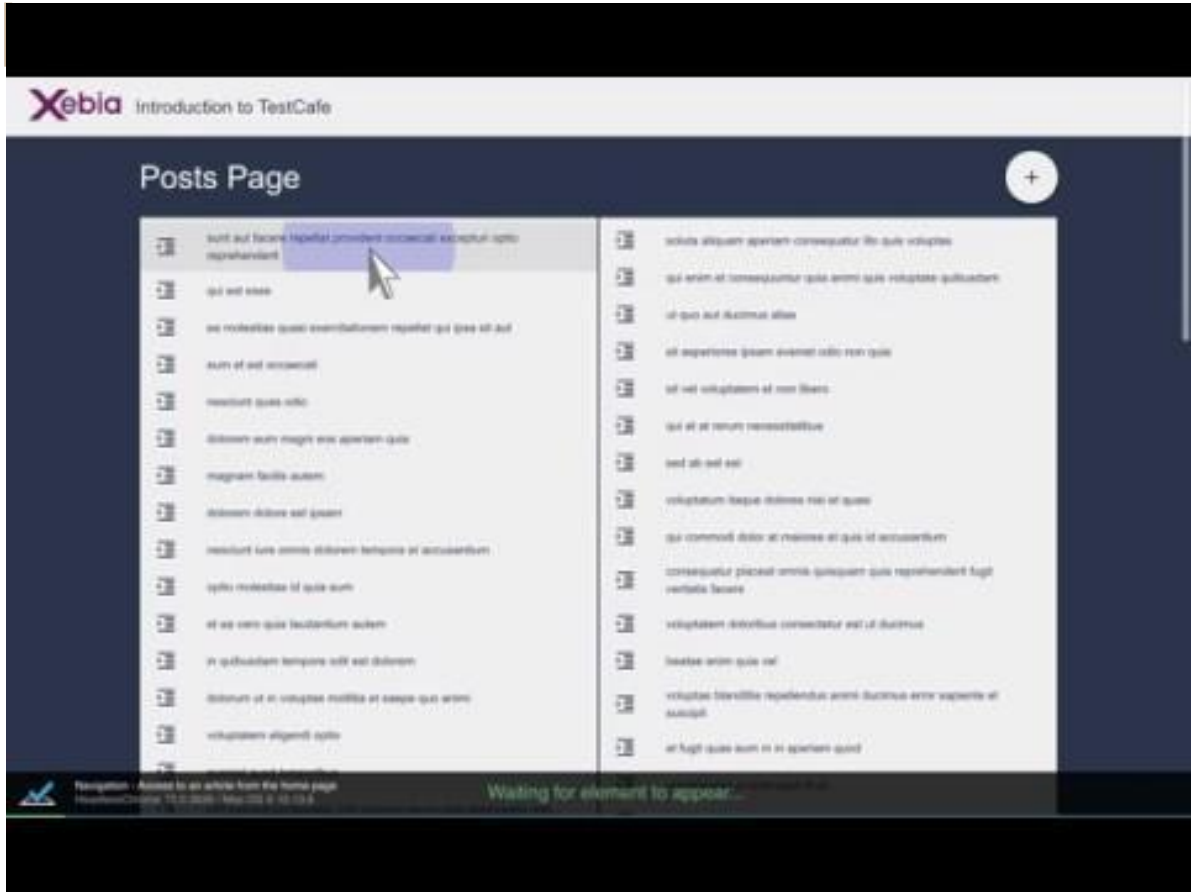
```
// WHEN (enter click on home page)
const startBtn = await Selector('a').nth(1);
await t.click(startBtn);

// THEN (posts page elements verification)
await t.expect(await Selector('h4').innerText).eql('Posts Page');
const links = await Selector('ul').child('a');
await t.expect(await links.count).eql(100);

// WHEN (click on first article link)
const firstLink = await links.nth(0).child('div');
const firstLinkText = await firstLink.innerText; // save link value
await t.click(firstLink);

// THEN (article page)
const titleArticleEl = await Selector('h4');
const titleArticleText = await titleArticleEl.innerText;
await t.expect(titleArticleText).eql(firstLinkText);
```

TestCafé



TestCafé

What about the second test?

```
test('Access to the form and posting an article, coming from home',
async t => {
 // WHEN (enter click on home page)
 const startBtn = await Selector('a').nth(1);
 await t.click(startBtn);
 // THEN (posts page elements verification)
 await t.expect(await Selector('h4').innerText).eql('Posts Page');
 const links = await Selector('ul').child('a');
 await t.expect(await links.count).eql(100);

 // TODO
 // Check if a "plus button" exists
 // Click on the button
 // Check if we've navigated to the form page
 // Enter a title and a content
 // Submit the form
 // Check the success notification
});
```

TestCafé

Let's begin with the well known part!

```
// Check if a "plus button" exists
const addBtn = await Selector('button[aria-label=Add]');
await t.expect(addBtn.exists).ok();

// Click on the button
await t.click(addBtn);

// Check if we've navigated to the form page
const formEl = await Selector('form');
await t.expect(formEl.exists).ok();
```

TestCafé

Typing text

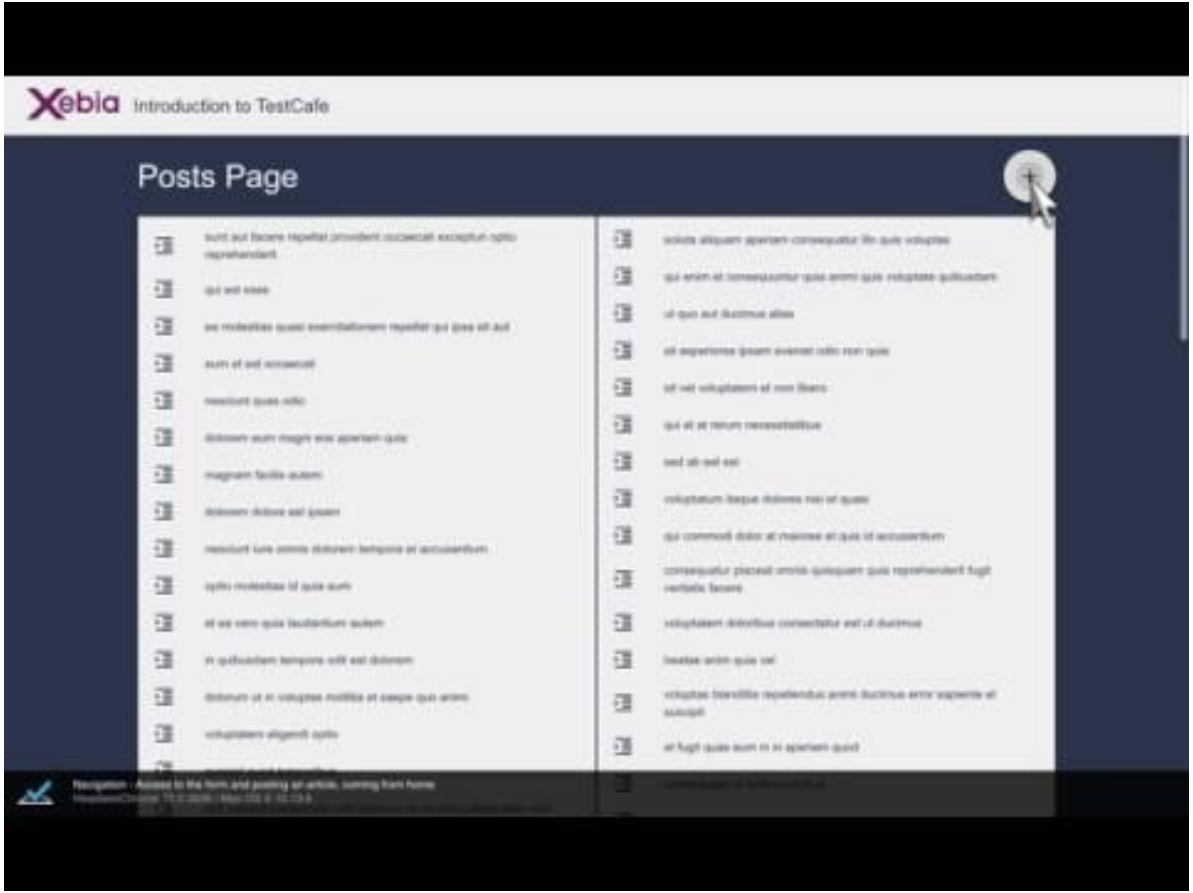
```
// Enter a title
const inputTitle = await formEl.find('#field-title');
await t.typeText(inputTitle, 'How TestCafe is awesome!');

// Enter a content
const textAreaField = await formEl.find('#field-content');
await t.typeText(textAreaField, 'Bla Bli Blou');

// Submit the form
const submitBtn = await formEl.find('button');
await t.click(submitBtn);

// Check the success notification
const toastEl = await Selector('.Toastify').child('div');
await t.expect(toastEl.exists).ok();
```

TestCafé



The page object pattern

- Page Model is a test automation pattern that allows you to create an abstraction of the tested page and use it in test code to refer to page elements.
- Keep page representation in the Page Model, while tests remain focused on the behavior.
- Improve maintainability
- Read TestCafe documentation

TestCafé

Reorganizing our code: create home model

```
import { Selector } from 'testcafe';

export default class HomePage {
 constructor () {
   this.startBtn = Selector('a').nth(1);
 }
}
```

TestCafé

Reorganizing our code: use home model

```
import { Selector } from 'testcafe';
import HomePage from './models/home';
const homePage = new HomePage();

fixture `Navigation`.page `http://localhost:3000`;

test.only('Access to an article from the home page', async t => {
  // const startBtn = await Selector('a').nth(1);
  // await t.click(startBtn);
  await t.click(homePage.startBtn);
 ...
});
```

TestCafé

Reorganizing our code: create posts model

```
import { Selector } from 'testcafe';

export default class PostsPage {
 constructor () {
   this.title = Selector('h4');
   this.links = Selector('ul').child('a');
   this.addBtn = Selector('button[aria-label=Add]');
 }
}
```

TestCafé

Reorganizing our code: use posts model

- Replace title and links selectors:

```
// THEN (posts page elements verification)
await t.expect(await Selector('h4').innerText).eql('Posts Page');
const links = await Selector('ul').child('a');
await t.expect(await links.count).eql(100);
```

By:

```
// THEN (posts page elements verification)
await t.expect(await postsPage.title.innerText).eql('Posts Page');
const links = await postsPage.links;
await t.expect(await links.count).eql(100);
```

TestCafé

Reorganizing our code: use posts model

- Replace add button selector:

```
// Check if a "plus button" exists
const addBtn = await Selector('button[aria-label=Add]');
await t.expect(addBtn.exists).ok();

// Click on the button
await t.click(addBtn);
```

By:

```
// Check if a "plus button" exists
await t.expect(await postsPage.addBtn.exists).ok();

// Click on the button
await t.click(await postsPage.addBtn);
```

TestCafé

Reorganizing our code: create article model

```
import { Selector } from 'testcafe';

export default class ArticlePage {
 constructor () {
   this.title = Selector('h4');
 }
}
```

TestCafé

Reorganizing our code: use article model

Replace:

```
// THEN (article page)
const titleArticleEl = await Selector('h4');
const titleArticleText = await titleArticleEl.innerText;
```

By:

```
// THEN (article page)
const titleArticleText = await articlePage.title.innerText;
```

TestCafé

Reorganizing our code: create add page model

```
import { Selector } from 'testcafe';

export default class AddPage {
 constructor () {
   this.form = Selector('form');
   this.inputTitle = this.form.find('#field-title');
   this.textAreaField = this.form.find('#field-content');
   this.submitBtn = this.form.find('button');
 }
}
```

TestCafé

Reorganizing our code: use add page model

- Replace:

```
// Check if we've navigated to the form page
const formEl = await Selector('form');
await t.expect(formEl.exists).ok();

// Enter a title
const inputTitle = await formEl.find('#field-title');
await t.typeText(inputTitle, 'How TestCafe is awesome!');

// Enter a content
const textAreaField = await formEl.find('#field-content');
await t.typeText(textAreaField, 'Bla Bli Blou');

// Submit the form
const submitBtn = await formEl.find('button');
await t.click(submitBtn);
```

TestCafé

Reorganizing our code: use add page model

By:

```
// Check if we've navigated to the form page
const formEl = await addPage.form;
await t.expect(formEl.exists).ok();

// Enter a title, a content and submit form
const inputTitle = await addPage.inputTitle;
const textAreaField = await addPage.textAreaField;
const submitBtn = await addPage.submitBtn;
await t
 .typeText(inputTitle, 'How TestCafe is awesome!')
 .typeText(textAreaField, 'Bla Bli Blou')
 .click(submitBtn);
```

TestCafé

Adding actions to the model pages

- TestCafe allows to use Test Controller outside of test code by importing it:

```
import { Selector, t } from 'testcafe';
```

- Add this to our add page model:

```
async submitForm (title, content) {
 await t
   .typeText(this.inputTitle, title)
   .typeText(this.textAreaField, content)
   .click(this.submitBtn);
}
```

TestCafé

Use actions from our add model:

Replace:

```
// Enter a title, a content and submit form
const inputTitle = await addPage.inputTitle;
const textAreaField = await addPage.textAreaField;
const submitBtn = await addPage.submitBtn;
await t
 .typeText(inputTitle, 'How TestCafe is awesome!')
 .typeText(textAreaField, 'Bla Bli Blou')
 .click(submitBtn);
```

By:

```
// Enter a title, a content and submit form
await addPage.submitForm(
 'How TestCafe is awesome!',
 'Bla Bli Blou'
);
```

TestCafé

Complete add page model with:

```
async isPageDisplayed () {
 await t.expect(await this.form.exists).ok();
 await t.expect(await this.inputTitle.exists).ok();
 await t.expect(await this.textAreaField.exists).ok();
 await t.expect(await this.submitBtn.exists).ok();
}
```

TestCafé

Complete posts page model with:

```
async isPageDisplayed () {
 await t.expect(await this.title.innerText).eql('Posts Page');
 await t.expect(await this.links.count).eql(100);
 await t.expect(await this.addBtn.exists).ok();
}

async clickFirstLink () {
 const link = await this.links.nth(0).child('div');
 const linkText = await link.innerText;
 await t.click(link);
 return linkText;
}
```

TestCafé

Create a toast model as follows:

```
import { Selector, t } from 'testcafe';

export default class ToastPage {
 constructor () {
   this.toastEl = Selector('.Toastify').child('div');
 }

 async isToastDisplayed () {
   await t.expect(await this.toastEl.exists).ok();
 }
}
```

Our tests

become

more simpler!

The first test

Now

```
test('Access to an article from the home page', async t =>
{
 await t.click(homePage.startBtn);
 await postsPage.isPageDisplayed();

 const text = await postsPage.clickFirstLink();
 await t
   .expect(await articlePage.title.innerText)
   .eql(text);
});
```

The first test

Before

```
test('Access to an article from the home page', async t =>
{
 // WHEN (enter click on home page)
 const startBtn = await Selector('a').nth(1);
 await t.click(startBtn);
 // THEN (posts page elements verification)
 await t.expect(await Selector('h4').innerText).eql('Posts
Page');
 const links = await Selector('ul').child('a');
 await t.expect(await links.count).eql(100);
 // WHEN (click on first article link)
 const firstLink = await links.nth(0).child('div');
 const firstLinkText = await firstLink.innerText;
 await t.click(firstLink);
 // THEN (article page)
 const titleArticleEl = await Selector('h4');
 const titleArticleText = await titleArticleEl.innerText;
 await t.expect(titleArticleText).eql(firstLinkText);
});
```

The second test

Now

```
test('Access to the form and posting an article, coming
from home', async t => {
 await t.click(homePage.startBtn);
 await postsPage.isPageDisplayed();

 await t.click(await postsPage.addBtn);
 await addPage.isPageDisplayed();

 await addPage.submitForm(
   'How TestCafe is awesome!',
   'Bla Bli Blou'
 );
 await toastPage.isToastDisplayed();
});
```

The second test

Before

```
test('Access to the form and posting an article, coming from home', async t => {

 const startBtn = await Selector('a').nth(1);
 await t.click(startBtn);

 await t.expect(await Selector('h4').innerText).eql('Posts Page');
 const links = await Selector('ul').child('a');
 await t.expect(await links.count).eql(100);

 const addBtn = await Selector('button[aria-label=Add]');
 await t.expect(addBtn.exists).ok();

 await t.click(addBtn);

 const formEl = await Selector('form');
 await t.expect(formEl.exists).ok();

 const inputTitle = await formEl.find('#field-title');
 await t.typeText(inputTitle, 'How TestCafe is awesome!');

 const textAreaField = await formEl.find('#field-content');
 await t.typeText(textAreaField, 'Bla Bli Blou');

 const submitBtn = await formEl.find('button');
 await t.click(submitBtn);

 const toastEl = await Selector('.Toastify').child('div');
 await t.expect(toastEl.exists).ok();
});
```

TestCafé

Hooks

- TestCafe allows you to specify functions that are executed before a fixture or test is started and after it is finished.
  - fixture.beforeEach(fn(t))
  - fixture.afterEach(fn(t))
  - test.before(fn(t))
  - test.after(fn(t))

- Note that test hooks override fixture hooks

TestCafé

Hooks in our tests

- We need to click on the start button in each of our tests
- So we can use

```
fixture`Navigation`
 .page`http://localhost:3000`
 .beforeEach( async t => {
   await t.click(homePage.startBtn);
   await postsPage.isPageDisplayed();
 });
```

Then remove the lines from our tests

TestCafé

Debugging

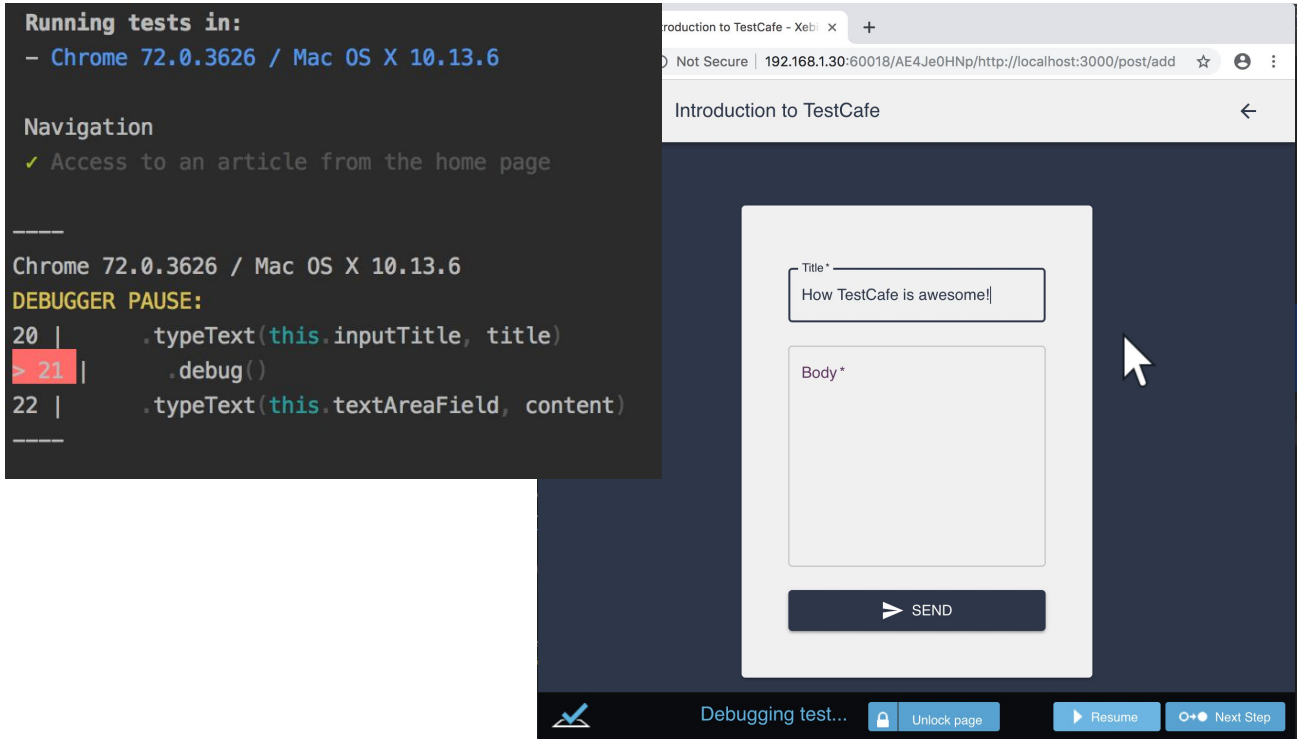
- TestCafe provides the `t.debug` method that pauses the test and allows you to debug using the browser's developer tools.

- Example:

```
async submitForm(title, content) {
 await t
   .typeText(this.inputTitle, title)
   .debug()
   .typeText(this.textAreaField, content)
   .click(this.submitBtn);
}
```

TestCafé

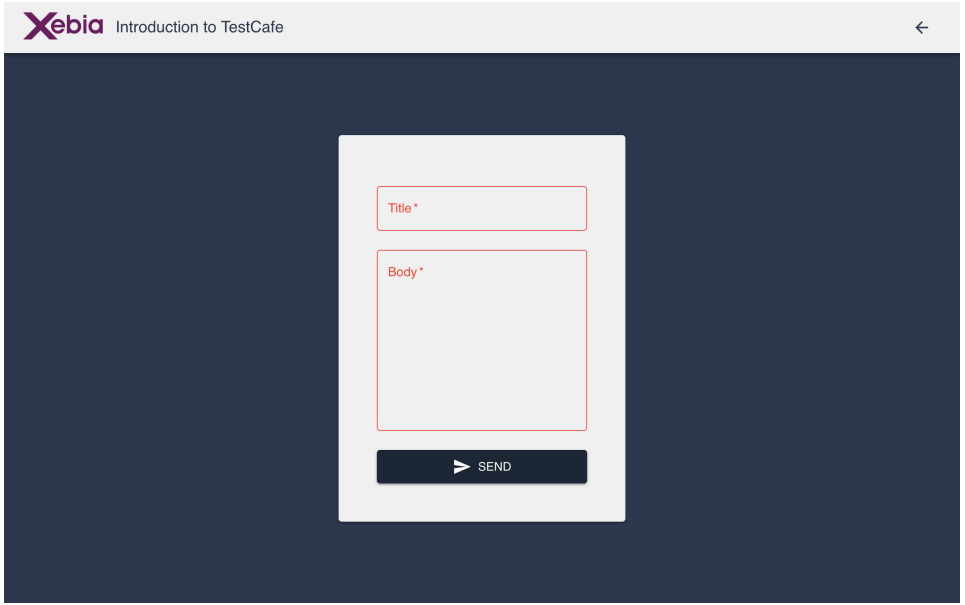
Debugging



TestCafé

Debugging with screenshots on fails

```
testcafe chrome e2e/**/* --screenshots
./screenshots --screenshots-on-fails
```



TestCafé

Debugging with videos

- `testcafe chrome e2e/**/* --video videos`
- `testcafe chrome e2e/**/* --video videos --video-options singleFile=true,failedOnly=true`

Running tests

Browsers support: installed browsers

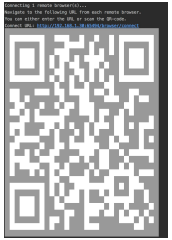
- Most of modern browsers locally installed can be detected:
  - Google Chrome: Stable, Beta, Dev and Canary
  - Internet Explorer (11+)
  - Microsoft Edge
  - Mozilla Firefox
  - Safari
  - Android browser
  - Safari mobile
- Run `testcafe --list-browsers` to list which browsers TestCafe automatically detects

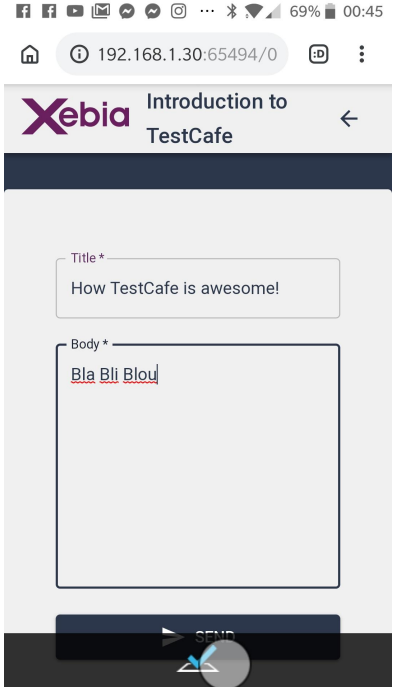
TestCafé

Browsers support: remote device

- To run tests on a remote mobile and desktop device, this device must have network access to the TestCafe server.

- `testcafe remote e2e/**/*`
- `testcafe remote e2e/**/* --qr-code`





TestCafé

Browsers support: chrome device emulation

- To do this, use the emulation browser parameter.
- Specify the target device with the device parameter.

```
testcafe
"chrome:emulation:device=iphone
6/7/8" e2e/**/* --video
artifacts/videos
```

Browsers support: cloud testing services

- TestCafe allows you to use browsers from cloud testing services.
- The following plugins for cloud services are currently provided by the TestCafe team.
  - Sauce Labs: plugin
  - BrowserStack: plugin
- You can also create your own plugin: see the doc

TestCafé

Some other CLI options

- Concurrent test execution:

```
testcafe -c 2 chrome e2e/**/*
```

- Speed:

```
testcafe chrome e2e/**/* --speed 0.5
```

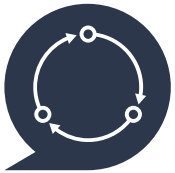
- Multiple browsers with or without headless:

```
testcafe chrome:headless,firefox e2e/**/*
```

- See the documentation for more options
- You can also use a .testcaferc.json file for the config

Reporters

- Reporters are plugins used to output test run reports in a certain format.
- TestCafe ships with the following reporters: spec (used by default), list, minimal, xUnit, JSON
- Here are some custom reporters developed by the community: NUnit, Slack, TeamCity
- Read the documentation to know more



Continuous integration

Prerequisites

- Remember we should run `yarn start` before running our tests because we target `localhost:3000` as the start webpage.
- Our goal is to test exactly the same code that will be deployed or not (depending on results).
- So we need to serve the content of build after we ran yarn build.
  - install serve package
    ```
    yarn add --dev serve
    ```
  - add a npm script to our package.json
    ```
    "serve:build": "serve -s build"
    ```

TestCafé

Modify our tests to target build sources

- In `e2e/index.js`, replace `localhost:3000` by `localhost:5000`.
- Build the app with `yarn build`
- Serve the built content in a terminal then run testcafe with headless mode in another terminal:

```
yarn serve:build
testcafe chrome:headless e2e/**/*
```

The all in one with the appCommand

- The appCommand of TestCafe executes the specified shell command before running tests.

- So we can use it to serve our built instead of running the yarn serve:build manually. Add it to a npm script as follows:

```
"e2e:ci": "testcafe chrome:headless e2e/*.js --app
'yarn serve:build'"
```

- Now we're ready for CircleCI, if you don't have any account, create it before going to the next step

TestCafé

Create config.yml in .circleci

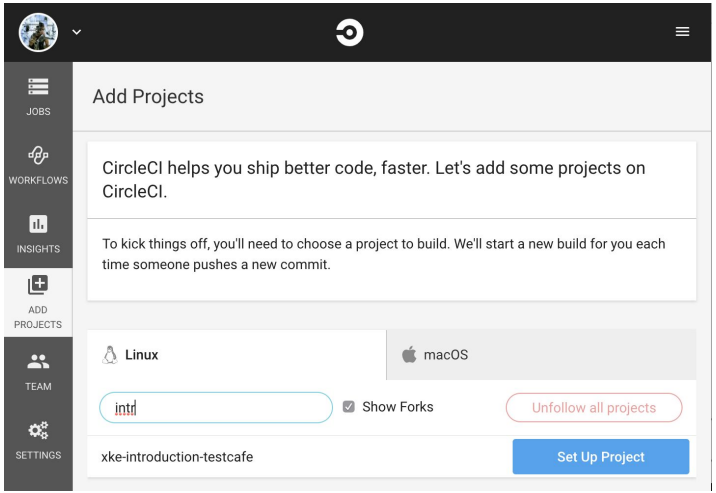
- Then commit and push

```yaml
version: 2.0
jobs:
 build:
  docker:
   - image: circleci/node:8-browsers
  working_directory: /home/circleci/project
  steps:
   - checkout
   - setup_remote_docker:
      docker_layer_caching: true
   # Download and cache dependencies
   - restore_cache:
      name: Restore Yarn Package Cache
      keys:
       - yarn-packages-{{ checksum "yarn.lock" }}
   - run:
      name: Install Dependencies
      command: yarn install --frozen-lockfile
   - save_cache:
      name: Save Yarn Package Cache
      key: yarn-packages-{{ checksum "yarn.lock" }}
      paths:
       - ~/.cache/yarn
   # Build
   - deploy:
      name: Build
      command: yarn build
    # End-to-end tests
    - run:
        name: Run e2e tests
        command: yarn e2e:ci
```

TestCafé

Add your repo to CircleCI

- On your dashboard click on “add projects” in the left bar menu
- Find your repo, then click on “Set Up Project”



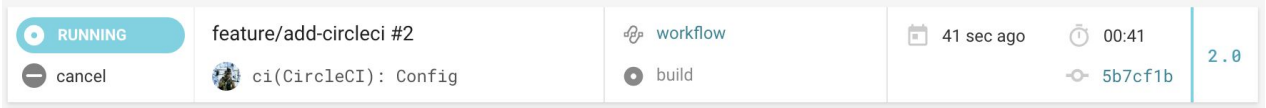
TestCafé

Start building the project on CircleCI

5. Start building! This will launch your project on CircleCI and make our webhooks listen for updates to your work.

Start building

You should see your project running:



TestCafé

Detailed workflow on CircleCI

Spin up Environment	00:05
Checkout code	00:00
Setup a remote Docker engine	00:06
Restore Yarn Package Cache	00:00
Install Dependencies	00:33
Save Yarn Package Cache	00:17
Build	00:24
Run e2e tests	00:11

TestCafé

Our E2E tests running

Run e2e tests 00:11

```
$ #!/bin/bash -eo pipefail                                    Exit code: 0
  yarn e2e:ci

yarn run v1.12.3
$ testcafe chrome:headless e2e/*.js --app 'yarn serve:build'
 Running tests in:
 - HeadlessChrome 72.0.3626 / Linux 0.0.0

 Navigation
 ✓ Access to an article from the home page
 ✓ Access to the form and posting an article, coming from home


 2 passed (7s)
Done in 11.73s.
```

TestCafé

Use store_test_results

- Add xunit reporter to the project:

```
yarn add --dev testcafe-reporter-xunit
```

- Modify the CircleCI config:

```yaml
# End-to-end tests
- run:
    name: Run e2e tests
    command: yarn e2e:ci
- store_test_results:
    path: /tmp/test-results
```

- Modify the npm script as follows:

```
"e2e:ci": "testcafe chrome:headless e2e/*.js --app 'yarn serve:build' -r xunit:/tmp/test-results/res.xml"
```

TestCafé

See the summary on your dashboard

Test Summary	Queue (00:01)	Artifacts	Config

Your job ran **2** tests in unknown with **0 failures**
Slowest test: Navigation Access to the form and posting an article, coming from home (took 3.21 seconds).

Be proud! Add badges to your readme:

```
[![CircleCI](https://circleci.com/gh/proustibat/xke-introduction-testcafe/tree/master.svg?style=svg&circle-token=49a7ca92ed8ebbd224600c4c57b5718c12057102)](https://circleci.com/gh/proustibat/xke-introduction-testcafe/tree/master)
```


```
[![Tested with TestCafe](https://img.shields.io/badge/tested%20with-TestCafe-2fa4cf.svg)](https://github.com/DevExpress/testcafe)
```

TestCafé

Now CircleCI is running for each pull request

Feature/add circleci #2

Open **proustibat** wants to merge 3 commits into `master` from `feature/add-circleci`

Conversation 0 · Commits 3 · Checks 0 · Files changed 4

proustibat commented a minute ago

No description provided.

proustibat added some commits an hour ago

- `ci(CircleCI): Config` ... ✓ 5b7cf1b
- `ci(CircleCI): Add xunit reporter` ✓ fc993a8
- `docs(Readme): Update with badges and e2e explanations` ✓ 1ed1049

Add more commits by pushing to the **`feature/add-circleci`** branch on **proustibat/xke-introduction-testcafe**.

All checks have passed Hide all checks
1 successful check

✓ **ci/circleci: build** — Your tests passed on CircleCI! Details

This branch has no conflicts with the base branch when rebasing
Rebase and merge can be performed automatically.

Rebase and merge ▾ You can also open this in GitHub Desktop or view command line instructions.

Features you should know it exists

TestCafé

Intercepting HTTP requests

Logging HTTP Requests

```
import { RequestLogger } from 'testcafe';
// ... < model imports and instantiations  here >
const logger = RequestLogger(
 { url, method: 'get' },
 { logResponseHeaders: true, logResponseBody: true }
);
// ...
test
 .requestHooks(logger)
 ('Access to the form and posting an article, coming from home', async t => {
    // ...
    await t.expect(
      logger.contains(r => r.response.statusCode === 200)
    ).ok();
});
```

TestCafé

Intercepting HTTP requests

Mocking HTTP Responses

```
import { RequestLogger, RequestMock } from 'testcafe';

const collectDataGoogleAnalyticsRegExp = new
RegExp('https://www.google-analytics.com/r/collect');
const mockedResponse = Buffer.from([0x47, 0x49, 0x46, 0x38, 0x39, 0x61, 0x01, 0x00,
0x01]);

const mock = RequestMock().onRequestTo(collectDataGoogleAnalyticsRegExp)
 .respond(mockedResponse, 202, {
   'content-length': mockedResponse.length,
   'content-type': 'image/gif'
 });

fixture`Navigation`
 .page(url)
 .requestHooks(mock)
 .beforeEach(async t => {
   await t.click(homePage.startBtn);
   await postsPage.isPageDisplayed();
 });
```

Framework-Specific Selectors

- TestCafe team and community developed libraries of dedicated selectors for the most popular frameworks. So far, the following selectors are available.
  - React
  - Angular
  - AngularJS
  - Vue
  - Aurelia

- Read the documentation

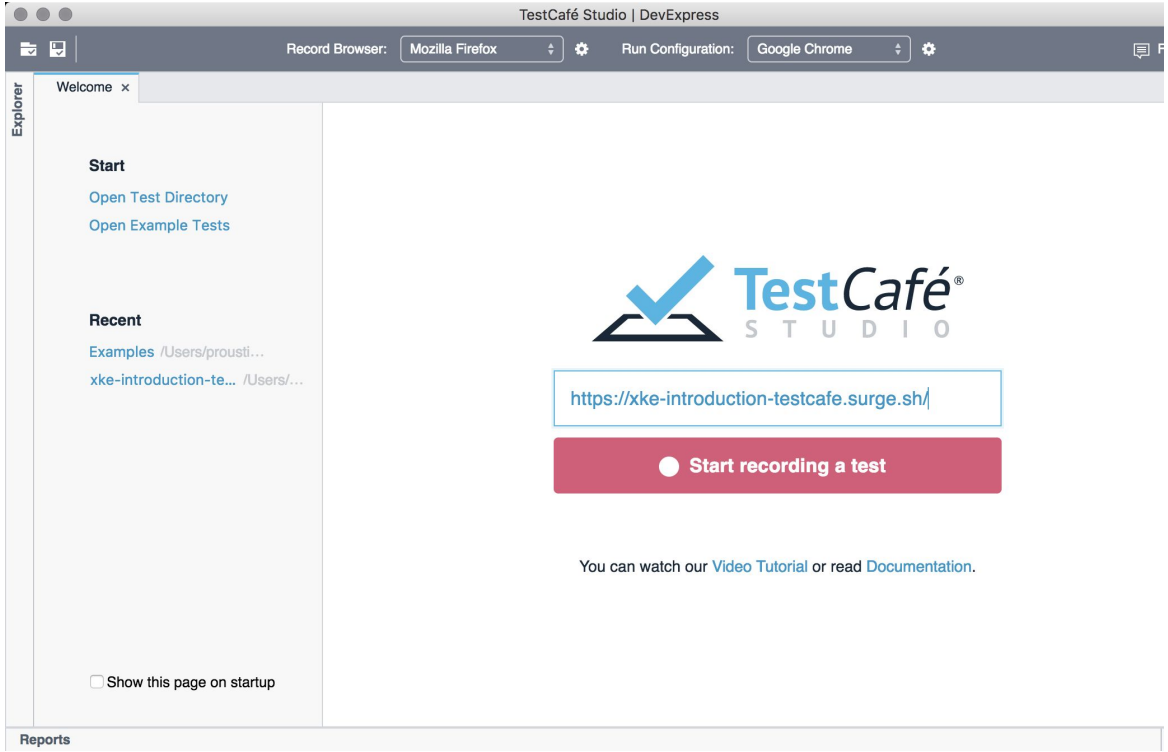
TestCafé

TestCafe Studio: cross-platform IDE

- Create, edit and maintain end-to-end tests in a visual recorder without writing code. See Record Tests.
- Generates a report with overall results and details for each test after completion.
- Code Editor with syntax highlight, code completion and parameter hints.
- Write code from scratch, convert recorded tests to JavaScript to edit them later.

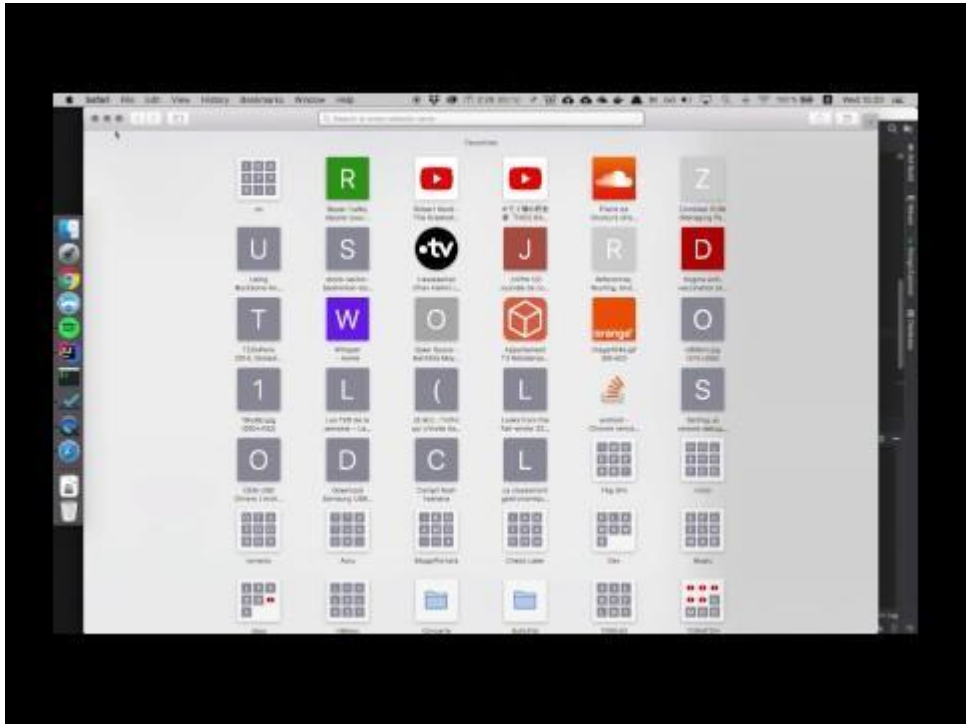
TestCafé

TestCafe Studio: cross-platform IDE



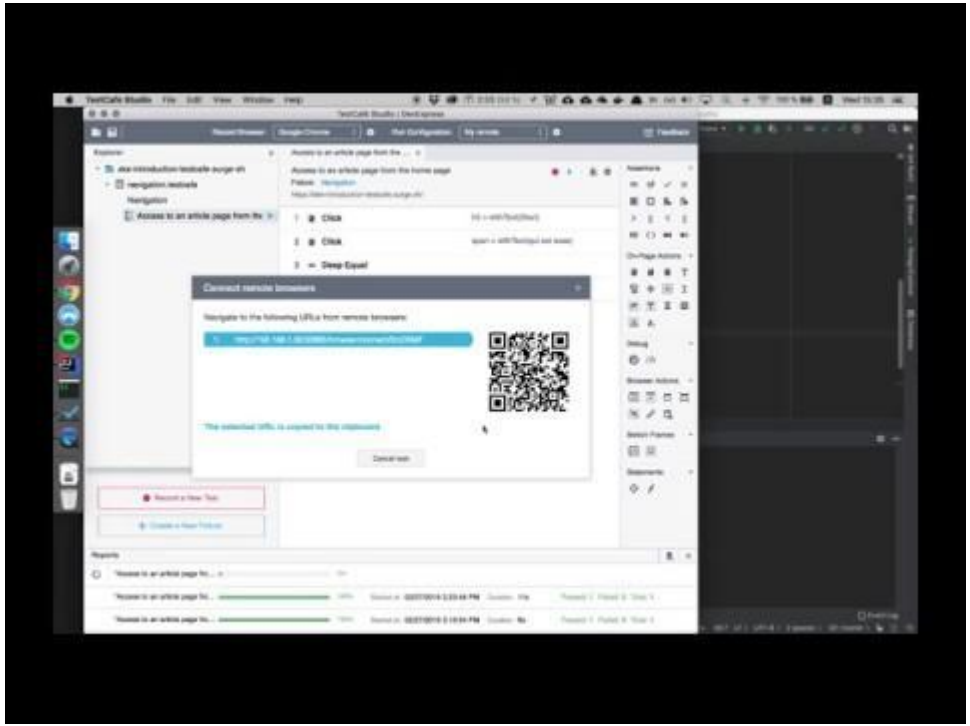
TestCafé

TestCafe Studio: our first test



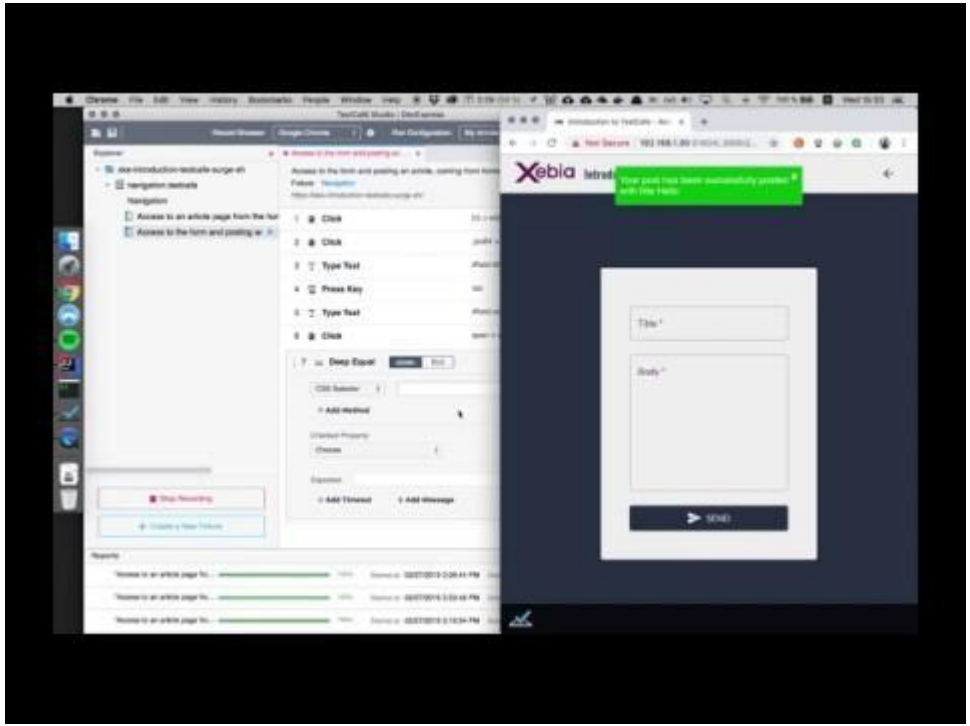
TestCafé

TestCafe Studio: running in remote



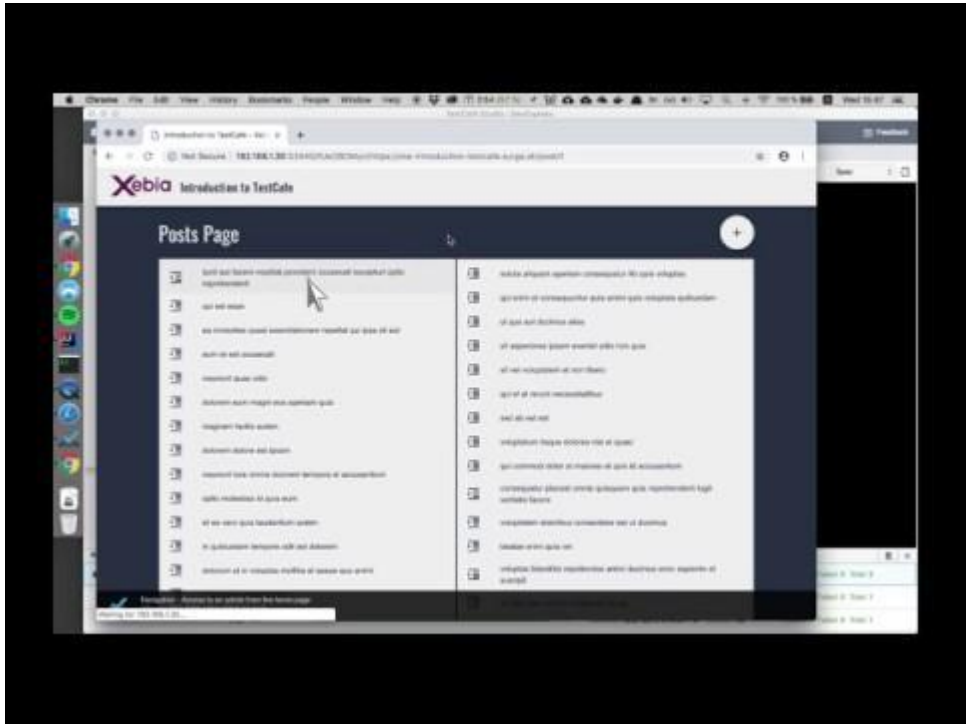
TestCafé

TestCafe Studio: our second test



TestCafé

TestCafe Studio: code editor



Thanks!

TestCafé® [Tech doc](#)

[TestCafe Studio](#)

Continue the tests by yourself:
[xke-introduction-testcafe](#)

[xke-introduction-testcafe.surge.sh](#)