Introduction to TestCafe:

Training for UI Automation

What is TestCafe?

- A node.js tool to automate end-to-end web testing
- Write tests in JS or TypeScript

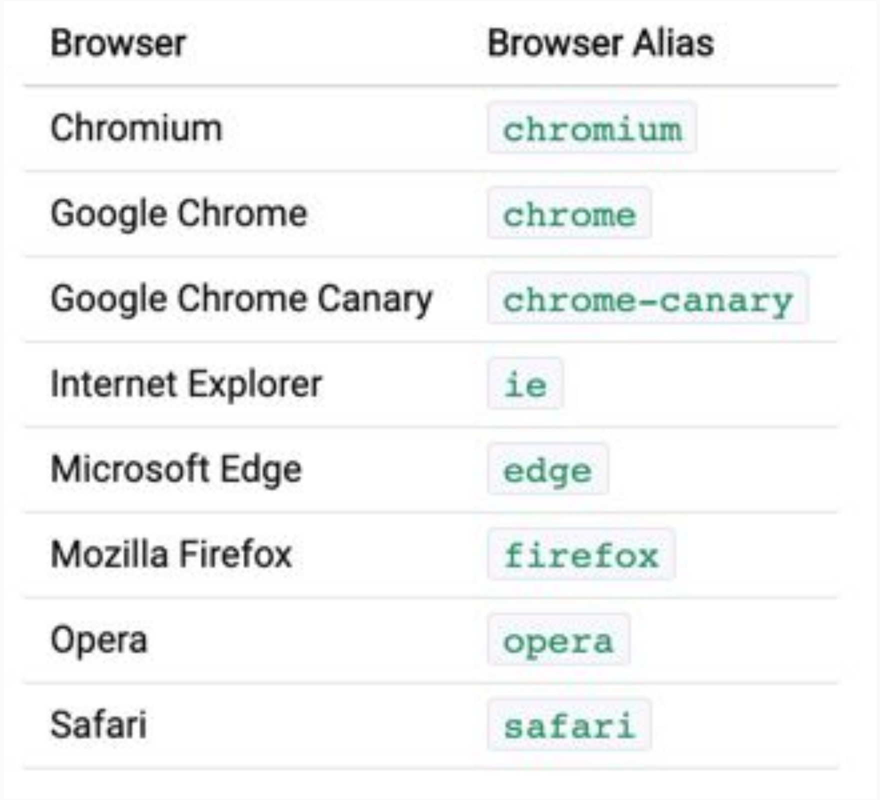
https://devexpress.github.io/testcafe/

Why TestCafe

- Runs on Windows, MacOS, and Linux
- It supports desktop, mobile, remote and cloud browsers (UI or headless).
- Easy to set up
- Quick to create
- Free and Open Source

Locally Installed Browsers

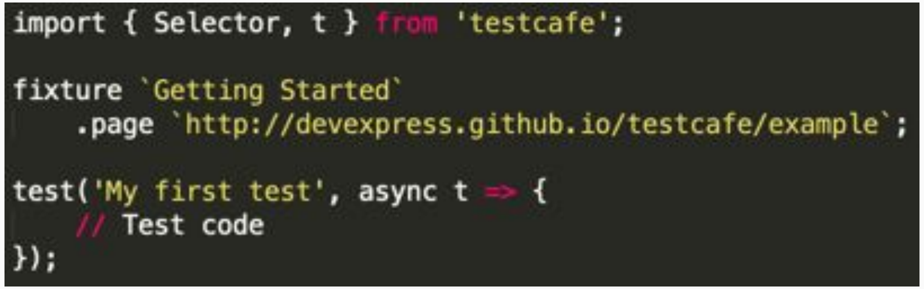
- TestCafe can automatically detect popular browsers installed on the local computer. You can use a short name - *browser alias* - to identify these browsers when launching tests. Can run headless. Example: chrome:headless



Browser	Browser Alias
Chromium	`chromium`
Google Chrome	`chrome`
Google Chrome Canary	`chrome-canary`
Internet Explorer	`ie`
Microsoft Edge	`edge`
Mozilla Firefox	`firefox`
Opera	`opera`
Safari	`safari`

What’s in a test?

- To create a test, create a new .js file
- Import TestCafe
  - import { Selector } from 'testcafe’;
- Declare a fixture
  - fixture `Getting Started`
- Create a test function
  - test('My first test', async t => {
    // Test code
    });



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

fixture `Getting Started`
    .page `http://devexpress.github.io/testcafe/example`;

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

Running test from Command Shell

- ❖ Run from command shell
  - ➢ testcafe chrome test1.js

- ❖ Variety of browsers supported
  - ➢ Google Chrome: Stable, Beta, Dev and Canary
  - ➢ Internet Explorer (11+)
  - ➢ Microsoft Edge
  - ➢ Mozilla Firefox
  - ➢ Safari
  - ➢ Google Chrome mobile
  - ➢ Safari mobile

https://devexpress.github.io/testcafe/documentation/using-testcafe/common-concepts/browsers/browser-support.html

Running test from Script

- Can run from script in package.json

  -
    ```
    "scripts": {
      "test": "testcafe chrome:headless ./feature/example.js -e"
    }
    ```

- npm run test

Running test from Shell

- Shell scripts, .sh files

  -
    ```
    testcafe --test-meta Critical=1 \
       chrome \
       ../feature/example.js \
       --skip-js-errors
    ```

Selectors?

A selector is a function that identifies a webpage element in the test.

- ❖ Initialize a selector
  - ➢ const submitButton = Selector('#submit-button');

- ❖ Use a selector
  - ➢
    ```
    await t
              .click(submitButton);
    ```

https://devexpress.github.io/testcafe/documentation/test-api/selecting-page-elements/selectors/

Actions

- Click
- Right Click
- Double Click
- Drag Element
- Hover
- Take Screenshot
- Navigate
- Press Key
- Select Text
- Type Text
- Upload
- Resize Window

Actions - Example

```
await t
	.typeText(example.youName, "John Smith")
```

https://devexpress.github.io/testcafe/documentation/test-api/actions/

Assertions

- 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

Assertions - Example

```
await t
    .expect(example.youName.value).contains('John Smith')
```

https://devexpress.github.io/testcafe/documentation/test-api/assertions/

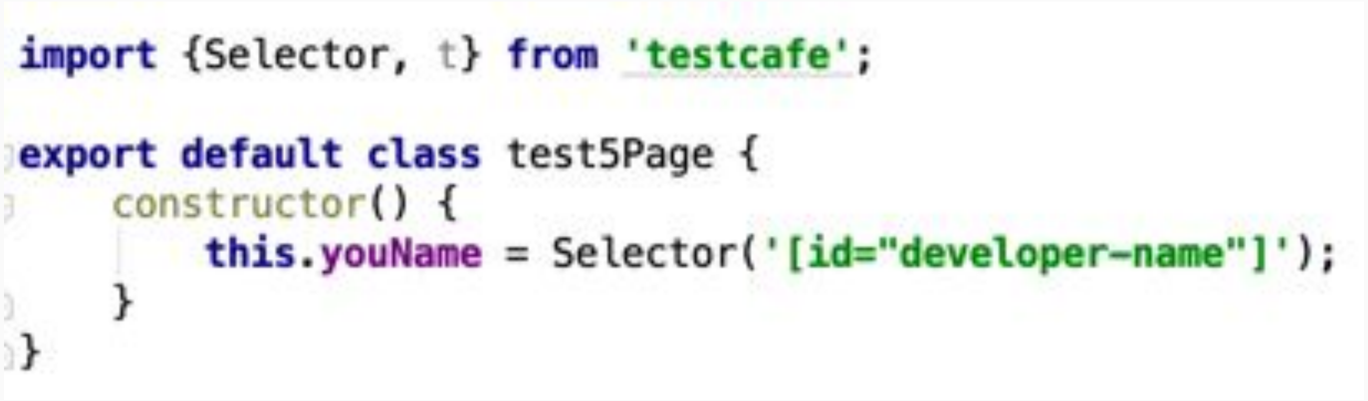
Page Model

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.

https://devexpress.github.io/testcafe/documentation/recipes/extract-reusable-test-code/use-page-model.html

Page Model

Create Page

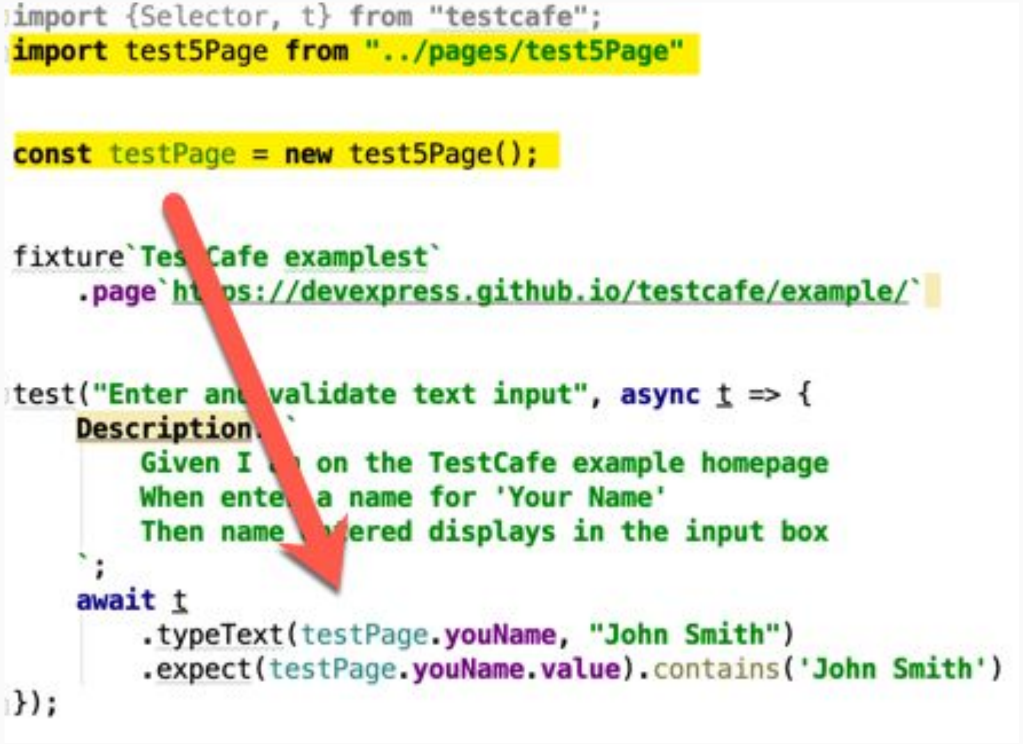


```
import {Selector, t} from 'testcafe';

export default class test5Page {
    constructor() {
        this.youName = Selector('[id="developer-name"]');
    }
}
```

Page Model

Link to Test



```
import {Selector, t} from "testcafe";
import test5Page from "../pages/test5Page"

const testPage = new test5Page();

fixture`Tes Cafe examplest`
    .page`ht ps://devexpress.github.io/testcafe/example/`

test("Enter an  validate text input", async t => {
    Description: `
        Given I    on the TestCafe example homepage
        When ente  a name for 'Your Name'
        Then name    ered displays in the input box
    `;
    await t
        .typeText(testPage.youName, "John Smith")
        .expect(testPage.youName.value).contains('John Smith')
});
```

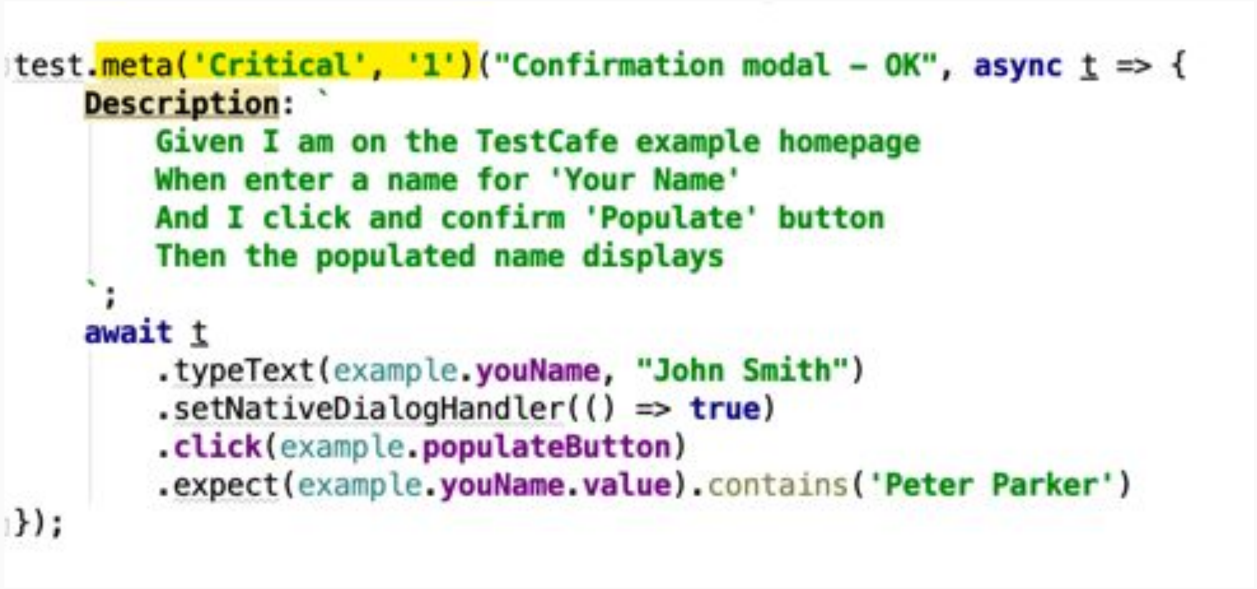
Tagging Test

Why tag test with Metadata?

- ❖ Group test
  - ➢ CI
  - ➢ QA
  - ➢ PROD
  - ➢ Critical Run
- ❖ Can Group by Fixture or Test
- ❖ Example or running all test in folder, but limit by tag
  - ➢ testcafe --test-meta Critical=1 chrome feature/

https://devexpress.github.io/testcafe/documentation/test-api/test-code-structure.html#specifying-testing-metadata

Tagging Test - Example

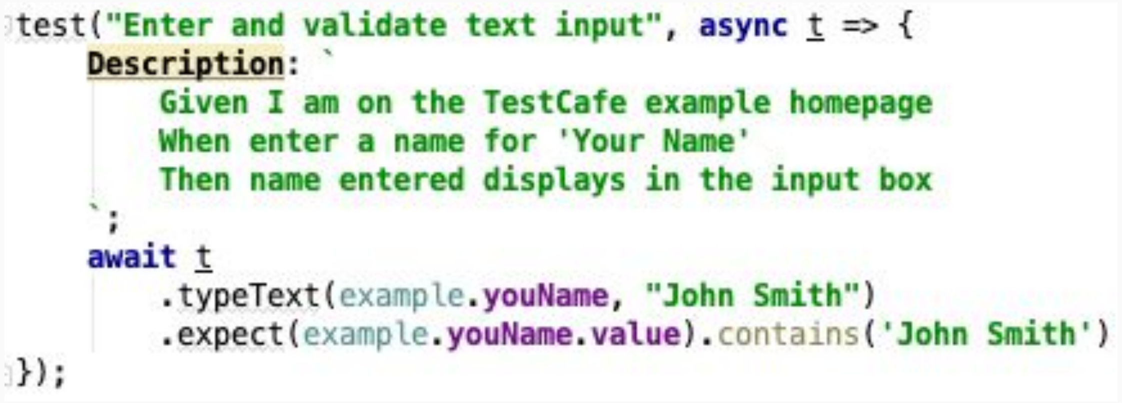


```
test.meta('Critical', '1')("Confirmation modal - OK", async t => {
    Description: `
        Given I am on the TestCafe example homepage
        When enter a name for 'Your Name'
        And I click and confirm 'Populate' button
        Then the populated name displays
    `;
    await t
        .typeText(example.youName, "John Smith")
        .setNativeDialogHandler(() => true)
        .click(example.populateButton)
        .expect(example.youName.value).contains('Peter Parker')
});
```

testcafe --test-meta Critical=1 chrome .feature/example.js -e

Gherkin as documentation

Purpose is to identify what our test does. Example of using Gherkin for documentation purpose.

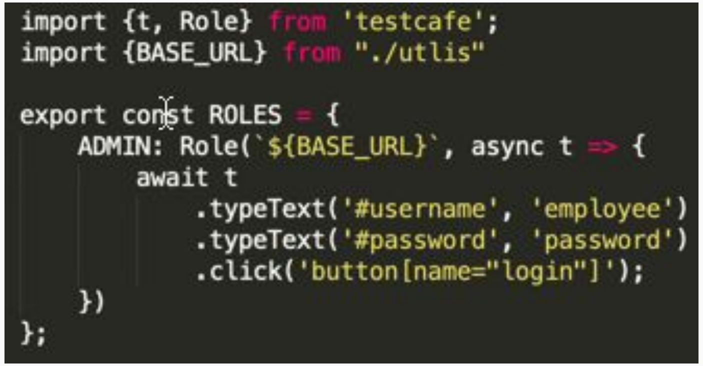


```
test("Enter and validate text input", async t => {
    Description: `
        Given I am on the TestCafe example homepage
        When enter a name for 'Your Name'
        Then name entered displays in the input box
    `;
    await t
        .typeText(example.youName, "John Smith")
        .expect(example.youName.value).contains('John Smith')
});
```

User Roles

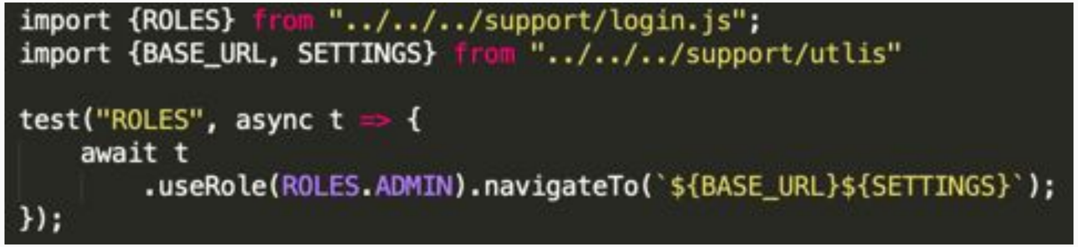
Roles are more than just another way to extract reusable test logic

Example: Login test should not be covered on every test…go straight to the source



```
import {t, Role} from 'testcafe';
import {BASE_URL} from "./utlis"

export const ROLES = {
    ADMIN: Role(`${BASE_URL}`, async t => {
        await t
            .typeText('#username', 'employee')
            .typeText('#password', 'password')
            .click('button[name="login"]');
    })
};
```



```
import {ROLES} from "../../../support/login.js";
import {BASE_URL, SETTINGS} from "../../../support/utlis"

test("ROLES", async t => {
    await t
        .useRole(ROLES.ADMIN).navigateTo(`${BASE_URL}${SETTINGS}`);
});
```

https://devexpress.github.io/testcafe/documentation/test-api/authentication/user-roles.html

Helpful things

- test.only
  - Test with this tag only run. Can be used more than once
- test.skip
  - Skips only this test. Can be used more than once.
- test.meta('key2', 'value2’)
- .debug()
- .wait(9000)