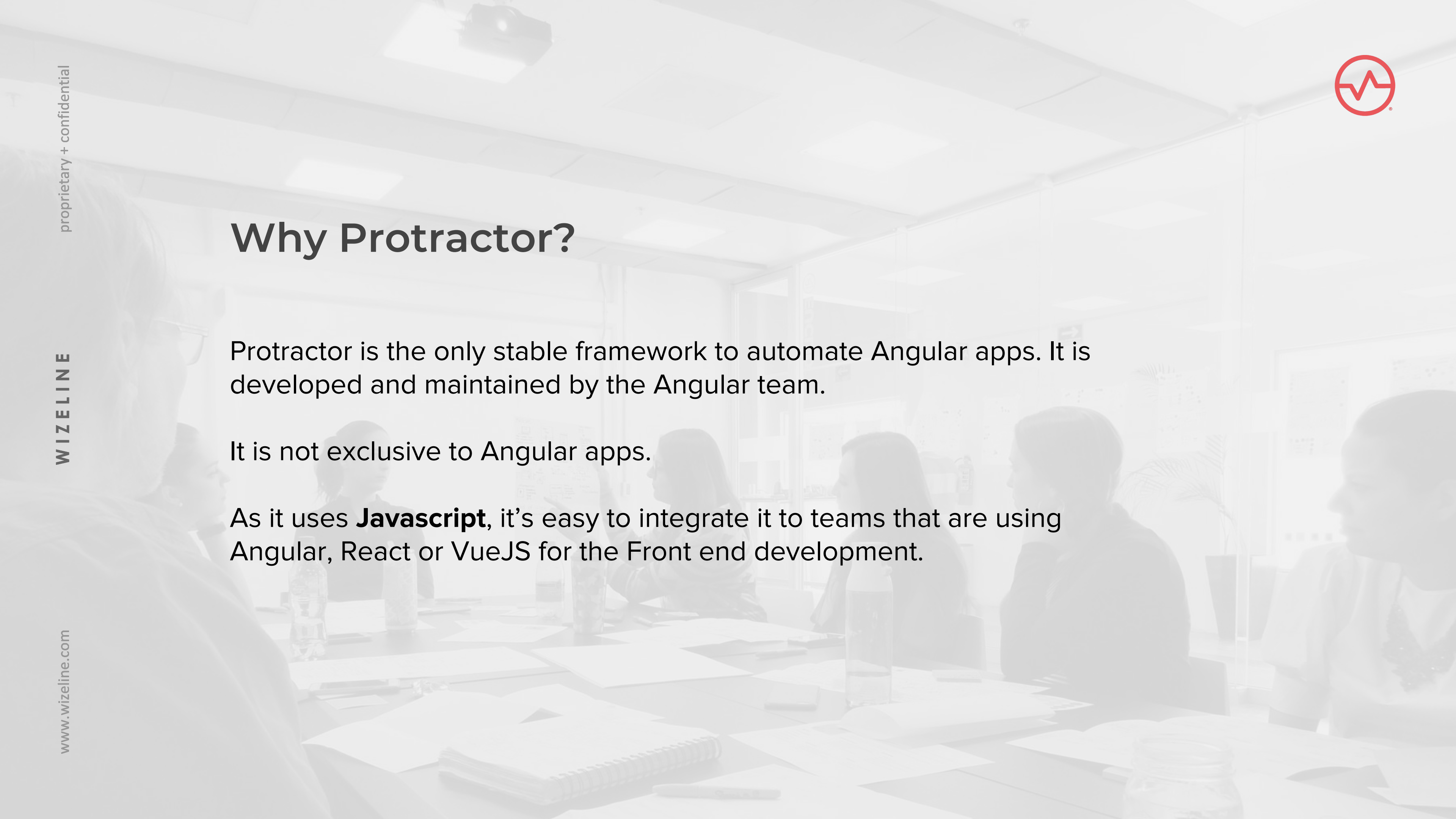
Frontend Automation

QA Automation Certification - Week 1

Automated End-to-End Testing

E2E or end-to-end or UI testing is a methodology used to test whether the flow of an application is performing as designed from start to finish. In simple words, it is testing of your application from the **user point of view** where the whole system is a black box with only the UI exposed to the user.

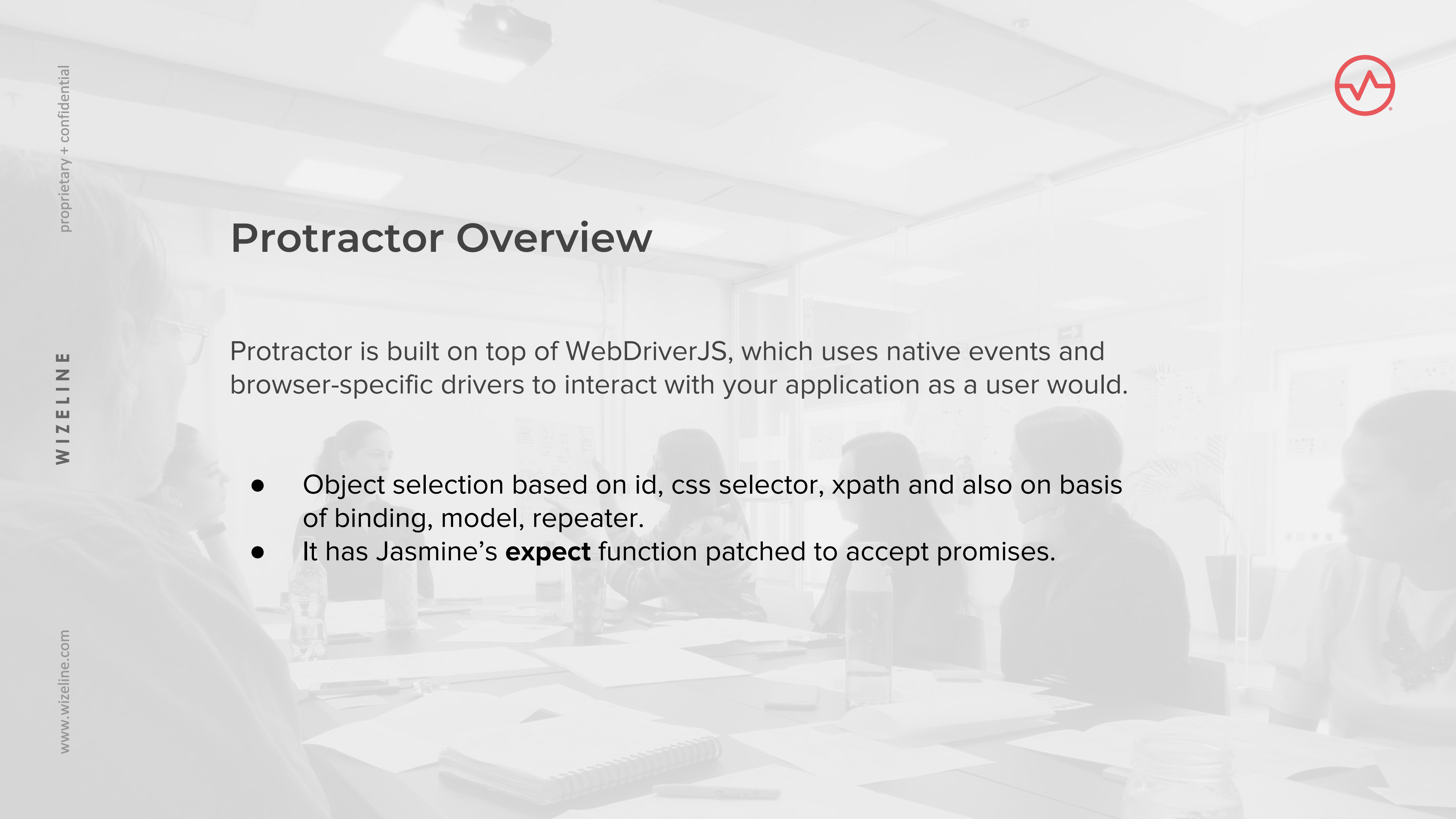


Why Protractor?

Protractor is the only stable framework to automate Angular apps. It is developed and maintained by the Angular team.

It is not exclusive to Angular apps.

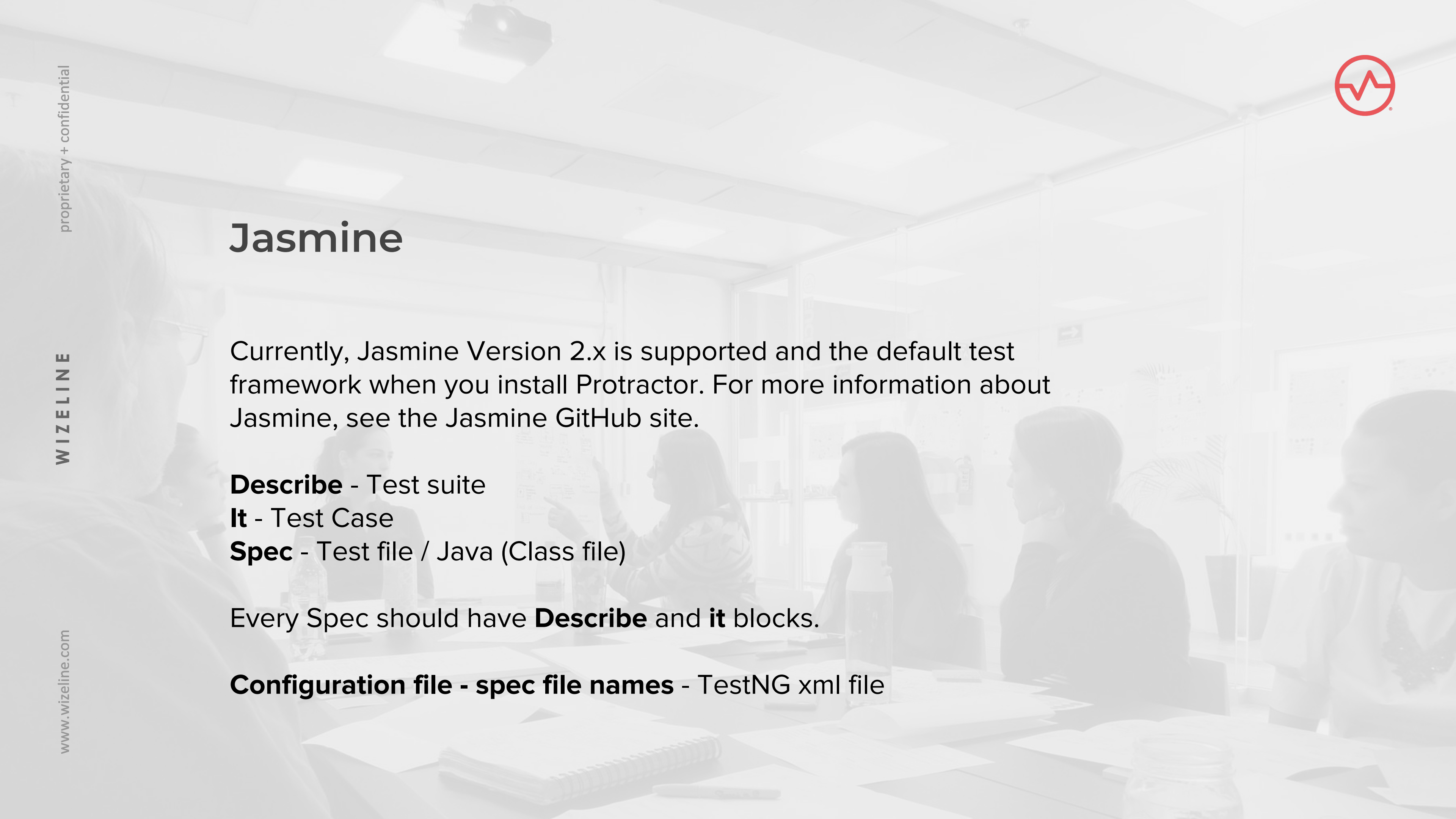
As it uses **Javascript**, it’s easy to integrate it to teams that are using Angular, React or VueJS for the Front end development.



Protractor Overview

Protractor is built on top of WebDriverJS, which uses native events and browser-specific drivers to interact with your application as a user would.

- Object selection based on id, css selector, xpath and also on basis of binding, model, repeater.
- It has Jasmine’s **expect** function patched to accept promises.



Jasmine

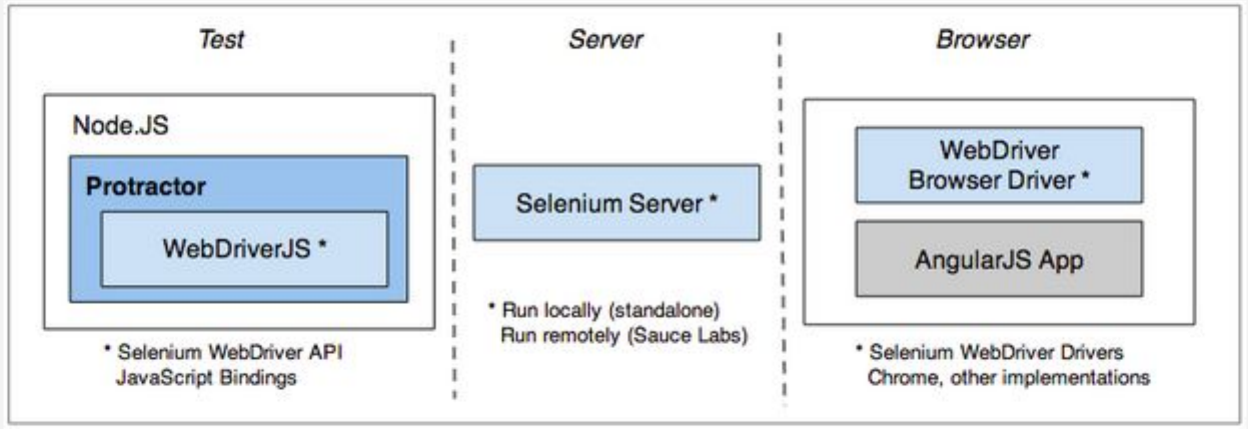
Currently, Jasmine Version 2.x is supported and the default test framework when you install Protractor. For more information about Jasmine, see the Jasmine GitHub site.

Describe - Test suite
It - Test Case
Spec - Test file / Java (Class file)

Every Spec should have **Describe** and **it** blocks.

Configuration file - spec file names - TestNG xml file

How does Protractor work?

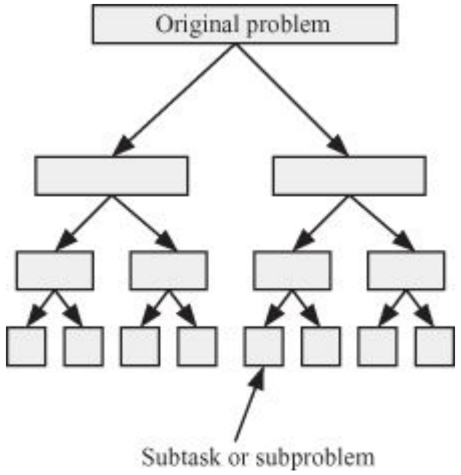

Test
Server
Browser
Node.JS
Protractor
WebDriverJS *
Selenium Server *
WebDriver Browser Driver *
AngularJS App
* Selenium WebDriver API JavaScript Bindings
* Run locally (standalone) Run remotely (Sauce Labs)
* Selenium WebDriver Drivers Chrome, other implementations

Automation: Best Practices

QA Automation Certification - Week 1

Start small

- Don’t rush to automate every test case.
- Identify test case priority, then automate.
- Don’t automate end-to-end.
- Divide and conquer.



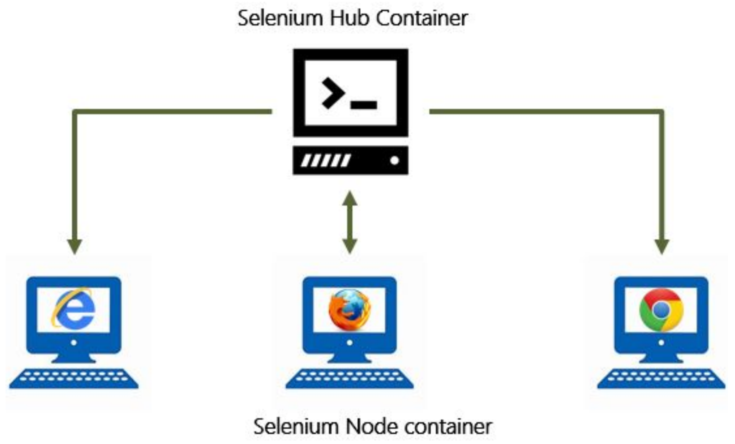
Test Constantly

ATLASSIAN
Bitbucket

- Don’t ever stop testing.
- Having tests run on every commit is going to provide significantly higher confidence in your software.
- You have unlimited time to test. Use it!

Parallelization

- Serialized tests take more time and cost more money.
- Using a Selenium Grid like Sauce Labs, parallelized tests can load multiple browsers for you or multiple tests simultaneously.
- 300 minute long tests can take either 5 hours or as little as 1 minute.



SAUCELABS

BrowserStack

Page Object Pattern

proprietary + confidential

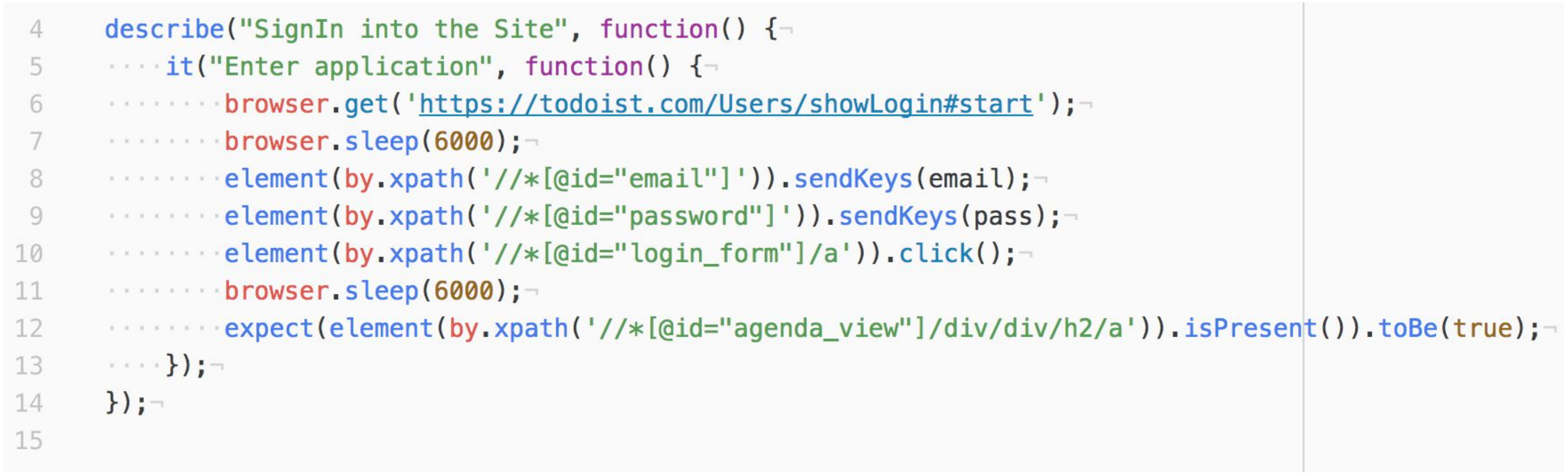
WIZELINE

www.wizeline.com

Page Object Pattern

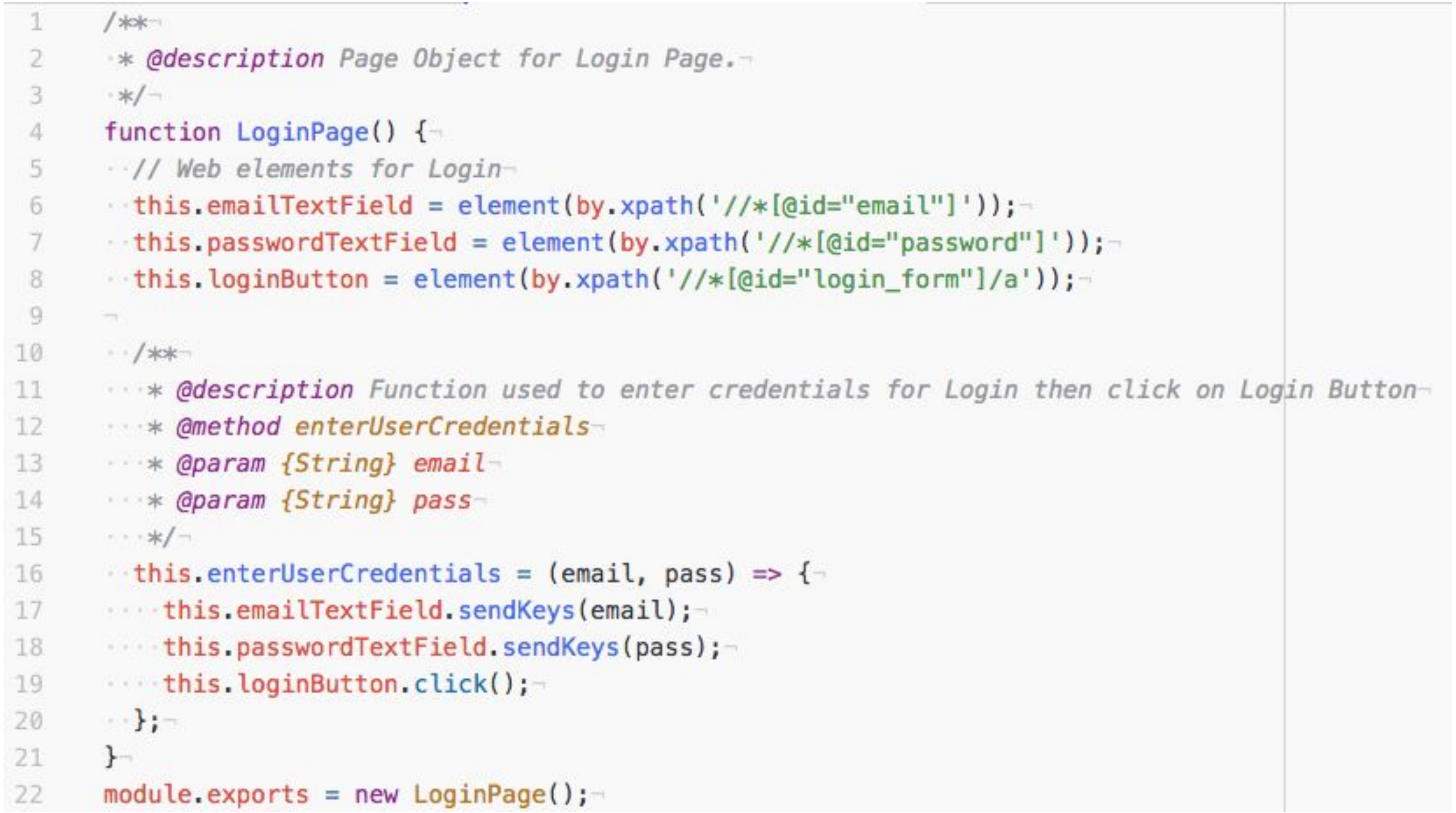
- Encourage reusable code throughout your test app.
- Test code is more organized for update if test parameters like selectors change.
- Tests are more readable.

test.js without Page Object Pattern



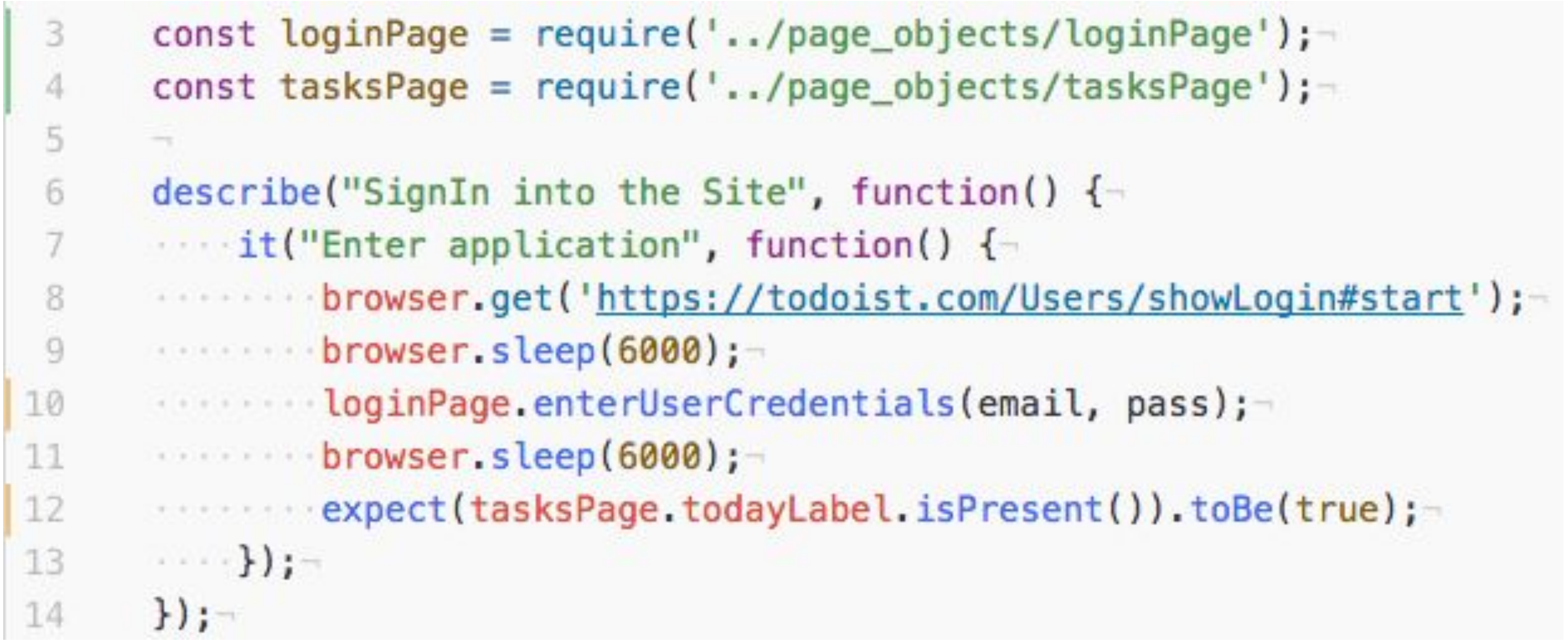
```
describe("SignIn into the Site", function() {
    it("Enter application", function() {
        browser.get('https://todoist.com/Users/showLogin#start');
        browser.sleep(6000);
        element(by.xpath('//*[@id="email"]')).sendKeys(email);
        element(by.xpath('//*[@id="password"]')).sendKeys(pass);
        element(by.xpath('//*[@id="login_form"]/a')).click();
        browser.sleep(6000);
        expect(element(by.xpath('//*[@id="agenda_view"]/div/div/h2/a')).isPresent()).toBe(true);
    });
});
```

loginPage.js



```
/**
 * @description Page Object for Login Page.
 */
function LoginPage() {
  // Web elements for Login
  this.emailTextField = element(by.xpath('//*[@id="email"]'));
  this.passwordTextField = element(by.xpath('//*[@id="password"]'));
  this.loginButton = element(by.xpath('//*[@id="login_form"]/a'));

  /**
   * @description Function used to enter credentials for Login then click on Login Button
   * @method enterUserCredentials
   * @param {String} email
   * @param {String} pass
   */
  this.enterUserCredentials = (email, pass) => {
    this.emailTextField.sendKeys(email);
    this.passwordTextField.sendKeys(pass);
    this.loginButton.click();
  };
}
module.exports = new LoginPage();
```

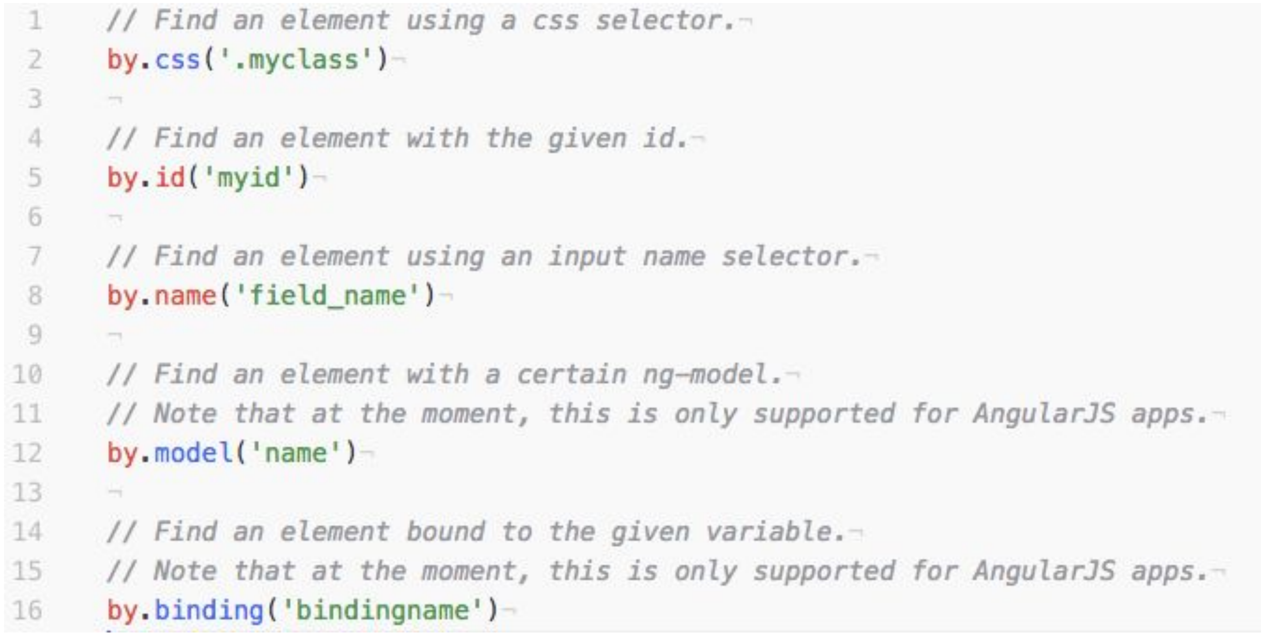
test.js with Page Object Pattern



```
const loginPage = require('../page_objects/loginPage');
const tasksPage = require('../page_objects/tasksPage');

describe("SignIn into the Site", function() {
    it("Enter application", function() {
        browser.get('https://todoist.com/Users/showLogin#start');
        browser.sleep(6000);
        loginPage.enterUserCredentials(email, pass);
        browser.sleep(6000);
        expect(tasksPage.todayLabel.isPresent()).toBe(true);
    });
});
```

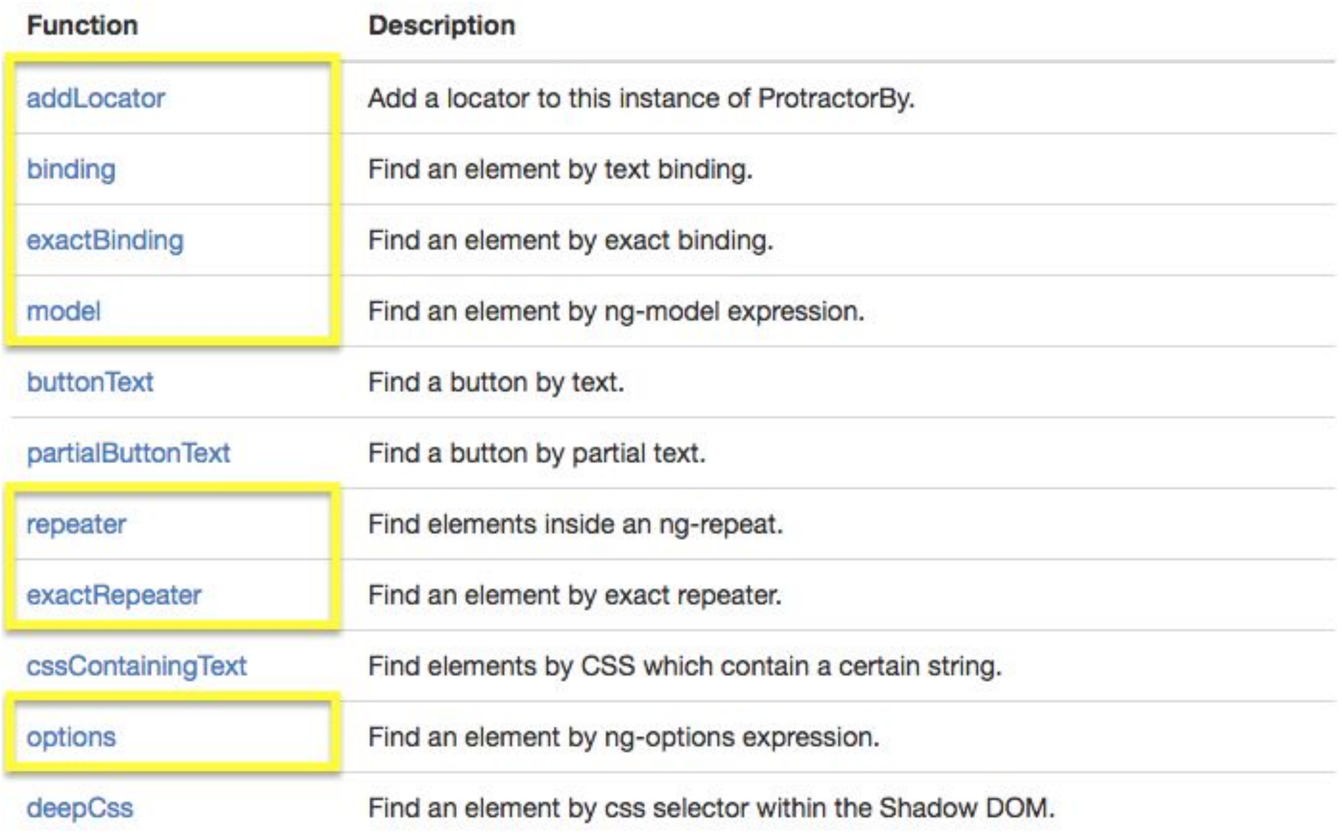
Correct use of locators

- ID
- Name
- CSS Selector
  - Tag and ID
  - Tag and class
  - Tag and attribute
  - Tag, class, and attribute
- XPath



```
// Find an element using a css selector.
by.css('.myclass')

// Find an element with the given id.
by.id('myid')

// Find an element using an input name selector.
by.name('field_name')

// Find an element with a certain ng-model.
// Note that at the moment, this is only supported for AngularJS apps.
by.model('name')

// Find an element bound to the given variable.
// Note that at the moment, this is only supported for AngularJS apps.
by.binding('bindingname')
```

Protractor locators

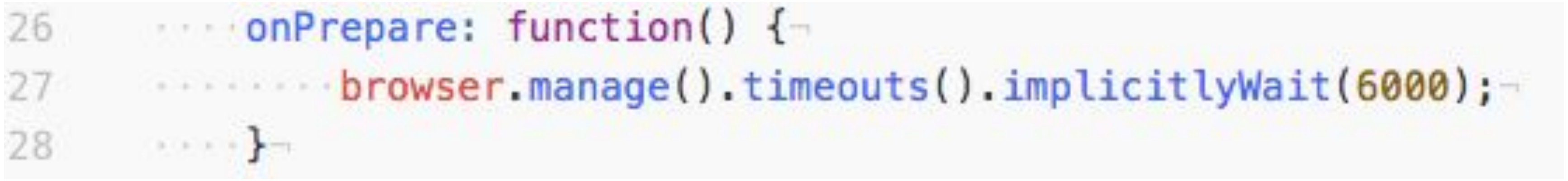


Function	Description
addLocator	Add a locator to this instance of ProtractorBy.
binding	Find an element by text binding.
exactBinding	Find an element by exact binding.
model	Find an element by ng-model expression.
buttonText	Find a button by text.
partialButtonText	Find a button by partial text.
repeater	Find elements inside an ng-repeat.
exactRepeater	Find an element by exact repeater.
cssContainingText	Find elements by CSS which contain a certain string.
options	Find an element by ng-options expression.
deepCss	Find an element by css selector within the Shadow DOM.

Don’t forget to wait!

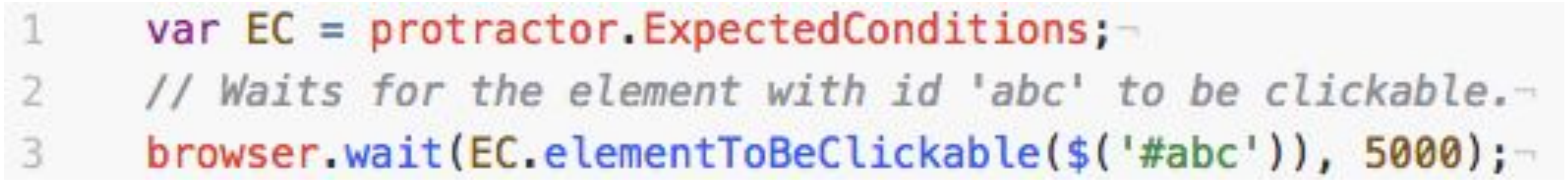
- Web pages don’t load instantaneously, and not all elements load at the same time.
- A lot of your failures in finding an element will disappear if you use these.
- Explicit waits are preferred for waiting for single elements.
- Implicit waits for waiting all the time.

Implicit Wait



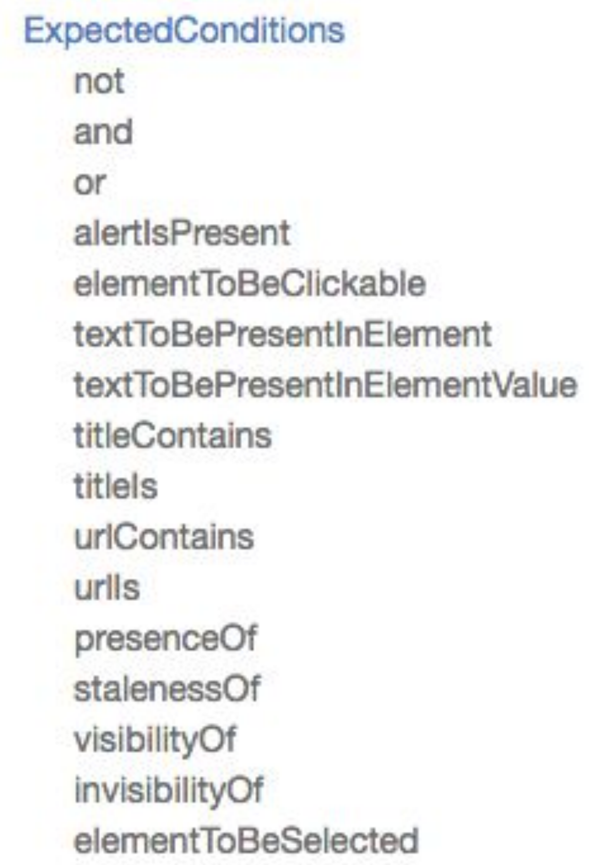
```
    onPrepare: function() {
        browser.manage().timeouts().implicitlyWait(6000);
    }
```

Explicit wait



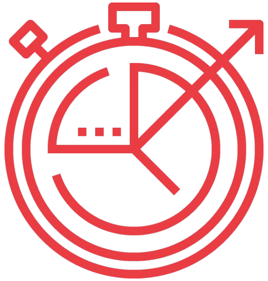
```
var EC = protractor.ExpectedConditions;
// Waits for the element with id 'abc' to be clickable.
browser.wait(EC.elementToBeClickable($('#abc')), 5000);
```

Protractor Expected Conditions



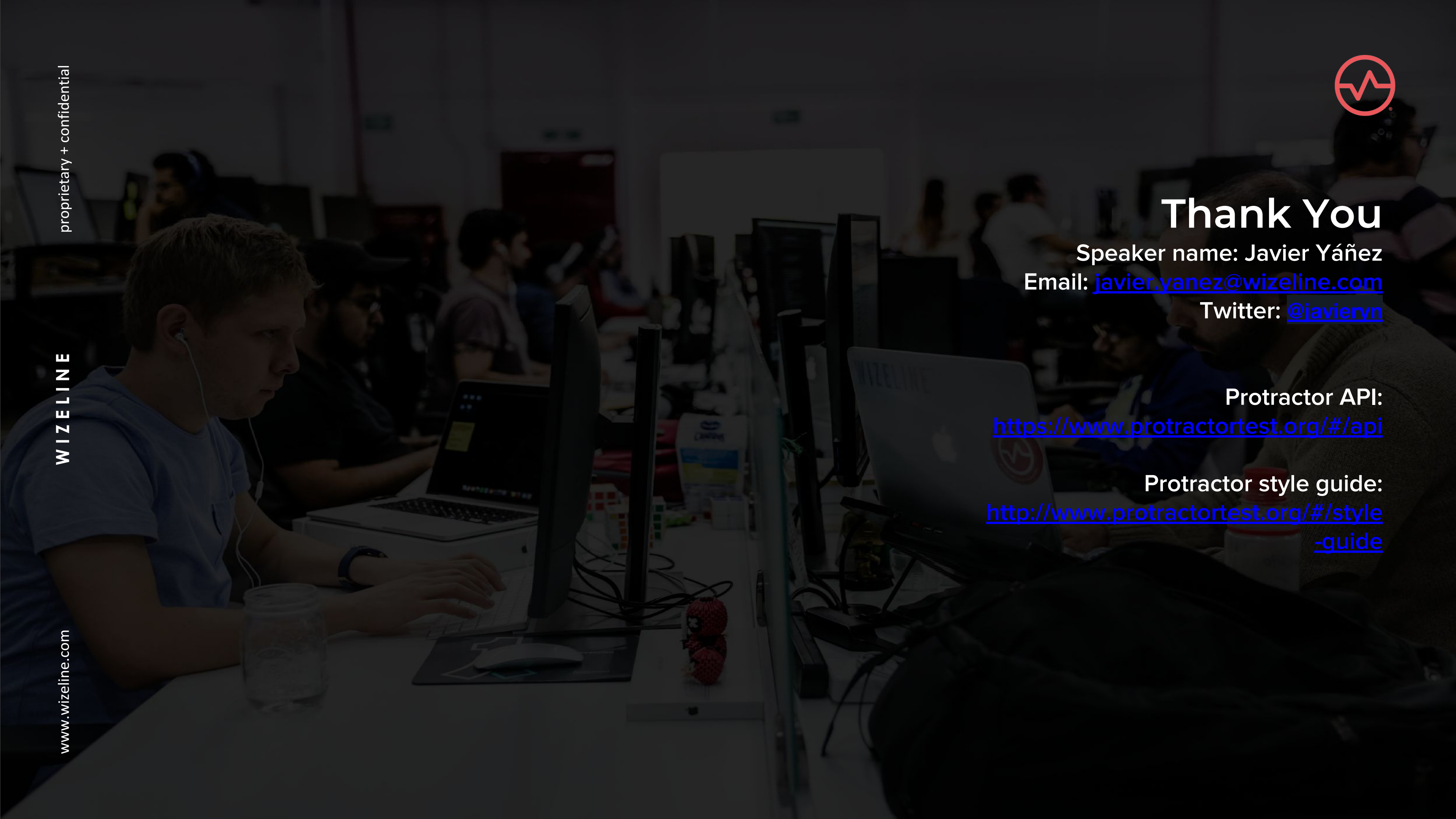
ExpectedConditions
- not
- and
- or
- alertIsPresent
- elementToBeClickable
- textToBePresentInElement
- textToBePresentInElementValue
- titleContains
- titleIs
- urlContains
- urlIs
- presenceOf
- stalenessOf
- visibilityOf
- invisibilityOf
- elementToBeSelected

Collect metrics



- How long did test runs take before and after automation?
- How many bugs do automated tests identify per release?
- How many engineers does a test run require?

Q&A



Thank You

Speaker name: Javier Yáñez
Email: javier.yanez@wizeline.com
Twitter: @javieryn

Protractor API:
https://www.protractortest.org/#/api

Protractor style guide:
http://www.protractortest.org/#/style-guide

Survival kit

Github

- Have an account.
- Have your git handle.
- Repository: https://github.com/wizelineacademy/qa-automation

Frontend Testing

- install node: https://www.npmjs.com/get-npm
- install protractor: https://www.npmjs.com/package/protractor
- install webdriver manager plugin: https://www.npmjs.com/package/webdriver-manager

Javascript IDE:

- Atom: https://ide.atom.io/
- Visual Studio Code: https://code.visualstudio.com/

Todoist

CREATE AN ACCOUNT HERE!
https://todoist.com/Users/showRegister