Building a testing project from scratch

Aleksandr Martiushov

About me

- 8 years of QA
  - backend
  - mobile
  - UI
- 1 year of Backend Dev
- Like conferences
- Like speaking on the conferences
  - Heisenbug
  - SeleniumConf
  - SQA Days
- worked in CERN
- https://twitter.com/amartyushov
- https://www.linkedin.com/in/amartyushov

Setting up a topic

- Talk about architecture of **e2e** testing projects.

Setting up a topic

- Talk about architecture of e2e testing projects.
- Talk about useful tools.

Setting up a topic

- Talk about architecture of e2e testing projects.
- Talk about useful tools.
- Show some code

Plan for today

Plan for today

- Having artificial project establish

  backend testing

Plan for today

- Having artificial project establish **backend testing**
- Continue evolution of the test project to **UI testing**

Backend testing plan

- Core
- Create first test
- Test data preparation
- Reporting
- Properties
- Dependency injection
- CI execution
- Env deployment

Backend testing plan

- **Core**
- Create first test
- Test data preparation
- Reporting
- Properties
- Dependency injection
- CI execution
- Env deployment

Project is starting

We have to establish an **architecture** of the project

Core of architecture should be **independent** of tooling

Core

Good pattern of separation,

which is based on usual test flow

- (Arrange) **Prepare** condition for the action
- (Act) **Perform** action
- (Assert) **Check** results

More concrete for **API** testing

In terms of api testing:

- Prepare condition for the action => **Dto providers**

More concrete for **API** testing

In terms of api testing:

- Prepare condition for the action => **Dto providers**
- Perform action => **Request executors**

More concrete for **API** testing

In terms of api testing:

- Prepare condition for the action => **Dto providers**
- Perform action => **Request executors**
- Check results => **Result checkers**

Provider

- Convenient methods to **build Dto** for requests

Provider

- Convenient methods to **build Dto** for requests
- Have a method for building a Dto with **default values**

Provider

- Convenient methods to **build Dto** for requests
- Have a method for building a Dto with **default values**
- Swagger is useful (as usual)

Provider

```
public class TeacherProvider {
    int id = 1;

    public Teacher buildDefaultTeacher() {
        Teacher teacher = new Teacher();
        int randomInt = new Random().nextInt( bound: 70);

        teacher.setId(id++);
        teacher.setAge(randomInt);
        teacher.setFirstName("first_name");
        teacher.setLastName("last_name");
        teacher.setSpeciality("Math");
        return teacher;
    }
```

Executor

- Convenient methods to **invoke** end-points

Executor

- Convenient methods to **invoke** end-points
- Method impl: RestAssured, swagger

Executor

- Convenient methods to **invoke** end-points
- Method impl: RestAssured, swagger
- **Logging** is useful at this step, which Dtos are sent to end-points

Executor

```
public Teacher get(Integer id){
    log.info("Get teacher with id: " + id);
    return RestAssured
            .given() RequestSpecification
            .pathParam(ID, id) RequestSpecification
            .get( path: PATH + "/{id}") Response
            .then() ValidatableResponse
            .extract().as(Teacher.class);
}
```

Checker

- **Complex** assertions should be extracted to methods

Checker

- **Complex** assertions should be extracted to methods
- Simple assertions can stay in the test
  - the idea is to **quickly** understand what is checked

Checker

- **Complex** assertions should be extracted to methods
- Simple assertions can stay in the test
  - the idea is to **quickly** understand what is checked
- Grey zone

Backend testing plan

- Core
- **Create first test**
- Test data preparation
- Reporting
- Properties
- Dependency injection
- CI execution
- Env deployment

Time to execute test

Time to choose test execution framework

(We were postponing this decision so far).

Body of the test

Provider - Executor - Checker

perfect fit for

Arrange - Act - Assert

Why? You read logic of the test but not implementation of each step.

Body of the test

```
@Test
public void canCreateTeacher() {

    // Arrange
    String name = "ExplicitName";
    Teacher teacher = teacherProvider.buildDefaultTeacher()
            .setFirstName(name);

    // Act
    Teacher createdTeacher = teacherExecutor.create(teacher);

    // Assert
    teacherChecker.hasName(createdTeacher, name);
}
```

Backend testing plan

- Core
- Create first test
- **Test data preparation**
- Reporting
- Properties
- Dependency injection
- CI execution
- Env deployment

Test data preparation

Rules written by pain:

- always **generate test data** in isolation for a test or group of tests
  - forget about any shared database dumps
    - it does not work for **multibranch** test creation
    - info about dump **internals** is lost
    - dump **can be lost**
    - tests are **fragile**, you can affect other tests

Test data preparation

Rules written by pain:

- test data preparation must be **idempotent**, you need to have a possibility to execute test multiple times.

Test data preparation

Rules written by pain:

- use only **API** for data preparation, do not go to database directly (do not depend on impl of the database)

Test data preparation

In case of TestNG/JUnit:

- **do not use data_providers**
  - not to break AAA principle
  - exception: one/two input parameters

Test data preparation

In case of TestNG/JUnit:

- **do not use data_providers**
  - not to break AAA principle
  - exception: one/two input parameters
- if you still want data_providers
  - https://github.com/sskorol/test-data-supplier
  - not to work with ugly array of Objects

Test data preparation

```
@DataSupplier
public Stream<User> getData() {
    return Stream.of(
        new User("Petya", "password2"),
        new User("Virus Petya", "password3"),
        new User("Mark", "password1"))
            .filter(u -> !u.getName().contains("Virus"))
            .sorted(comparing(User::getPassword));
}

@Test(dataProvider = "getData")
public void shouldSupplyStreamData(final User user) {
    // ...
}
```

https://github.com/sskorol/test-data-supplier

Test data preparation

In case of TestNG/JUnit hooks **should not be multilevel** deep, it leads to

- tests **nest unnecessary** data preparation steps, just because of class hierarchy

Test data preparation

In case of TestNG/JUnit hooks **should not be multilevel** deep, it leads to

- tests **nest unnecessary** data preparation steps, just because of class hierarchy
- test execution can be **unnecessary longer**

Test data preparation

In case of TestNG/JUnit hooks **should not be multilevel** deep, it leads to

- tests **nest unnecessary** data preparation steps, just because of class hierarchy
- test execution can be **unnecessary longer**
- hard to understand **what steps** are actually **executed**, especially having tags, groups

Backend testing plan

- Core
- Create first test
- Test data preparation
- **Reporting**
- Properties
- Dependency injection
- CI execution
- Env deployment

Reporting

As **number** of tests is **growing** =>

it is time to introduce helpful **reporting**.

What are the **criteria** for reporting tool?

Reporting requirements

- to **see** execution **steps** of the test

Reporting requirements

- to **see** execution **steps** of the test
- have some **meta info** of the test
  - **severity**
  - **link** to tickets and defects
  - **group** tests by features, epics

Reporting requirements

- to **see** execution **steps** of the test
- have some **meta info** of the test
- to have a particular **test history on ci** between builds

Reporting requirements

- to **see** execution **steps** of the test
- have some **meta info** of the test
- to have a particular **test history on ci** between builds
- to have **same** reporting for diff kind of testing frameworks

Reporting requirements

- to **see** execution **steps** of the test
- have some **meta info** of the test
- to have a particular **test history on ci** between builds
- to have **same** reporting for diff kind of testing frameworks
- to have a **build parameters** in the report

Reporting requirements

- to **see** execution **steps** of the test
- have some **meta info** of the test
- to have a particular **test history on ci** between builds
- to have **same** reporting for diff kind of testing frameworks
- to have a **build parameters** in the report
- properly handle **test retry results**

Allure

https://github.com/allure-framework/allure2

Allure

- to **see** execution **steps** of the test

```
@Step("HTTP. Get teacher with id: {0}")
public Teacher get(Integer id){
    log.info("Get teacher with id: " + id);
    return RestAssured
```

Allure

- to **see** execution **steps** of the test

```
@BeforeClass(description = "Create initial teachers")
public void setUp() {
    teacherExecutor.deleteAll();

    Teacher teacher1 = teacherProvider.buildDefaultTeacher().setFirstName("Alex");
    Teacher teacher2 = teacherProvider.buildDefaultTeacher().setFirstName("David");
    teacherExecutor.create(teacher1);
    teacherExecutor.create(teacher2);
}
```

Allure

- to add **meta info** for the test (title)

```
@Test(description = "Check all first names of teachers")
public void getAllTeachers_returnCorrectList() {
```

Allure

- to add **meta info** for the test (description)

```
@Test(description = "Check all first names of teachers")
@Description("Test gets all teachers and verifies first names of returned teachers")
public void getAllTeachers_returnCorrectList() {
```

Allure

- to add **meta info** for the test (severity)

```
@Test
@Severity(SeverityLevel.BLOCKER)
public void getAllTeachers_returnCorrectList() {
```

Allure

- to add **meta info** for the test (link to the defect)

```
@Test
@Issue("DEFECT-123")
public void getAllTeachers_returnCorrectList() {
```

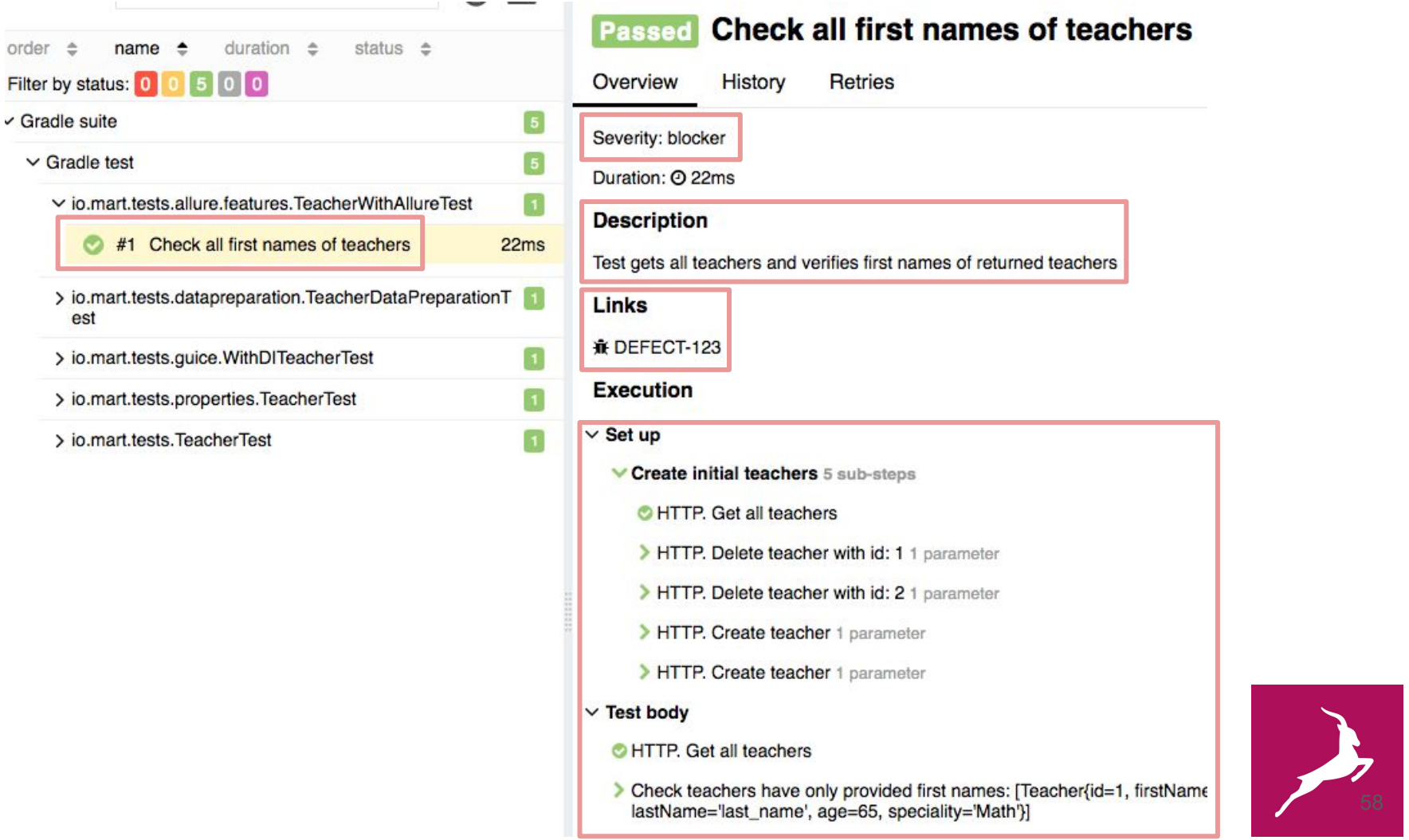
Allure

- to add **meta info** for the test (feature mapping)

```
@Test
@Feature("Teacher CRUD")
public void getAllTeachers_returnCorrectList() {
```

Allure

- Add build params, env params to the report
  - just create a **file** key=value
  - environment.properties
  - build/allure-results
  - e.g. use After* hook for it


order
name
duration
status
Filter by status: 0 0 5 0 0
Gradle suite 5
Gradle test 5
io.mart.tests.allure.features.TeacherWithAllureTest 1
#1 Check all first names of teachers 22ms
io.mart.tests.datapreparation.TeacherDataPreparationTest 1
io.mart.tests.guice.WithDITeacherTest 1
io.mart.tests.properties.TeacherTest 1
io.mart.tests.TeacherTest 1
Passed
Check all first names of teachers
Overview
History
Retries
Severity: blocker
Duration: 22ms
Description
Test gets all teachers and verifies first names of returned teachers
Links
DEFECT-123
Execution
Set up
Create initial teachers 5 sub-steps
HTTP. Get all teachers
HTTP. Delete teacher with id: 1 1 parameter
HTTP. Delete teacher with id: 2 1 parameter
HTTP. Create teacher 1 parameter
HTTP. Create teacher 1 parameter
Test body
HTTP. Get all teachers
Check teachers have only provided first names: [Teacher{id=1, firstName
lastName='last_name', age=65, speciality='Math'}]

Backend testing plan

- Core
- Create first test
- Test data preparation
- Reporting
- **Properties**
- Dependency injection
- CI execution
- Env deployment

What do we need from properties management

- type safety

What do we need from properties management

- type safety
- ability to **override** values using environment properties

What do we need from properties management

- type safety
- ability to **override** values using environment properties
- **placeholders** resolution

What do we need from properties management

- type safety
- ability to **override** values using environment properties
- **placeholders** resolution
- **easy** configuration

Property management

The options are:

- **Typesafe** (https://github.com/lightbend/config)
- Apache configuration (https://commons.apache.org/proper/commons-configuration/)

Configuration

1. testCompile **group**: **'com.typesafe'**, **name**: **'config'**
2. src/test/resources/application.conf
3. Config config = ConfigFactory.load();
4. config.getString(“base.url”);

Backend testing plan

- Core
- Create first test
- Test data preparation
- Reporting
- Properties
- **Dependency injection**
- CI execution
- Env deployment

Why DI and choice

- **Guice**
- Spring

Configuration

```
public class CustomGuiceModule extends AbstractModule {

    @Override
    protected void configure() {
        Config config = ConfigFactory.load();

        bind(Config.class).toInstance(config);
    }
}
```

Configuration

```
@Guice(modules = { CustomGuiceModule.class })
public class WithDITeacherTest {

    @Inject
    TeacherProvider teacherProvider;
```

Backend testing plan

- Core
- Create first test
- Test data preparation
- Reporting
- Properties
- Dependency injection
- **CI execution**
- Env deployment

CI execution

- **QA** may/should take **care** of it

CI execution

- **QA** may/should take **care** of it
- if (Jenkins) **keep** Jenkins file **in tests**

CI execution

- **QA** may/should take **care** of it
- if (Jenkins) **keep** Jenkins file **in tests**
- provide **possibility** to **debug** tests on ci
  - **flags** like deploy/run_with_debug/undeploy

CI execution

- **QA** may/should take **care** of it
- if (Jenkins) **keep** Jenkins file **in tests**
- provide **possibility** to **debug** tests on ci
  - **flags** like deploy/run_with_debug/undeploy
- informative **logging** and **reporting**
  - gather logs from application (tips about it https://www.youtube.com/watch?v=ZQ0hjf3P-FM

    My Story of Microservices Testing)

Backend testing plan

- Core
- Create first test
- Test data preparation
- Reporting
- Properties
- Dependency injection
- CI execution
- **Env deployment**

Deployment of the test env for the job

- **QA** may/should take **care** of it

Deployment of the test env for the job

- **QA** may/should take **care** of it
- easy to start with **docker**/docker-compose

Deployment of the test env for the job

- **QA** may/should take **care** of it
- easy to start with **docker**/docker-compose
- create **separate env** for each job

Deployment of the test env for the job

- **QA** may/should take **care** of it
- easy to start with **docker**/docker-compose
- create **separate env** for each job
- keep deployment **script in tests**

Backend testing plan. Summary

- Core
- Create first test
- Test data preparation
- Reporting
- Properties
- Dependency injection
- CI execution
- Env deployment

Going up on testing pyramid

UI testing is requested to **establish** in the team

And we already have a **good background** of testing project.

(SUT is a fake page https://www.saucedemo.com/)

UI testing plan

- PageObjects
- Selenide
- Selenide + Allure
- Selenoid

UI testing plan

- **PageObjects**
- Selenide
- Selenide + Allure
- Selenoid

Creating PageObjects

- real quick **why** do we need **PageObject**
  - easy to **read flow** of the test
  - easy to **maintain** tests
  - **abstracts** tooling from the test

Creating PageObjects

- real quick **why** do we need **PageObject**
- you will always find this pattern in UI tests

Creating PageObjects

- real quick **why** do we need **PageObject**
- you will always find this pattern in UI tests
- using pure **Selenium** does **not** make **sense** any more

Creating PageObjects

- real quick **why** do we need **PageObject**
- you will always find this pattern in UI tests
- using pure **Selenium** does **not** make **sense** any more
- **competition** between frameworks is a **speed** of **creating** these pages + **speed** of selenium **configuration**

Creating PageObjects

- real quick **why** do we need **PageObject**
- you will always find this pattern in UI tests
- using pure **Selenium** does **not** make **sense** any more
- **competition** between frameworks is a **speed** of **creating** these pages + **speed** of selenium **configuration**
- Codeborne: **Selenide** https://selenide.org/index.html
- Yandex: Atlas https://github.com/qameta/atlas
- Epam: JDI https://jdi.epam.com/

UI testing plan

- PageObjects
- **Selenide**
- Selenide + Allure
- Selenoid

making PageObjects alive using Selenide

- **minimum** selenium configuration

making PageObjects alive using Selenide

- **minimum** selenium configuration
- a lot of **boilerplate** code is **hidden**

making PageObjects alive using Selenide

- **minimum** selenium configuration
- a lot of **boilerplate** code is **hidden**
- embedded **waits** by default

making PageObjects alive using Selenide

- **minimum** selenium configuration
- a lot of **boilerplate** code is **hidden**
- embedded **waits** by default
- webdriver instance is stored in **thread safe wrapper** (com.codeborne.selenide.WebDriverRunner)

PageObject

```
public class LoginPage {

    @FindBy(id = "user-name")
    private SelenideElement login_field;

    public LoginPage inputName(String userName){
        login_field.shouldBe(visible).setValue(userName);
        return this;
    };
```

Simple test example

```
@BeforeClass
public void setUp() {
    Config config = ConfigFactory.load();

    Configuration.browser = config.getString( path: "browser.type");
    Configuration.baseUrl = config.getString( path: "base.path");
    Configuration.browserCapabilities = prepareCapabilities();
}

@Test
public void checkLoginWithCorrectCredentials_succeeds() {
    // Arrange
    LoginPage loginPage = Selenide.open( relativeOrAbsoluteUrl: "/", LoginPage.class);

    // Act
    InventoryPage inventoryPage = loginPage
            .inputName("standard_user")
            .inputPassword("secret_sauce")
            .pressLogin();

    // Assert
    inventoryPage.numberOfProductsIs(6);
}
```

UI testing plan

- PageObjects
- Selenide
- **Selenide + Allure**
- Selenoid

Add step titles to pages

```
@Step("Set login \"{0}\" at login page")
public LoginPageWithAllure inputName(String userName){
    login_field.shouldBe(visible).setValue(userName);
    return this;
};

@Step("Set password \"{0}\" at login page")
public LoginPageWithAllure inputPassword(String password) {
    password_field.shouldBe(visible).setValue(password);
    return this;
}

@Step("Pressing login button")
public InventoryPage pressLogin() {
    login_btn.click();
    return Selenide.page(InventoryPage.class);
}
```

Integrate allure and selenide

- **group**: **'io.qameta.allure'**, **name**: **'allure-selenide'**
- SelenideLogger.*addListener*(**"AllureSelenide"**,
  new AllureSelenide()
  .screenshots(**true**)
  .savePageSource(**false**));

Integrate allure and selenide

- **group**: **'io.qameta.allure'**, **name**: **'allure-selenide'**
- SelenideLogger.*addListener*(**"AllureSelenide"**,
  new AllureSelenide()
  .screenshots(**true**)
  .savePageSource(**false**));
- additional **technical info** in Allure report about test flow
- **screenshot** on failing tests

Test body

$(open) https://www.saucedemo.com/ 118ms

Set login "standard_user" at login page 1 parameter, 2 sub-steps 108ms

Set password "secret_sauce" at login page 1 parameter, 2 sub-steps 117ms

password secret_sauce

$(By.id: password) should be(visible) 31ms

$(By.id: password) set value(secret_sauce) 85ms

Pressing login button 1 sub-step 128ms

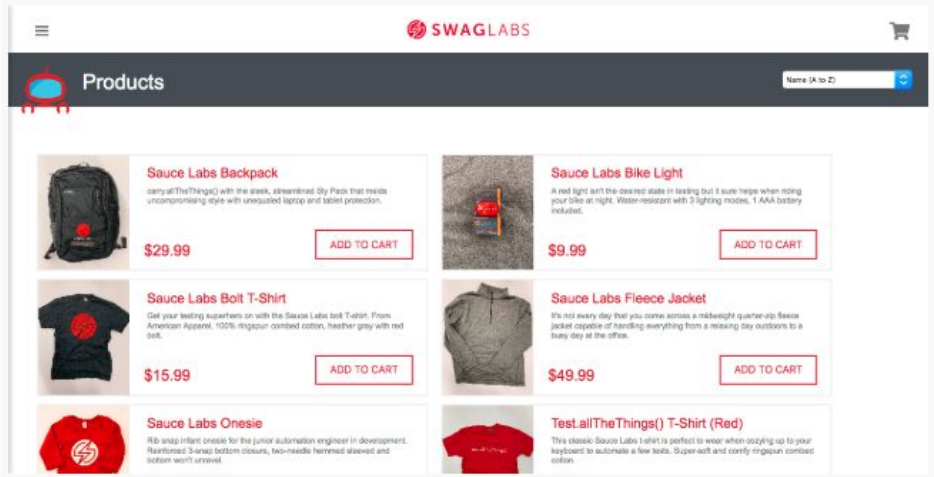
$(.btn_action) click() 127ms

Check number of products on inventory page is 5 1 parameter, 1 sub-step, 1 attachment 5s 222ms

number 5

$(.inventory_item) should have(size(5)) 1 attachment 5s 219ms

Screenshot 388.6 KB



Screenshot comparison

- If there is a goal to perform screenshot comparison
  - https://github.com/pazone/ashot
    - threshold
    - crop
    - compare elements only

Sidenotes about UI test classes

- **no** usage of **webdriver** in tests, only through PageObjects

Sidenotes about UI test classes

- **no** usage of **webdriver** in tests, only through PageObjects
- same ideas about **test data** generation before tests
  - using api (recommended)

Sidenotes about UI test classes

- **no** usage of **webdriver** in tests, only through PageObjects
- same ideas about **test data** generation before tests
  - using api (recommended)
- Test **retry** is a cool feature to “greenify” tests, but it can **increase** execution time and hide bugs

Sidenotes about UI test classes

- **no** usage of **webdriver** in tests, only through PageObjects
- same ideas about **test data** generation before tests
  - using api (recommended)
- Test **retry** is a cool feature to “greenify” tests, but it can **increase** execution time and hide bugs
- no Thread.sleep

Sidenotes about UI test classes

- **no** usage of **webdriver** in tests, only through PageObjects
- same ideas about **test data** generation before tests
  - using api (recommended)
- Test **retry** is a cool feature to “greenify” tests, but it can **increase** execution time and hide bugs
- no Thread.sleep
- try to avoid **if statements** in PageObjects as this is a source of **flaky** tests

UI testing plan

- PageObjects
- Selenide
- Selenide + Allure
- **Selenoid**

UI remote test execution

- Selenium **Grid** has many disadvantages:
  - frozen **sessions**
  - **no visual** control of the browser
  - **fragile**
  - consumes a lot of **memory** for just proxying

UI remote test execution

- Selenium **Grid** has many disadvantages
- My choice is **Selenoid**:
  - Proxy written in GO
  - dockerized browsers
  - UI

UI remote test execution

- Selenium **Grid** has many disadvantages
- My choice is **Selenoid**
- Ability to build cluster with up to **5k** running sessions

UI remote test execution

- Selenium **Grid** has many disadvantages
- My choice is **Selenoid**
- Ability to build cluster with up to **5k** running sessions
- Video recording

Selenoid articles

- all articles are here https://aerokube.com/
- also for Windows browsers on Windows vms
- the way to dockerize Windows + browsers
- also works for Android emulators

How to install https://aerokube.com/cm/latest/

UI testing summary

- **Quickly** create project, configure and execute

UI testing summary

- **Quickly** create project, configure and execute
- Misuse webdriver => flaky tests

UI testing summary

- **Quickly** create project, configure and execute
- Misuse webdriver => flaky tests
- Opposite approaches of writing tests like
  - https://github.com/GoogleChrome/puppeteer
  - https://github.com/DevExpress/testcafe
  - https://www.cypress.io/
  - trying to conquer the market

Sources

https://github.com/amartyushov/signavioQAmeetup2019

@amartyushov