DERIVCO

DELIVERING A LEGENDARY GAMING EXPERIENCE

ESTONIA

Functional autotests development with asynchronous JavaScript

Sergei Chipiga

2017

Sergei Chipiga Derivco Senior Automation Engineer

recent work experience

DERIVCO

Yandex

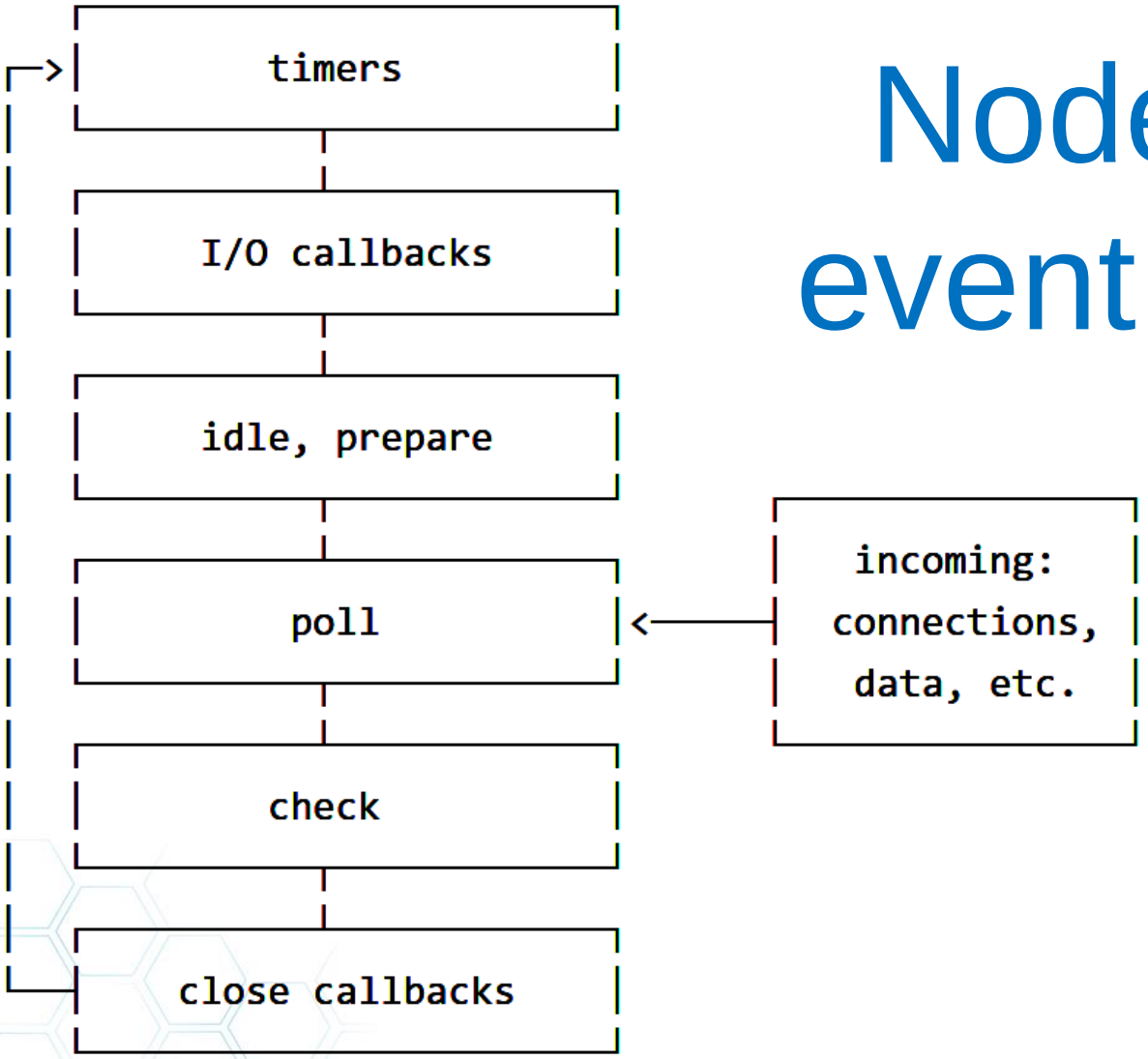
MIRANTIS
Pure Play OpenStack®

Agenda

1. Asynchronous problem in autotests
2. Selenium framework async solutions
3. MochaJS tactics, specifics and pitfalls
4. Yet another test development approach
5. Yet another functional testing framework

JavaScript is asynchronous

NodeJS event loop



JavaScript test example

```
suite("Browser tests", () => {
    test("should open home page", () => {
        openBrowser("chrome");
        gotoPage("https://derivco.com/");
        closeBrowser();
    });
});
```

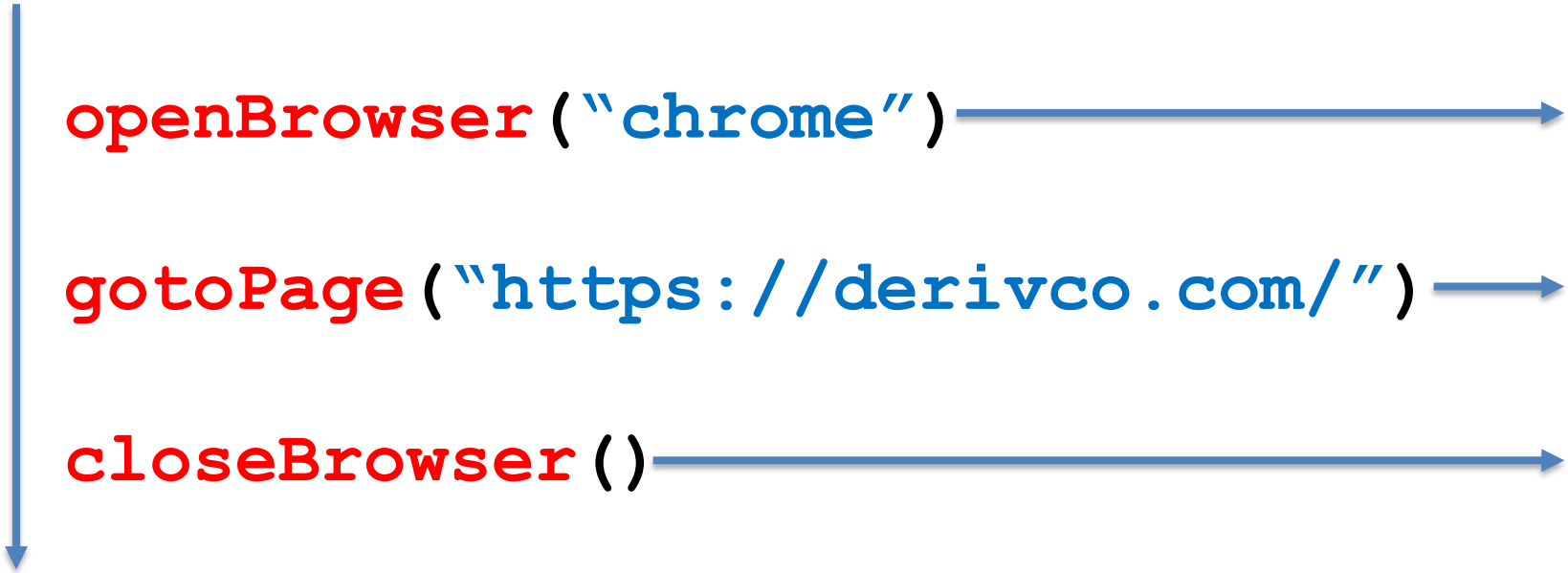
Calls order in test

```
openBrowser("chrome")
```

```
gotoPage("https://derivco.com/")
```

```
closeBrowser()
```

Calls order in test



selenium-webdriver **async** solution

```
var driver = new webdriver.Builder().forBrowser('firefox').build();
driver.get('http://www.google.com/ncr');
driver.findElement(By.name('q')).sendKeys('webdriver');
driver.findElement(By.name('btnG')).click();
driver.wait(until.titleIs('webdriver - Google Search'), 1000);
driver.quit();
```

CodeceptJS **async** solution

```
Scenario('submit form successfully', I => {
    I.amOnPage('/documentation')
    I.fillField('Email', 'hello@world.com')
    I.fillField('Password', '123456')
    I.checkOption('Active')
    I.checkOption('Male');
    I.click('Create User')
    I.see('User is valid')
    I.dontSeeInCurrentUrl('/documentation')
});
```

selenium-webdriver / CodeceptJS **async** solution

+ looks like sequential code execution;
+ doesn’t need to understand async specifics;

- is not normal for JavaScript;
- works with this library only;
- very complicated ~~magic~~ logic under hood;

DERIVCO

Nightwatch.js **async** solution

```
browser
   .url('http://www.google.com')
   .waitForElementVisible('body', 1000)
   .setValue('input[type=text]', 'rembrandt van rijn')
   .waitForElementVisible('button[name=btnG]', 1000)
   .click('button[name=btnG]')
   .end();
```

WebdriverIO **async** solution

```
webdriverio
    .remote(options)
    .init()
    .url('http://www.google.com')
    .getTitle().then(title=>{
        console.log('Title was: ' + title);
    })
    .end();
```

Nightwatch.js / WebdriverIO **async** solution

+ easy to read, understand, support such code;

+ doesn't need to use async specifics for simple and routine actions;

- library specific;

- needs special method to register async functions as chainable;

- non usual actions (debug) require promises usage;

MochaJS

+ very popular and excellent testing framework with large community and plugins and so on;

Pitfalls:

- async tests execution;
- uncaught exceptions;
- multiple **`after`**-hooks;

MochaJS **async** solution

```
describe('User', () => {
  describe('#save()', () =>{
    it('should save without error', done => {
      var user = new User('Luna');
      user.save(done);
    });
  });
});
```

MochaJS **async** solution

```
describe('User', () => {
  describe('#save()', () =>{
    it('should save without error', done => {
      var user = new User('Luna');
      user.save(done);
    });
  });
});
```

MochaJS async test without **done**

```
var sleep = timeout => {
    setTimeout(() => {
        console.log(`I was sleeping ${timeout} ms`);
    }, timeout);
};

describe("scope", () => {
    it("test #1", () => {
        sleep(1000);
    });
    it("test #2", () => {
        sleep(1000);
    });
});
```

MochaJS async test without **done**

```
var sleep = timeout => {
    setTimeout(() => {
        console.log(`I was sleeping ${timeout} ms`);
    }, timeout);
};

describe("scope", () => {
    it("test #1", () => {
        sleep(1000);
    });
    it("test #2", () => {
        sleep(1000);
    });
});
```

MochaJS async test without **done**

```
$ mocha mochaTest.js --no-exit


  scope
    √ test #1
    √ test #2


  2 passing (9ms)

I was sleeping 1000 ms
I was sleeping 1000 ms
```

MochaJS async test without **done**

```
$ mocha mochaTest.js --no-exit


  scope
    √ test #1
    √ test #2


  2 passing (9ms)

I was sleeping 1000 ms
I was sleeping 1000 ms
```

MochaJS async test with **done**

```
var sleep = (timeout, done) => {
    setTimeout(() => {
        console.log(`I was sleeping ${timeout} ms`);
        done();
    }, timeout);
};

describe("scope", () => {
    it("test #1", done => {
        sleep(1000, done);
    });
    it("test #2", done => {
        sleep(1000, done);
    });
});
```

MochaJS async test with **done**

```
var sleep = (timeout, done) => {
    setTimeout(() => {
        console.log(`I was sleeping ${timeout} ms`);
        done();
    }, timeout);
};

describe("scope", () => {
    it("test #1", done => {
        sleep(1000, done);
    });
    it("test #2", done => {
        sleep(1000, done);
    });
});
```

MochaJS async test with **done**

```
$ mocha mochaTest.js


  scope
I was sleeping 1000 ms
    √ test #1 (1005ms)
I was sleeping 1000 ms
    √ test #2 (1002ms)


  2 passing (2s)
```

MochaJS async test with **done**

+ there is a way to execute async tests

- needs to pass and process redundant argument not related with a function
- problems with multiple `done` calls
- looks ugly

MochaJS async test with **Promise**

```
var sleep = timeout => {
    return new Promise(resolve => {
        setTimeout(() => {
            console.log(`I was sleeping ${timeout} ms`);
            resolve();
        }, timeout);
    });
};

describe("scope", () => {
    it("test #1", done => {
        sleep(1000).then(done);
    });
    it("test #2", done => {
        sleep(1000).then(done);
    });
});
```

MochaJS async test with **Promise**

```
var sleep = timeout => {
    return new Promise(resolve => {
        setTimeout(() => {
            console.log(`I was sleeping ${timeout} ms`);
            resolve();
        }, timeout);
    });
};

describe("scope", () => {
    it("test #1", done => {
        sleep(1000).then(done);
    });
    it("test #2", done => {
        sleep(1000).then(done);
    });
});
```

MochaJS async test with **Promise**

+ Promises are nodejs specifics (not mochajs)

- needs to call **`done`** to notify mochajs **or**
- needs to return promise in a test and it looks ugly
- multiple promises usage involve **`then`** function which creates redundant code

MochaJS **async / await** test

```
var sleep = timeout => {
    return new Promise(resolve => {
        setTimeout(() => {
            console.log(`I was sleeping ${timeout} ms`);
            resolve();
        }, timeout);
    });
};

describe("scope", () => {
    it("test #1", async () => {
        await sleep(1000);
    });
    it("test #2", async () => {
        await sleep(1000);
    });
});
```

MochaJS **async / await** test

```
var sleep = timeout => {
    return new Promise(resolve => {
        setTimeout(() => {
            console.log(`I was sleeping ${timeout} ms`);
            resolve();
        }, timeout);
    });
};

describe("scope", () => {
    it("test #1", async () => {
        await sleep(1000);
    });
    it("test #2", async () => {
        await sleep(1000);
    });
});
```

MochaJS **async / await** test

+ Laconic and readable code
+ Native implementation in nodejs
+ Sequential step-by-step execution

- Often usage of **`async`** operator

MochaJS uncaught exceptions

```
var error = timeout => {
    setTimeout(() => {
        throw new Error("BOOM!!!")
    }, timeout);
};

describe("scope", () => {
    it("test #1", async () => {
        error(1000);
        await sleep(1000);
    });
    it("test #2", async () => await sleep(1000));
    it("test #3", async () => {
        error(1000);
        await sleep(1000);
    });
    it("test #4", async () => await sleep(1000));
    it("test #5", async () => await sleep(1000));
    it("test #6", async () => await sleep(1000));
});
```

MochaJS uncaught exceptions

```
var error = timeout => {
    setTimeout(() => {
        throw new Error("BOOM!!!")
    }, timeout);
};

describe("scope", () => {
    it("test #1", async () => {
        error(1000);
        await sleep(1000);
    });
    it("test #2", async () => await sleep(1000));
    it("test #3", async () => {
        error(1000);
        await sleep(1000);
    });
    it("test #4", async () => await sleep(1000));
    it("test #5", async () => await sleep(1000));
    it("test #6", async () => await sleep(1000));
});
```

MochaJS uncaught exceptions

```
var error = timeout => {
    setTimeout(() => {
        throw new Error("BOOM!!!")
    }, timeout);
};

describe("scope", () => {
    it("test #1", async () => {
        error(1000);
        await sleep(1000);
    });
    it("test #2", async () => await sleep(1000));
    it("test #3", async () => {
        error(1000);
        await sleep(1000);
    });
    it("test #4", async () => await sleep(1000));
    it("test #5", async () => await sleep(1000));
    it("test #6", async () => await sleep(1000));
});
```

MochaJS uncaught exceptions

```
  scope
    1) test #1
I was sleeping 1000 ms
    √ test #1 (1003ms)
    2) test #3
I was sleeping 1000 ms
I was sleeping 1000 ms
    √ test #4
    √ test #4
I was sleeping 1000 ms
I was sleeping 1000 ms
I was sleeping 1000 ms
    √ test #6 (1008ms)
    √ test #6 (1008ms)
    √ test #6 (1009ms)

  6 passing (3s)
  2 failing

  1) scope test #1:
     Uncaught Error: BOOM!!!
      at Timeout.setTimeout [as _onTimeout] (mochaTest.js:12:15)

  2) scope test #3:
     Uncaught Error: BOOM!!!
      at Timeout.setTimeout [as _onTimeout] (mochaTest.js:12:15)
```

MochaJS uncaught exceptions

```
  scope
    1) test #1
I was sleeping 1000 ms
    √ test #1 (1003ms)
    2) test #3
I was sleeping 1000 ms
I was sleeping 1000 ms
    √ test #4
    √ test #4
I was sleeping 1000 ms
I was sleeping 1000 ms
I was sleeping 1000 ms
    √ test #6 (1008ms)
    √ test #6 (1008ms)
    √ test #6 (1009ms)

  6 passing (3s)
  2 failing

  1) scope test #1:
     Uncaught Error: BOOM!!!
      at Timeout.setTimeout [as _onTimeout] (mochaTest.js:12:15)

  2) scope test #3:
     Uncaught Error: BOOM!!!
      at Timeout.setTimeout [as _onTimeout] (mochaTest.js:12:15)
```

MochaJS uncaught exceptions

```
  scope
    1) test #1
I was sleeping 1000 ms
    √ test #1 (1003ms)
    2) test #3
I was sleeping 1000 ms
I was sleeping 1000 ms
    √ test #4
    √ test #4
I was sleeping 1000 ms
I was sleeping 1000 ms
I was sleeping 1000 ms
    √ test #6 (1008ms)
    √ test #6 (1008ms)
    √ test #6 (1009ms)

  6 passing (3s)
  2 failing

  1) scope test #1:
     Uncaught Error: BOOM!!!
      at Timeout.setTimeout [as _onTimeout] (mochaTest.js:12:15)

  2) scope test #3:
     Uncaught Error: BOOM!!!
      at Timeout.setTimeout [as _onTimeout] (mochaTest.js:12:15)
```

MochaJS uncaught exceptions

```
var Mocha = require("mocha");

Mocha.Runner.prototype.uncaught = function (err) {
    logger.error("UNCAUGHT ERROR", err);
};
```

MochaJS multiple **after**

```
describe("scope", () => {
    it("test", () => {});
    after(() => {
        console.log("After #1");
        throw new Error("BOOM!");
    });
    after(() => {
        console.log("After #2");
        throw new Error("BOOM!");
    });
});
```

MochaJS multiple **after**

```
describe("scope", () => {
    it("test", () => {});
    after(() => {
        console.log("After #1");
        throw new Error("BOOM!");
    });
    after(() => {
        console.log("After #2");
        throw new Error("BOOM!");
    });
});
```

MochaJS multiple **after**

```
$ mocha mochaTest.js


  scope
    ✓ test
After #1
    1) "after all" hook


  1 passing (14ms)
  1 failing

  1) scope "after all" hook:
     Error: BOOM!
      at Context.after (mochaTest.js:8:15)
```

MochaJS multiple **after**

```
describe("scope", () => {
    before(async () => {
        await startProxy();
        await openBrowser();
    });
    it("test", async () => await openUrl("https://derivco.com"));

    after(async() => await closeBrowser());
    after(async() => await stopProxy());
});
```

X - failed
X - skipped

MochaJS multiple **after**

https://github.com/mochajs/mocha/blob/master/lib/runner.js#L329

GitHub, Inc. [US] | https://github.com/mochajs/mocha/blob/master/lib/runner.js

```
325            } else {
326              self.failHook(hook, err);
327
328              // stop executing hooks, notify callee of hook err
329              return fn(err);
330            }
```

```
if (!name.startsWith("after")) return fn(err);
```

https://gist.github.com/schipiga/f8176a368a91121a2316c97389f05556

Yet Another Framework Syndrome

STEPS

STEPS

- Any complex test may be divided to atomic steps
- Almost all tests contain steps which are (or may be) used in other tests
- Each step should be finished with verification that its result is correct
- Verification of step may be disabled if it needs for negative scenarios
- Steps are separated to **change**-steps, **get**-steps, **check**-steps

STEPS

- Any complex test may be divided to atomic steps
- Almost all tests contain steps which are (or may be) used in other tests
- Each step should be finished with verification that its result is correct
- Verification of step may be disabled if it needs for negative scenarios
- Steps are separated to **change**-steps, **get**-steps, **check**-steps

STEPS

- Any complex test may be divided to atomic steps
- Almost all tests contain steps which are (or may be) used in other tests
- Each step should be finished with verification that its result is correct
- Verification of step may be disabled if it needs for negative scenarios
- Steps are separated to **change**-steps, **get**-steps, **check**-steps

STEPS

- Any complex test may be divided to atomic steps
- Almost all tests contain steps which are (or may be) used in other tests
- Each step should be finished with verification that its result is correct
- Verification of step may be disabled if it needs for negative scenarios
- Steps are separated to **change**-steps, **get**-steps, **check**-steps

STEPS

- Any complex test may be divided to atomic steps
- Almost all tests contain steps which are (or may be) used in other tests
- Each step should be finished with verification that its result is correct
- Verification of step may be disabled if it needs for negative scenarios
- Steps are separated to **change**-steps, **get**-steps, **check**-steps

STEPS

- Any complex test may be divided to atomic steps
- Almost all tests contain steps which are (or may be) used in other tests
- Each step should be finished with verification that its result is correct
- Verification of step may be disabled if it needs for negative scenarios
- Steps are separated to **change**-steps, **get**-steps, **check**-steps

stepler

test openstack step by step

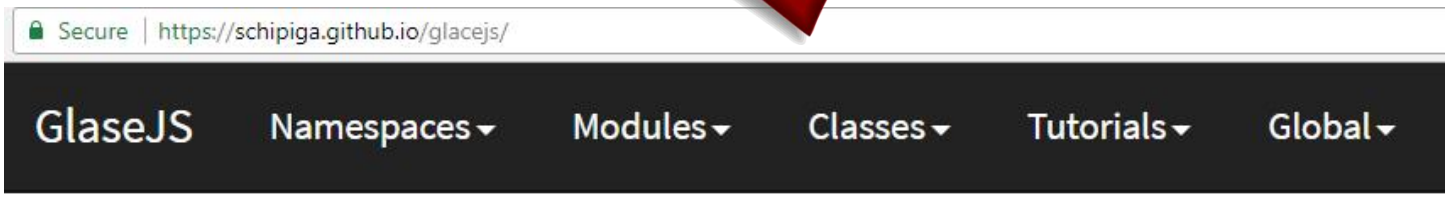
https://github.com/mirantis/stepler

https://stepler.readthedocs.io

GlaceJS (β)

https://github.com/schipiga/glacejs

GlaceJS ❤ JSDoc



https://schipiga.github.io/glacejs

GlaceJS

Oriented to complex functional scenarios

```
test("App statics should be gzipped", () => {
    before(async () => {
        await SS.startGlobalProxy();
        await SS.launchBrowser();
        await SS.measureResponses();
    });
    chunk(async () => {
        await SS.openApp();
        await SS.checkStaticsGzip([ ".js", ".css" ]);
    });
    after(async () => await SS.closeBrowser());
    after(async () => await SS.stopGlobalProxy());
});
```

GlaceJS

Multiple independent verifications in test

```
test("App index page is correct", () => {
    before(async () => {
        await SS.openApp();
    });
    chunk("check header", async () => {
        await SS.checkHeader();
    });
    chunk("check body", async () => {
        await SS.checkIndexBody();
    });
    chunk("check footer", async () => {
        await SS.checkFooter();
    });
});
```

GlaceJS

Parameterization

inside

```
CONF.languages = ["en", "ru", "ja", "ee"];
test("My multilingual test", () => {
    forEachLanguage(lang => {
        chunk(async () => {
            await SS.openApp(lang);
        });
    });
});
```

outside

```
["en", "ru", "ja", "ee"].forEach(lang => {
    test("My test for " + lang, () => {
        chunk(async () => {
            await SS.openApp(lang);
        });
    });
});
```

GlaceJS

Parameterization

inside

```
scope: Session Sun Oct 08 2017 12:26:58

  test: My multilingual test

    scope: for language "en"
      √ chunk

    scope: for language "ru"
      √ chunk

    scope: for language "ja"
      √ chunk

    scope: for language "ee"
      √ chunk

√ 1 passed test

----------------------------------------
Local report is C:\projects\glacejs\reports
```

outside

```
scope: Session Sun Oct 08 2017 12:34:53

  test: My test for en
    √ chunk

  test: My test for ru
    √ chunk

  test: My test for ja
    √ chunk

  test: My test for ee
    √ chunk

√ 4 passed tests

----------------------------------------
Local report is C:\projects\glacejs\reports
```

GlaceJS

Custom **retry** mechanism

```
glace /path/to/tests --retry 4

glace /path/to/tests --chunk-retry 4
```

GlaceJS

Multiple reports support
Easy to register own reporter

```
var reporter = require("glacejs").reporter;
reporter.register(myReporter);
```

https://github.com/schipiga/glacejs/tree/master/examples/ownReporter

GlaceJS

TestRail reporter included

14 Passed
58% set to Passed

0 Blocked
0% set to Blocked

0 Retest
0% set to Retest

7 Failed
29% set to Failed

58%
passed

3 / 24 untested
(13%).

GlaceJS

STEPS architecture
Page Object Pattern

```
var indexPage = new Page("index", "/",
    { searchField: "input#text",
      searchButton: "button[type='submit']"});

test("Page usage example", () => {
    before(() => SS.registerPages(indexPage));
    chunk(async () => {
        await SS.openPage(indexPage.name);
        await indexPage.searchField.setText("nodejs");
        await indexPage.searchButton.click();
    });
});
```

GlaceJS

Selenium standalone server included

```
before(async () => {
    if (CONF.installDrivers) {
        CONF.installDrivers = false;
        await SS.installSeleniumDrivers();
    };
    await SS.startSeleniumServer();
});

after(async () => {
    await SS.stopSeleniumServer();
});
```

GlaceJS

Images comparison mechanism

```
test("Compare equal screenshots", () => {
    chunk(async () => {
        await SS.openApp();
        var image1 = await SS.makeScreenshot();
        var image2 = await SS.makeScreenshot();
        await SS.checkImagesEquivalence(image1, image2);
    });
});
```

GlaceJS

Search image inside image with pixel-by-pixel comparison

```
test("Check image inclusion", () => {
    chunk(async () => {
        await SS.openApp();
        var searchImage = await SS.makeScreenshot(
            { element: "searchButton" });
        var fullImage = await SS.makeScreenshot();
        await SS.checkImageInclusion(fullImage, searchImage);
    });
});
```

GlaceJS

Video capture of executed tests

```
before(async () => {
    if (CONF.captureVideo) await SS.startVideo();
});

after(async () => {
    if (CONF.captureVideo) await SS.stopVideo();
});

after(() => {
    if (CONF.captureVideo && !CONF.forceVideo &&
            testCase.status === TestCase.PASSED) {
        fs.unlinkSync(SS.getVideo());
    };
});
```

GlaceJS

Support **Xvfb** with video capture

```
before(async () => {
    if (CONF.useXvfb) await SS.startXvfb();
});

after(async () => {
    if (CONF.useXvfb) await SS.stopXvfb();
});
```

GlaceJS

Simple proxy and transparent proxy for google chrome

```
before(async () => {
    if (CONF.useGlobalProxy) await SS.startGlobalProxy();
    if (CONF.useProxy) await SS.startProxy();
});

after(async () => {
    if (CONF.useProxy) await SS.stopProxy();
});
after(async () => {
    if (CONF.useGlobalProxy) await SS.stopGlobalProxy();
});
```

GlaceJS

Proxy middleware to cache and reply server responses

```
test("Cache responses", () => {
    chunk(async () => {
        SS.enableCache();
        await SS.openApp();  // cache population
        await SS.openApp();  // cache usage
        SS.disableCache();
        console.log("cache disabled");
        await SS.openApp();
    });
});
```

GlaceJS

Proxy middleware
to manage responses speed

```
test("Limit responses speed", () => {
    chunk(async () => {
        SS.limitProxySpeed(256 /* kb/s */);
        await SS.openApp();
        SS.unlimitProxySpeed();
        await SS.openApp();
    });
});
```

GlaceJS

Proxy middleware to gather responses data

```
test("Gather info about responses", () => {
    chunk(async () => {
        SS.measureResponses();
        await SS.openApp();
        console.log(SS.getResponsesData());
    });
});
```

GlaceJS

Easy to register own proxy middleware

```
var middleware = require("glacejs").middleware;
middleware.unshift(myMiddleware);
```

https://github.com/schipiga/glacejs/tree/master/examples/ownMiddleware


Q&A

Thank you!