Testing and validating Web Applications

Sylvain Fargier (BE-CEM-MRO)
Introduction
Introduction: forewords

- This presentation aims to:
  - Present a set of tools and methodology
  - Open discussion on test-driven-development for web-applications

- It is not:
  - A recommendation or guideline document
  - A document to debate on technologies
Introduction: General Overview
Introduction: Web-Application

- Web-Application
  - Dynamic web-page that interacts with a middleware
  - Similar to a regular application
- Web-Site
  - Static web-page
  - May interact with a middleware (ex: database)
  - Does not display "live" values
Back-end tests
Back-end tests: framework

- **Mocha.js** test framework
  - BDD/TDD oriented: `beforeEach()`, `afterEach()`, `describe()` and `it()`
- Basic, yet full of features
  - async/await, timeout, retry
  - code-coverage (nyc), Junit, browser-support...

```javascript
describe('KnexREST tests', function() {
  it('can list databases', async function() {
    /* ... */
  });
});
```

source: `mrest.js` (BE-CEM-MRO)
Back-end tests: assertions

- **Chai.js** assertion library
  - BDD/TDD oriented: expect, assert, should
  - `expect` syntax eases the code lecture

```javascript
/*...*/
expect(ret).to.deep.include({
  status: 200,
  body: [
    { id: 1, name: 'toto', value: 42 },
    { id: 2, name: 'titi', value: 43 },
    { id: 3, name: 'titi', value: 43 }
  ]});
```
source: mrest.js (BE-CEM-MRO)
Back-end tests: mocks & stubs

- **Sinon.js** spying, mocking and stubbing library
  - Can replace, spy or mock anything
  - Using sandbox to easily cleanup

```javascript
sinon.spy(DnsClient.prototype, 'query');
return q()
 .then(() => DicRpc.invoke(/...*/))
 .then(
   () => {
     throw 'should fail';
   },
   (err) => {
     expect(err).to.be.instanceOf(NotFound);
     expect(DnsClient.prototype.query.callCount)
       .to.equal(1, 'should query the DNS only once');
   }
)
 .finally(() => DnsClient.prototype.query.restore());
```

ex: checks that a deeply nested object is used only once, source: *dim.js* (BE-CEM-MRO)
Back-end tests: mocks & stubs

- **Sinon.js** is heavily artillery
  - Used really seldomly

- **Hand-written stubs** are more effective
  - Use SQLite when a DB is needed, ex: `mrest.js`
  - Implement both server and client part, ex: `dim.js, cmw-core.js`
  - Stub complete devices/middleware, ex: `ntof-stubs`
  - Those piece of codes are intensively re-used (for both back-end and front-end tests)!
Back-end tests: static analysis

- Linters: Eslint, JSHint
- TypeChecker: TypeScript
  - Using JSDoc comments (Doxygen like)
- Code-coverage: nyc (istanbul)

- GitLab-CI integration:
  - Test results and reports are processed by GitLab
  - Using an helper library (BE-CEM-MRO) to simplify the setup
  - Still exposing complete reports
Front-end tests
Front-end tests: Introduction

- Browser code (ECMAScript, CSS, HTML)
  - A library: only front-end code
  - Or an application: Node.js back-end + front-end
- Bundled using WebPack
- Transpiled with Babel
- Running in "moving" environment
  - Must be tested! tested again! and again!
Front-end tests: Introduction

- All back-end tools are re-used
  - Mocha.js test framework
  - Chai.js assertion library
  - Sinon.js spy and mocks library
  - Eslint, Jshint, JSDoc/TypeScript ...

- Some additional tools are needed to:
  - Run the code in browsers
  - Interact with DOM (as a regular user would do)
Front-end tests: test runner

- **Karma test runner**
  - Supports main browsers (Chrome, Firefox, Safari ...)
  - Headless mode
  - Integrates with WebPack, Mocha, nyc ...

asciinema record: `base-vue` (BE-CEM-MRO) tests in Headless Firefox (speed x2.5)
Front-end tests: test runner

- **Karma debugging**
  - Watch mode for TDD/BDD (fasten bundling)
  - Real browser connection (console, analysis ...)

Chrome running **ssvg-engine** (BE-CEM-MRO) test suite
Front-end tests: test runner

- **Karma.js** integration
  - Using an helper library (BE-CEM-MRO)
  - In GitLab-CI using helper scripts (BE-CEM-MRO)
  - For Web-Applications, front-end and back-end test results are merged (code-coverage, results, static-analysis ...)

```javascript
const { karmaConfig } = require('karma-mocha-webpack');
module.exports = function(karma) {
    karma.set(karmaConfig(karma));
};
```

*base-vue* (BE-CEM-MRO) configuration using *karma-mocha-webpack* (BE-CEM-MRO)
Front-end tests: user interactions

• **Vue-test-utils** unit testing utility library
  - Utilities to manipulate DOM and trigger events

• Some home made utilities
  - to wait for application states (ex: wait for animation to settle, or asynchronous work to finish)

```javascript
it('can use a dialog', async function() {
  wrapper = mount(BaseDialog, { propsData: { title: 'test dialog' } });

  let prom = wrapper.vm.request();
  let button = await waitForWrapper(wrapper,
    () => wrapper.findAll('button').filter((b) => b.text() === 'Ok'));
  button.trigger('click');
  expect(await prom).to.equal(true);
/*...*/
});
```

base-vue (BE-CEM-MRO) Dialog testing
Front-end tests: middleware interactions

- **karma-server-side**
  - Run code on the karma server side (Node.js)
  - Re-use stubs and back-end code we saw earlier (now it all makes sense 🎉)
  - Use Mocha beforeEach/afterEach to setup a custom middleware for each test

```javascript
1   await server.run(function() {
2     const { EACSStub } = serverRequire('ntof-stubs');
3     this.env.stub.eacs.state.setState(EACSStub.EACSState.State.RUNNING);
4   });
```

n_ToF operation application (op-control)
Front-end tests: continuous integration

- So far we've focused on unit-tests
  - Represents about 80% of bugs (totally arbitrary number)
- For the remaining bugs: continuous delivery
  - Deploy early, test often, give some attention to your work!
  - Be thankful to IT for the awesome infrastructure they provide!
Front-end tests: continuous integration

BE-CEM-MRO deployment workflow for web-apps