Starship Troopers

A different approach to UI5 tests execution

Presented by Arnaud Buchholz
Presentation made with Reveal.js

UI5con ON AIR

Arnaud Buchholz

- **Development Expert** @ SAP
- In the software industry for **23 years**

- **13 years** of experience with **JavaScript**
  - Front-End: 12 years
  - Node.js: 8 years
  - SapUI5: 6 years

- Passionate, curious & thinking **outside the box**
- French and almost **Canadian**

UI5con ON AIR

Agenda

- **Context**
- **Serving** an **UI5** application
- **Building** a test platform
- **Probing** the tests
- **Executing** the tests
- **Measuring** code coverage
- `ui5-test-runner`

About the presentation

- UI5 **ecosystem** is growing
- 🔗 **UI5 tooling** is the **recommended** solution for UI5 **development**

Innovation *sometimes* requires **ignoring** recommendations 😅

UI5con ON AIR

Context

«Back to 2018...»

UI5Con'18

A journey with **OPA**

recording
training-ui5con18-opa

UI5Con'19

Use **UI5** to **test** your **ODATA** service

recording
node-ui5

~~UI5Con'20~~

Testing UI5 applications

e-book

UI5con ON AIR

Quality focus

- Team is responsible of **10+** applications
- **Requirement** to have at least **80%** of coverage
- **Huge** tests suites *(up to 45 minutes)*

Pipeline issues

- CI pipeline uses the **Karma** runner
- OPA tests based on the **iFrame mode**
- **Huge** memory consumption
- Tests are abruptly **aborted** *(browser crash)*

Give me a 👍 if...

You **use** Karma

You **understand** how Karma **executes** the tests

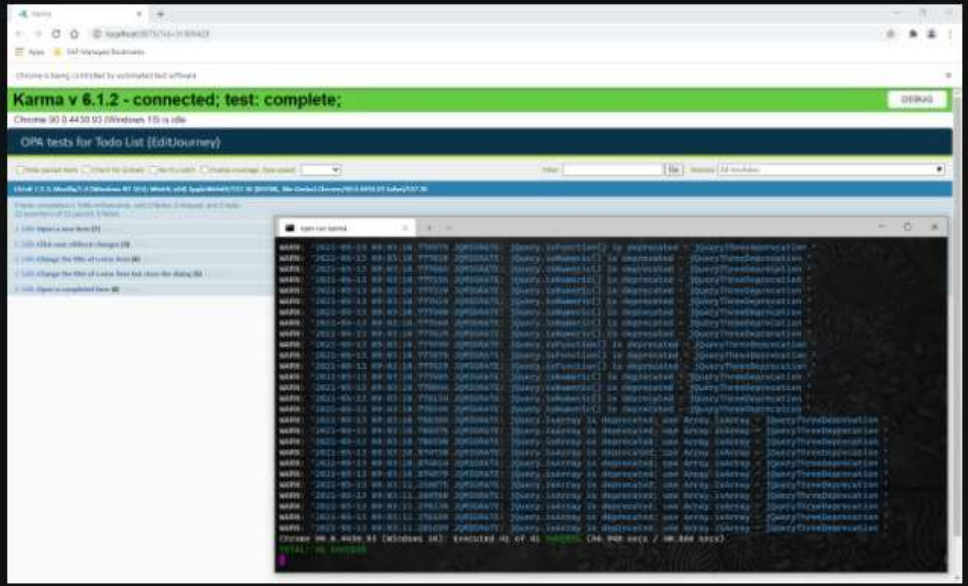
... I do **not** understand 😱

Karma execution model

- **All** tests in **one** window
- **One** iFrame to **run** them all
- **Sequential** execution
- **Coverage** information

Demo

Karma tests runner



npm run karma

Serving an UI5 application

Development setting

- UI5 **loaded** with `sap-ui-core.js`
  - No **specific** version
  - Path is **relative** to the project

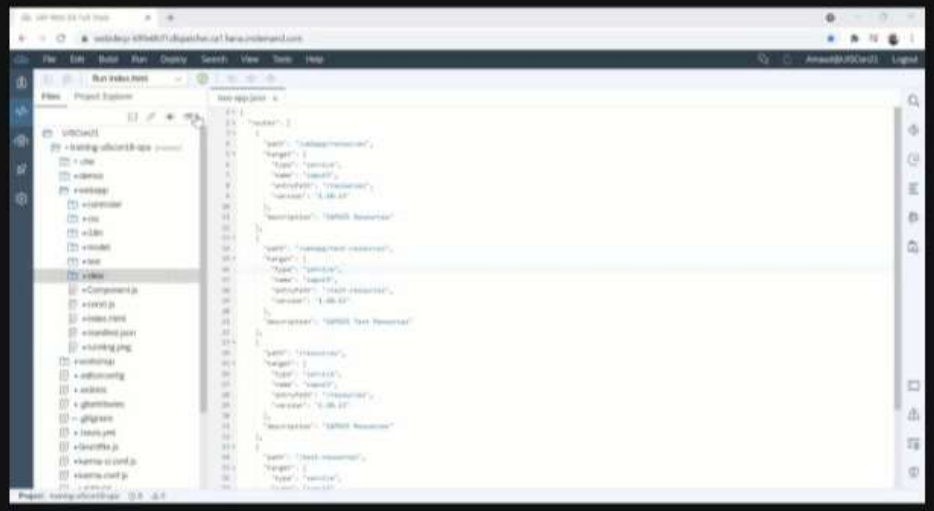
```
<script id="sap-ui-bootstrap" src="../resources/sap-ui-core.js"
  data-sap-ui-theme="sap_fiori_3"
  data-sap-ui-libs="sap.m"
  data-sap-ui-resourceroots='{"Demo": "./"}'
  data-sap-ui-onInit="module:Demo/index"
  data-sap-ui-async="true">
</script>
```

Example of UI5 bootstrap

- All **dependencies** are under `/resources/`
  or `/test-resources/`

Running the application (WebIDE)

- **Mapping** configured with `neo-app.json`
  - Enables **fast** version selection

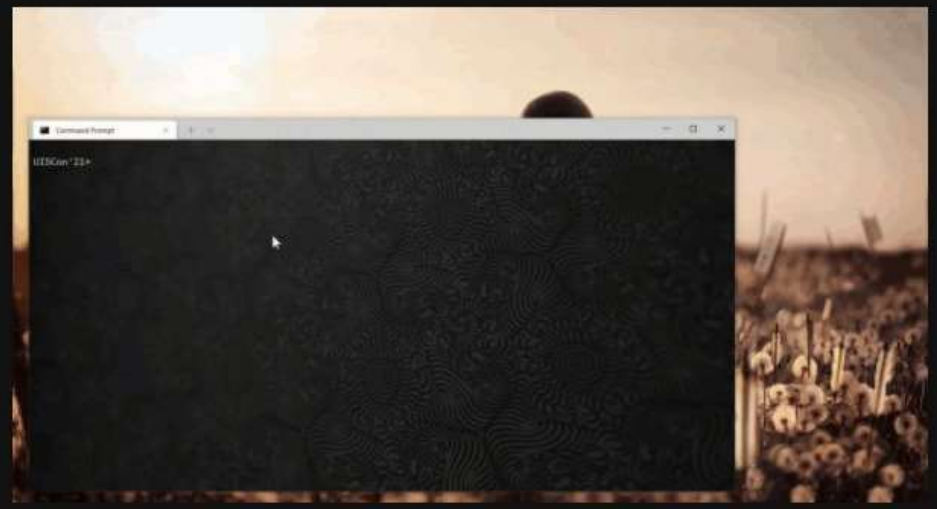


WebIDE

- **No** installation but **hard** to automate

Running the application (@ui5/cli)

- UI5 **packages** are installed **locally**
  - **Fastidious** version switch



ui5 serve -o

- **Heavy** but **automated**

Running the application (UI5 CDN)

- Load UI5 from a **Content Delivery Network**
  - Version selection requires to **change the bootstrap**
  - A **web server** is required to deliver the project files

```
<script id="sap-ui-bootstrap"
  src="https://openui5.hana.ondemand.com/1.87.0/resources/sap-ui-core.js"
  data-sap-ui-theme="sap_fiori_3"
  data-sap-ui-libs="sap.m"
  data-sap-ui-resourceroots='{"Demo": "./"}'
  data-sap-ui-onInit="module:Demo/index"
  data-sap-ui-async="true">
</script>
```

UI5 bootstrap from CDN

- **Lite** setup but **how** do we automate ?

Can we combine a **lightweight** setup
with a **fast & external** version selection ?

...In an **automatable** way ?

Introducing REserve

Small, configurable **and** reusable

```
{
  "port": 8080,
  "mappings": [{
    "match": "/(test-)?resources/(.*)",
    "url": "https://openui5.hana.ondemand.com/1.87.0/$1resources/$2"
  }, {
    "match": "^/(.*)",
    "file": "./webapp/$1"
  }]
}
```

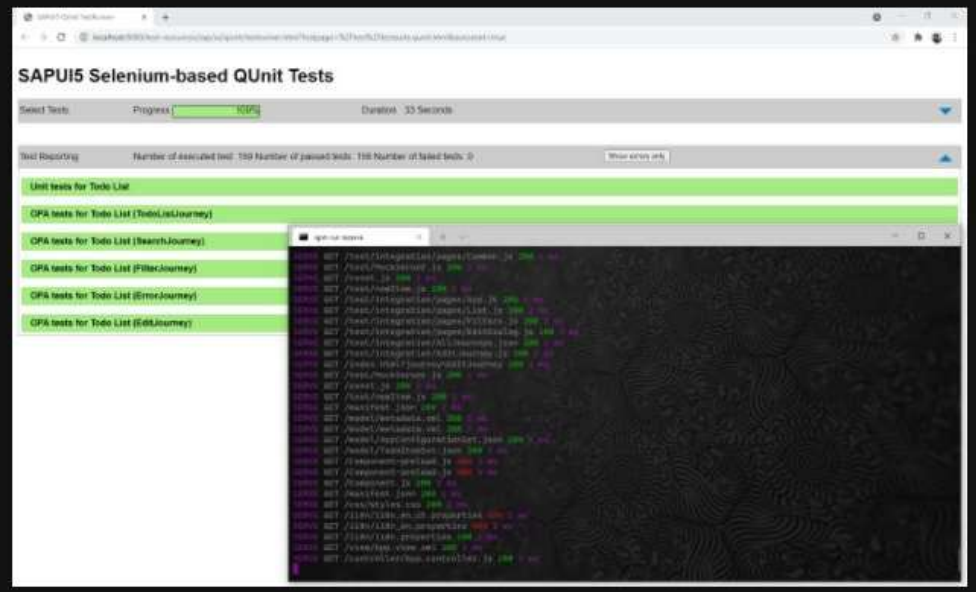
Example JSON configuration file to serve an UI5 application

reserve downloads 386/week install size 59.7 kB

UI5con ON AIR

Demo

Serving an UI5 application with REserve




```
npm run reserve
```

Test suite

UI5con ON AIR

Building a test platform

Defining the **options**

Every aspect of the platform is **configurable** through options :

- **Location** of the `webapp` folder
- **URL** of the `UI5` CDN
- **Serving** port
- ...

ui5-test-runner downloads 149/week

Serving the application

Done by embedding REserve

```
const { check, serve } = require('reserve')
check({
    port: job.port,
    mappings: [{
      match: '/(test-)?resources/(.*)', // UI5 mapping
      url: `${job.ui5}/$1`
    }, {
      match: /^\/(.*)/, // Project mapping
      file: join(job.webapp, '$1')
    }]
  })
    .then(configuration => serve(configuration))
}
```

Embedding REserve inside an application

Spawning a browser (1/3)

To **automate** the tests execution, the platform needs a way to **start** a browser and **stop** it when the tests are **done** *(or the **timeout** is reached)*.

The **browser instantiation** is deferred to a **configurable** script that is **spawned** by the platform.

Spawning a browser (2/3)

By **default**, a puppeteer script is provided

```
const puppeteer = require('puppeteer')
let browser
process.on('message', async message => {
  if (message.command === 'stop') { // End signal
    await browser.close()
    process.exit(0)
  }
})
async function main () {
  browser = await puppeteer.launch({
    headless: true,
    args: [process.argv[2] /* Test page to open */, /* ... */]
  })
}
main()
```

Browser instantiation script

Spawning a browser (3/3)

Tests were conducted to validate that **multiple and concurrent instances** do not **interfere** with each other :

- Timeouts
- Local storage
- Cache
- ...

Custom **endpoints**

During the tests execution, the platform needs to **receive feedback** from the browsers.

For instance : to know when a test ends

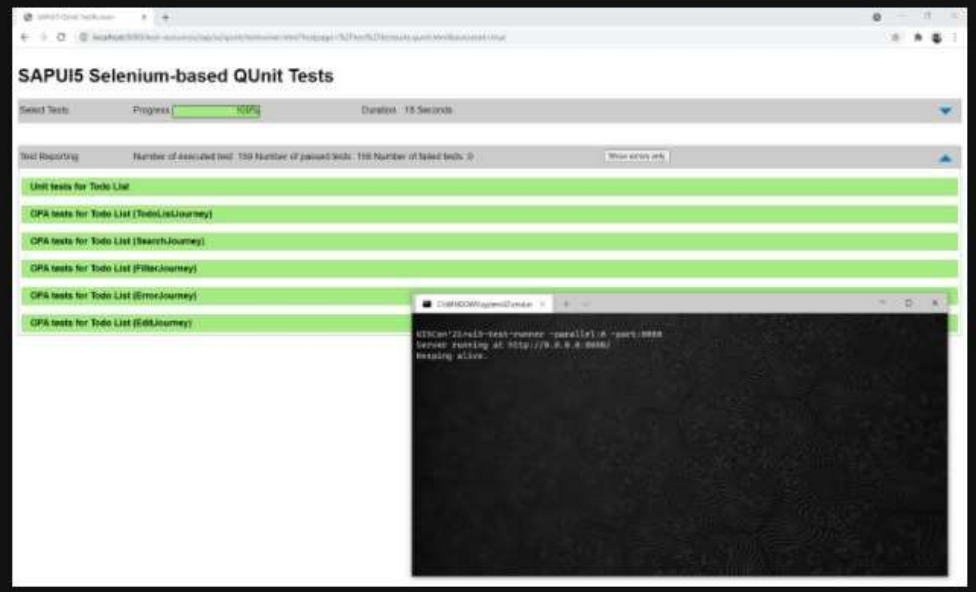
Dedicated **endpoints** are created.

(base URL : /_/)

The endpoints leverage the `referer` header to **identify** which test *(test page URL)* initiated the **request**.

Demo

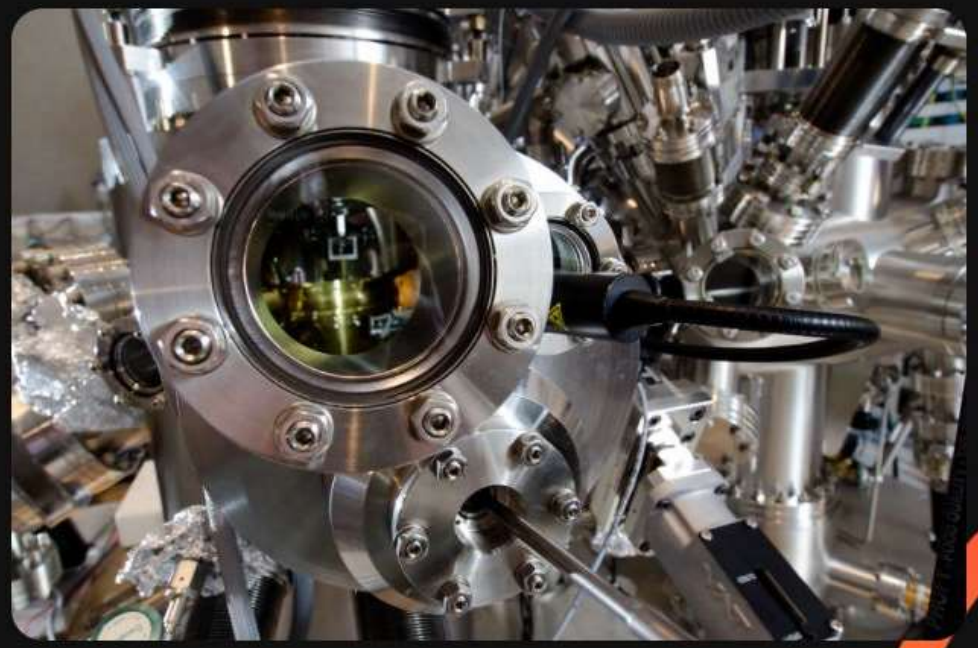
Serving an UI5 application with the platform




```
ui5-test-runner -parallel:0 -port:8080
```

Test suite

Probing the tests



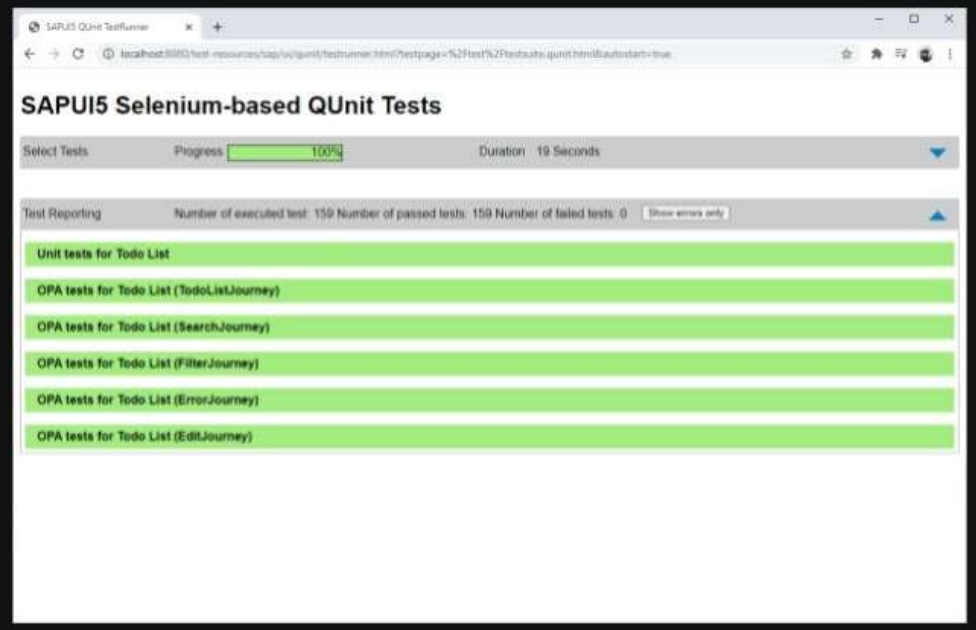
Declaring the tests (1/3)

The **test suite** page
declares every unit and OPA tests
contained in the project.

`webapp/test/testsuite.qunit.html`

Declaring the tests (2/3)

When opening the page, a **redirection** occurs and a runner **executes** the declared tests.



UI5 test runner

Declaring the tests (3/3)

```
<html>
  <head>
    <script src="../resources/sap/ui/qunit/qunit-redirect.js"></script>
    <script>
    function suite() {
      var oSuite = new parent.jsUnitTestSuite(),
        iLastSep = location.pathname.lastIndexOf("/") + 1,
        sContextPath = location.pathname.substring(0, iLastSep);
      oSuite.addTestPage(sContextPath + "unit/unitTests.qunit.html");
      oSuite.addTestPage(sContextPath + "integration/opaTests.qunit.html");
      return oSuite;
    }
    </script>
  </head>
</html>
```

Tests declaration in the `testsuite.qunit.html`

Test suite extraction (1/2)

By **substituting** `qunit-redirect.js`
with a **custom script** and
opening `testsuite.qunit.html`,
the platform **captures** the **list** of test pages.

Test suite extraction (2/2)

```
const pages = []
function jsUnitTestSuite () {}
jsUnitTestSuite.prototype.addTestPage = function (url) {
  pages.push(url)
}
window.jsUnitTestSuite = jsUnitTestSuite // Expose
window.addEventListener('load', function () {
  suite() // Trigger
  const xhr = new XMLHttpRequest()
  xhr.open('POST', '/_/addTestPages')
  xhr.send(JSON.stringify(pages))
})
```

Custom `qunit-redirect.js` used to extract test pages

Demo

Substituting `qunit-redirect.js`

```
C:\WINDOWS\system32\cmd.e

UI5Con'21>ui5-test-runner -parallel:-1 -port:8888
Server running at http://0.0.0.0:8888/
nyc instrument C:\Users\i850991\git\training-ui5con18-opa\webapp C:\Users\i850991\git\training-ui5con
18-opa\.nyc_output\instrumented --nycrc-path C:\Users\i850991\git\training-ui5con18-opa\.nyc_output\s
ettings\nyc.json
>> /test/testsuite.qunit.html
/test/testsuite.qunit.html [
  '/test/unit/unitTests.qunit.html',
  '/test/integration/opaTests.qunit.html?journey=TodoListJourney',
  '/test/integration/opaTests.qunit.html?journey=SearchJourney',
  '/test/integration/opaTests.qunit.html?journey=FilterJourney',
  '/test/integration/opaTests.qunit.html?journey=ErrorJourney',
  '/test/integration/opaTests.qunit.html?journey=EditJourney'
]
<< /test/testsuite.qunit.html
```

`ui5-test-runner -parallel:-1 -port:8080`

original `qunit-redirect.js`

substituted `qunit-redirect.js`

UI5con ON AIR

Executing the tests

QUnit hooks

The OPA framework is built **on top of** `QUnit`.

`QUnit` exposes **hooks** to **monitor** the tests :

- `QUnit.begin`
- `QUnit.testDone`
- `QUnit.done`

They provide information about **progress** and **status**.

Injecting the hooks

When the test page **loads** the `QUnit` module, the platform **injects** code to **leverage the hooks** and trigger specific **endpoints**.

qunit.js
qunit-2.js

Execution **queue** (1/2)

After **probing** the tests, the platform knows the **list** of test page URL.

Then, it instantiates concurrent browsers to run **each** page **individually**.

The number of concurrent browsers is given by the option `parallel`.

The test page is **stopped** when the test is done (`QUnit.testDone`) or when the **timeout** is reached.

Execution queue (2/2)

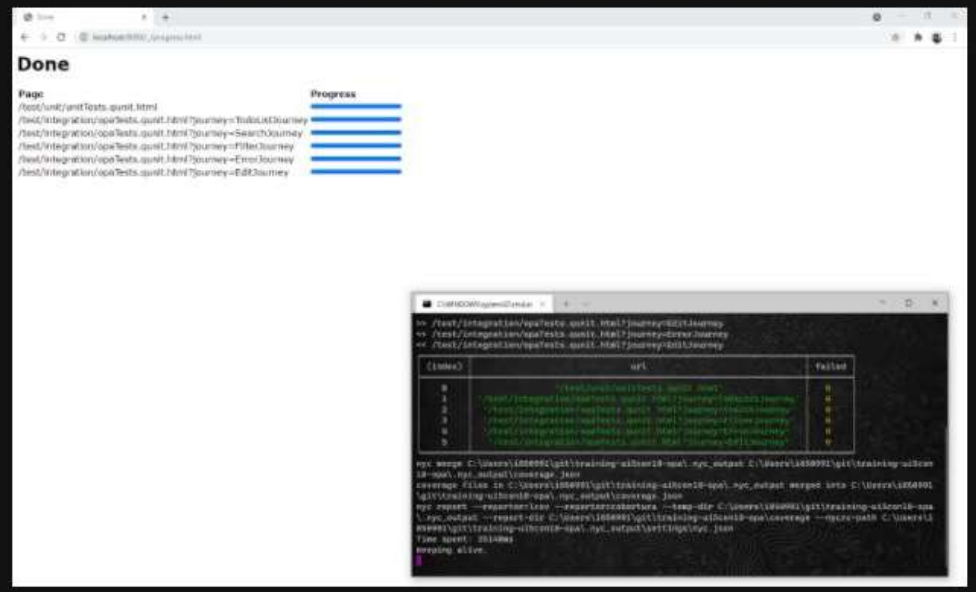
```
for (let i = 0; i < job.parallel; ++i) runTestPage()

async function runTestPage () {
  const { length } = job.testPageUrls
  if (job.testPagesCompleted === length) {
    return generateReport() // Last test completed
  }
  if (job.testPagesStarted === length) {
    return // No more tests to run
  }
  const url = job.testPageUrls[job.testPagesStarted++]
  await start(url) // Resolved when the browser is stopped (or timed out)
  ++job.testPagesCompleted
  runTestPage()
}
```

Execution queue simplified implementation

Demo

Executing the tests in parallel




```
ui5-test-runner -parallel:2 -port:8080 -cache:.ui5 -keepAlive:true
```

🔗 Progress 🔗 report

UI5con ON AIR

Measuring code coverage

Understanding code coverage

The process requires three steps :

- **Instrumentation** of the code
- **Evaluation** of the code
- **Extraction** and **consolidation** of measurement

NYC

`nyc` is a command line **wrapper** for `Istanbul`, a javascript code coverage tool.

The platform leverage `nyc` by **spawning** commands and **waiting** for their termination.

Instrumenting sources

The instrumentation step is triggered **before** executing the tests. By default, the test files are **excluded** from the coverage report.

NYC **aggregates** the collected coverage information at the **window level**. Because of the **IFrame** usage in OPA, this is changed to the **top window** by the platform.

Injecting instrumented sources

When an instrumented file **exists** for a source file, it is **substituted**.

```
/* Reserve mappings : */ [{
  match: /^\/(.*\.js)$/,
  file: join(instrumentedSourceDir, '$1'),
  'ignore-if-not-found': true
}, {
  match: /^\/(.*)/,
  file: join(job.webapp, '$1')
}]
```

Injecting instrumented files if they exist

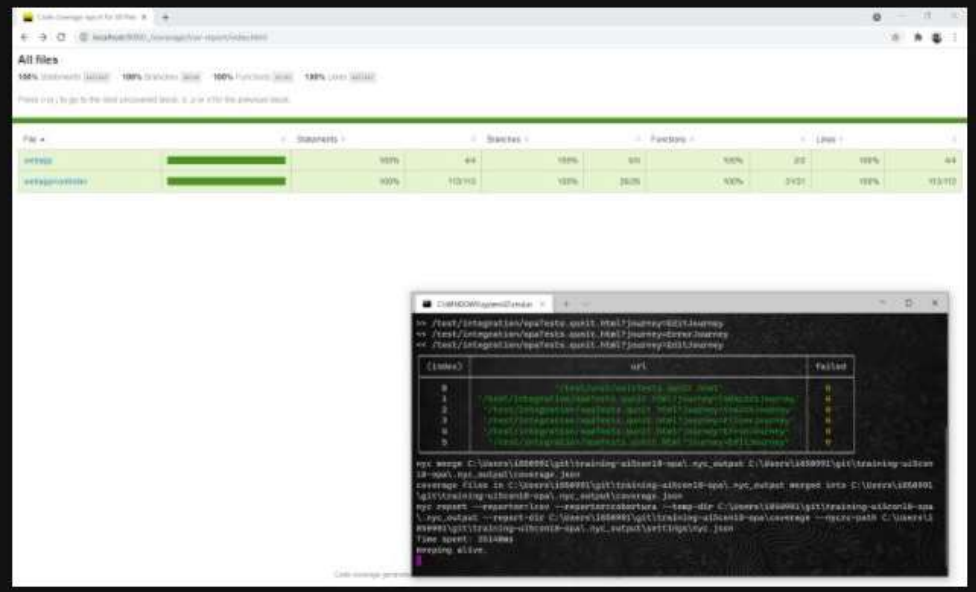
Consolidating code coverage

Each test page is executed **separately** : corresponding coverage data is **saved** at the end of each test page.

When all the tests are **completed**, NYC is used to **merge** the coverage information and **generate** the coverage report.

Demo

Coverage measurement




```
ui5-test-runner -parallel:2 -port:8080 -cache:.ui5 -keepAlive
```

Coverage report

ui5-test-runner

Karma execution model

- **Generic** but **adapted** to UI5
- **All** tests in **one** window
  One iFrame to **run** them all
- **Sequential** execution
- **Coverage** information

- **Obscure** implementation

karma downloads 2M/week install size 8.58 MB

ui5-test-runner execution model

- **Specific** to UI5
- **Each** test page in a **distinct** window
- **Parallel** execution
- **Coverage** information

- **Crystal clear** implementation

  ui5-test-runner downloads 149/week

Tips & Tricks

- To benefit from **parallelization**, **split** the OPA tests by **journeys**
  (see 🔗 `training-ui5Con18-opa`)
- **Cache** UI5 locally to **speed up** page execution

Ideas to improve the platform

- **Smarter** watch
  (test only what is impacted by the change)
- Test **grids**
  (using **Selenium** integration)

Thank you !

Arnaud Buchholz
Development Expert
npm ui5-test-runner

UI5con ON AIR