Kat - Casper's Best Friend

Kat is a data driven interpreter on top of CasperJS. Rather than having to fire up a new project that includes Casper and PhantomJS each time, Kat is driven from a template and data set file. Separating data from implementation.

Dependancies:

The following libraries and projects are utilized in Kat.

Coffee-Script - JavaScript Transcompiler
Lo-Dash - Functional programming support for JavaScript
Casper - A navigation scripting & testing utility for PhantomJS
Phantom - Scriptable Headless WebKit

Usage

Kat takes a JSON/YAML template file specified as an argument from the command line. The type of file is detected through the extension and does not need to be specified.

```
kat --template="./data/templates/test.json"
```

Currently the program will default to using four directories under *./data*.

- *./data/templates/* - primary configuration files to be used for testing
- *./data/logs/* - the result of those test
- *./data/screenshots/* - screen shots from the tests
- *./data/sets/* - screen shots from the tests

Config/Data File Formatting

test - base object with options that will be use for the entire test.
name - name of the actual test that will be used for log the files.
viewport - size of the virtual browser in pixels.
step - a sequence of steps that comprises the actual meat of the test.

Fully Qualified Data Format

Fully qualified objects have the parameter names called out as individual key/value pairs, generally starting with **name**.

```
{
    "name": "Amazon Test 1",
    "viewport" : { "width": 1920, "height": 3840 },
    "step": [
        {"action": "open", "link": "http://www.amazon.com"},
        {"action": "find", "select": "#twotabsearchtextbox" },
        {"action": "type", "select": "#twotabsearchtextbox", "text": "functional javascript"},
        {"action": "click", "select": "input[class='nav-submit-input']"},
        {"action": "visible", "select": "h1#breadCrumb"},
        {"action": "capture", "select": "body"}]
}
```

Short-hand for Steps

The above steps can be rewritten in shorthand as such.

Example of Short-hand

```
    "step": [{
        "open": "http://www.amazon.com",
        "find": "#twotabsearchtextbox",
        "type": ["#twotabsearchtextbox", "functional javascript"],
        "click": "input[class='nav-submit-input']",
        "visible": "h1#breadCrumb",
        "capture": "body"
    }]
```

Actions

Each step is a list of objects and the properties must be distinct. This is important to remember for short-hand step syntax, because this disallows you from calling out multiple of the same action in a single step.

- **fill: [form, map]** - fill the form with the selector:value object pairs.
- **click: [select]** - click the particular selector element
- **open: [link]** - open the link with authentication if provided.
- **capture:[select, height, width]** - take a screen shot of particular selector.
- **find: [select, timeout]** - waits until the element exists in the page
- **visible:[select, timeout]** - waits until the element is visible on the page.
- **text: [value, timeout]** - waits until the text is present in the page.
- **uri: [regex, timeout]** - waits until the page is on the url to match the specified regular expression.
- **wait: [timeout]** - waits this many milliseconds.
- **type: [select, text, modifiers]** - input text using key strokes with the text specified.
- **echo: [message]** - print message out to console and in the log.

The brackets can be omitted for actions with single arguments

Parameters

- ***form*** - a CSS Selector for a particular form. (`"#aspnetForm"`)
- ***url*** - a url.(http://www.amazon.com)
- ***map*** - key value pair of selectors and values.
  ({ `"#street": " 12365 Riata Trace Pkwy", "#city": "Austin", "#state": "Texas", "#zip": "78727"` })
- ***message*** - text to print to console and log.
- ***modifiers*** - alt, control, shift, etc. (`"ctrl+alt+shift"`)
- ***regex*** - a regular expression. (`"http://www\.google\.com"`)
- ***select*** - a CSS Selectors. (`"input[class='something']"`)
- ***text*** - text to be typed key by key, useful for strict validators.
- ***timeout*** - milliseconds to wait.
- ***value*** - literal value or text to match.

Data Sets

A data set a is a list of maps that correspond to information that the test can use during run-time. Data sets can be in JSON, YAML, Tab delimited, or Comma delimited. Comma, and Tab separated value files support both horizontal and vertical configurations.

`kat --template="./data/templates/test-template.json --set="./data/sets/test-set.tsv`

Data Sets drive test execution. All tests specified within the configuration file will be ran for every map in the set. The set below will echo `echoThis` twice.

Data Set file

```
//JSON configuration - *.json
[
    { "echoThis" : "Output this during test", "unusedKey": "This wont be used"},
    { "echoThis" : "Also Output this during test", "unusedKey": "neith will this"}
]

//Vertical configuration - *.csv, *.tsv
echoThis,Output this during test,Also Output this during test.
unusedKey,This wont be used, neither will this.

or

//Horizontal configration - *.comma, *.tab
echoThis,unusedKey
Output this during test,This wont be used
Also Output this during test,neith will this
```

Configuration File

```
{
    "name": "Data Set Test"
    "step": [
        {"echo": { "get" : "echoThis"}}
    ]
```

```
}

or

```javascript
{
    "name": "Data Set Test"
    "step": [
        {"echo": "$echoThis"}
    ]
}
```

Notes and Copyright

> Kat is a work in progress. If you have any questions, problems, or ideas you are more than welcome to contact me.
```