Lesson 4

Redis - Unit Test

https://www.nimbella.com

Plan

- Redis
  - local setup
  - searches
  - types
- Unit Test
  - Jest
  - Promises
  - Snapshot

Prerequisites

- Local Redis
  - Running in Docker
- Nimbella SDK
  - To connect to redis (either in cloud or local)
  - `@nimbella/sdk` use to locate redis:
    - `__NIM_REDIS_IP`
    - `__NIM_REDIS_PASSWORD`

Start and connect to local Redis

```
# setup a local redis
docker kill redis
docker run --name redis --rm -p 6379:6379 -d redis --requirepass password
# setup nimbella sdk
mkdir -p support/packages
cd support
npm install @nimbella/sdk
# connect the nimbella SDK to the local redis
export __NIM_REDIS_IP=127.0.0.1
export __NIM_REDIS_PASSWORD=password
# connecting to local redis
node --experimental-repl-await
let rd = require("@nimbella/sdk").redis()
rd.ready
```

Redis Data Types

- Keys: `set`, `get`, `del`, `keys`,...
  - default: String
- List: `rpush`, `lpush`, `lpop`, `linsert`, `llen`, `lrange`...
- Hashes: `hget`, `hset`, `hdel`, `hgetall`,...
- Set: `sadd`, `smembers`, `sismember`,...
- Others (specialized)

Redis key get/set/del

```
// Redis  get/set/del/keys
await rd.getAsync("k")
# null
rd.set("k","v")
await rd.getAsync("k")
# v
await rd.keysAsync("*")
# ['k']
rd.del("k")
await rd.getAsync("k")
# null
```

Redis Lists

```
// Redis lists
rd.lpush("l", 1)
await rd.llenAsync("l")
# 1
rd.rpush("l", 2)
await rd.lrangeAsync("l", 0,-1)
# [ '1', '2' ]
rd.linsert("l",  "AFTER", 1, 3)
await rd.lrangeAsync("l", 0,-1)
# [ '1', '3', '2' ]
rd.lpop("l")
await rd.lrangeAsync("l", 0,-1)
# [ '3', '2' ]
```

Redis Hashes

```
// Redis hashes
rd.hset("h", "a", 1)
await rd.hgetAsync("h", "a")
# '1'
await rd.hgetAsync("h", "b")
# null
rd.hset("h", "b", 2)
await rd.hgetallAsync("h")
# { a: '1', b: '2' }
rd.hdel('h', 'b')
await rd.hgetallAsync("h")
# { a: '1' }
rd.del('h')
await rd.hgetallAsync("h")
# null
```

Sets

```
// Redis sets
rd.sadd("s", "1")
rd.sadd("s", "3")
await rd.smembersAsync("s")
# ['1', '3']
rd.sadd("s", "2")
rd.sadd("s", "3")
await rd.smembersAsync("s")
#  '1', '2', '3' ]
await rd.sismemberAsync("s", "1")
# 1
await rd.sismemberAsync("s", "4")
# 0
```

Importing some values

```
## Importing some values
# url to faas wars robot list
URL=https://apigcp.nimbella.io/api/v1/web/nimbots/rumble/public
# getting the list
curl $URL >nimbots.json
cat nimbots.json
# extracting values
jq -r '.[]|" \(.name) \(.url)"'  <nimbots.json >nimbots.txt
cat nimbots.txt
# storing in redis online
nim kv clean
xargs -L1 nim kv set <nimbots.txt
nim kv list
```

Scan

```
let rd = require("@nimbella/sdk").redis()
return rd.scanAsync(cursor,
            "MATCH", match,
            "COUNT", count)
        .then(x => ({ result: x }))
```

- `cursor` is current page
- `match` is a "pattern"
- `count` is the page size

Patterns: scan, keys, hscan,...

- `h?llo` matches hello, hallo and hxllo
- `h*llo` matches hllo and heeeello
- `h[ae]llo` matches hello and hallo, but not hillo
- `h[^e]llo` matches hallo, hbllo, ... but not hello
- `h[a-b]llo` matches hallo and hbllo

scan.js

```
exports.main = function (args) {
    let rd = require("@nimbella/sdk").redis()
    let match = args.match || "*"
    let count = parseInt(args.count) || 10
    let cursor = parseInt(args.cursor) || 0
    return rd.scanAsync(cursor, "MATCH", match, "COUNT", count)
        .then(x => ({ result: x }))
}
```

Testing Scan

```
# Testing Scan
cd support
mkdir packages/support
cp ../src/scan.js packages/support/scan.js
nim project deploy .
nim action invoke support/scan
nim action invoke support/scan  -p cursor 20
nim action invoke support/scan  -p cursor 50
nim action invoke support/scan  -p cursor 110
nim action invoke support/scan  -p cursor 81
nim action invoke support/scan  -p cursor 121
nim action invoke support/scan  -p cursor 75
nim action invoke support/scan  -p cursor 63
```

Iterate scanning

```
//...
    return collect(rd, [], 0, match, count)
        .then(x => ({ result: x }))
//...

function collect(rd, res, cursor, match, count) {
    return rd.scanAsync(cursor, "MATCH", match, "COUNT", count)
    .then(r => {
        res = res.concat(r[1])
        if(r[0] == 0)
          return Promise.resolve(res)
        return collect(rd, res, r[0], match, count)
    })
}
```

Scan All

```
# deploy the scanall action
cp ../src/scanall.js packages/support
nim project deploy .
nim action invoke support/scanall
nim action invoke support/scanall -p "match" "h*"
nim action invoke support/scanall -p "match" "m*"
```

Setup Test Prerequisites

```
# Init Jest
npm install -g jest
cd support
touch jest.config.js
```

Unit Test Guinea Pig: `hello/index.js`

```
function main(args) {
    let name = args.name ||  "world"
    return {
        "hello": name
    }
}
// note
exports.main = main
```

Writing an unit test

- file: `index.test.js`
  - ignore it! `.ignore` with `*.test.js`
- load and define a test:

```
const main = require("./index.js").main
test("world", () => {
```

- expect the result and check with a matcher

```
    expect(main({}))
        .toEqual({ hello: 'world' })  })
```

hello/index.test.js

```
const main = require("./index.js").main

test("world", () => {
    expect(main({}))
        .toEqual({ hello: 'world' })
})

test("name", () => {
    expect(main({"name": "Mike"}))
        .toEqual({ hello: 'Mike' })
})
```

```
$
$ jest hello
 PASS  packages/support/hello/index.test.js
   ✓ world (2 ms)
   ✓ name

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.831 s, estimated 1 s
Ran all test suites matching /hello/i.
$
```

Running Tests with jest

```
# running tests with jest
mkdir -p packages/support/hello
cp ../src/.ignore packages/support/hello/.ignore
cp ../src/hello.js packages/support/hello/index.js
cp ../src/hello.test.js packages/support/hello/index.test.js
jest
# deploy and test "live"
nim project deploy .
nim action invoke support/hello
nim action invoke support/hello -p name Mike
```

Testing with a promise

```
const get = require("./get.js").main
```

- Wrong! Do not `expect` for a promise

```
expect( get({"key":"hello"}) ).toEqual({"result": null})
```

- Correct: return a promoise from a test
  - jest will wait for promises to resolve

```
get({"key":"hello"}).then(x => expect(x).toEqual({"result": null}))
```

get.js

```
exports.main = function (args) {
    let rd = require("@nimbella/sdk").redis()
     return rd.getAsync(args.key).then(x => {
         rd.end(true) // required to avoid test to be stuck
         return {result:x}
     })
}
```

similarly set.js and get.js (not shown)

get.test.js

```
// get.test.js
const get = require("./support/get.js").main

test("get", () =>
    get({"key":"hello"}).then(x => expect(x).toEqual({"result": null})
))
```

Testing with Jest get

```
# simple get test
mkdir -p packages/support
cp ../src/get.js packages/support/get.js
# put tests for single file actions outside of folders
cp ../src/get.test.js packages/get.test.js
jest /get
```

Chained tests

```
// chained tests
const get = require("./get.js").main
const set = require("./set.js").main

test("setget", () =>
    set({ "key": "hello", "value": "world" })
        .then(() => get({ "key": "hello" }))
        .then(x =>
            expect(x).toEqual({ "result": "world" })
        )
)
```

Test with Side Effect

```
# now a test succeed and another fails
cp ../src/set.js packages/support/set.js
cp ../src/setget1.test.js packages/setget.test.js
# success
jest /setget
# fail
jest /get
```

Fixtures

- The `get` test needs a "fixture" to prepare the environment

```
const get = require("./get.js").main
const del = require("./del.js").main

beforeAll( () => del({"key":"hello"}))

test("get", () =>
    get({"key":"hello"}).then(x =>
     expect(x).toEqual({"result": null})
))
```

Fixture fixes all tests

```
# Fixture fixes all tests
cp ../src/del.js packages/support/del.js
cp ../src/get.test2.js packages/get.test.js
# checking
jest /get
# checking all
jest
```

Deployment and Test Online

```
## Deployment and Test Online
nim project deploy .
nim action invoke support/get -p key hello
nim action invoke support/set -p key hello -p value world
nim action invoke support/get -p key hello
nim action invoke support/del -p key hello
nim action invoke support/get -p key hello
# can be automated too
```

Snapshot testing...

- Super easy testing
  - for lazy people (like me)

```
test("get", () =>
    get({ "key": "hello" })
        .then(x => expect(x).toMatchSnapshot()))
```

- use `toMatchSnapshot()`
  - execute once & check results
  - tests will check it keeps working

Sample test with snapshot

```
const set = require("./support/set.js").main
const get = require("./support/get.js").main
const del = require("./support/del.js").main
beforeAll( () => del({"key":"hello"}))

test("get", () => get({ "key": "hello" })
    .then(x => expect(x).toMatchSnapshot()))
test("set", () => set({ "key": "hello", "value": "world" })
    .then(x => expect(x).toMatchSnapshot()))
test("get2", () => get({ "key": "hello" })
    .then(x => expect(x).toMatchSnapshot()))
test("del", () => del({ "key": "hello" })
    .then(x => expect(x).toMatchSnapshot()))
test("get3", () => get({ "key": "hello" })
    .then(x => expect(x).toMatchSnapshot()))
```

Testing with snapshot

```
# get the tes
cp ../src/setgetdel.test.js packages/setgetdel.test.js
# run the test
jest
cat packages/__snapshots__/setgetdel.test.js.snap
# run it again
jest
# change something
cp ../src/get.js packages/support/set.js
jest
# fix
cp ../src/set.js packages/support/set.js
jest
```

Certification Exercise

Exercise 2 was "adding edit" to the simple CRUD example.

You have to add unit tests to the backend actions.