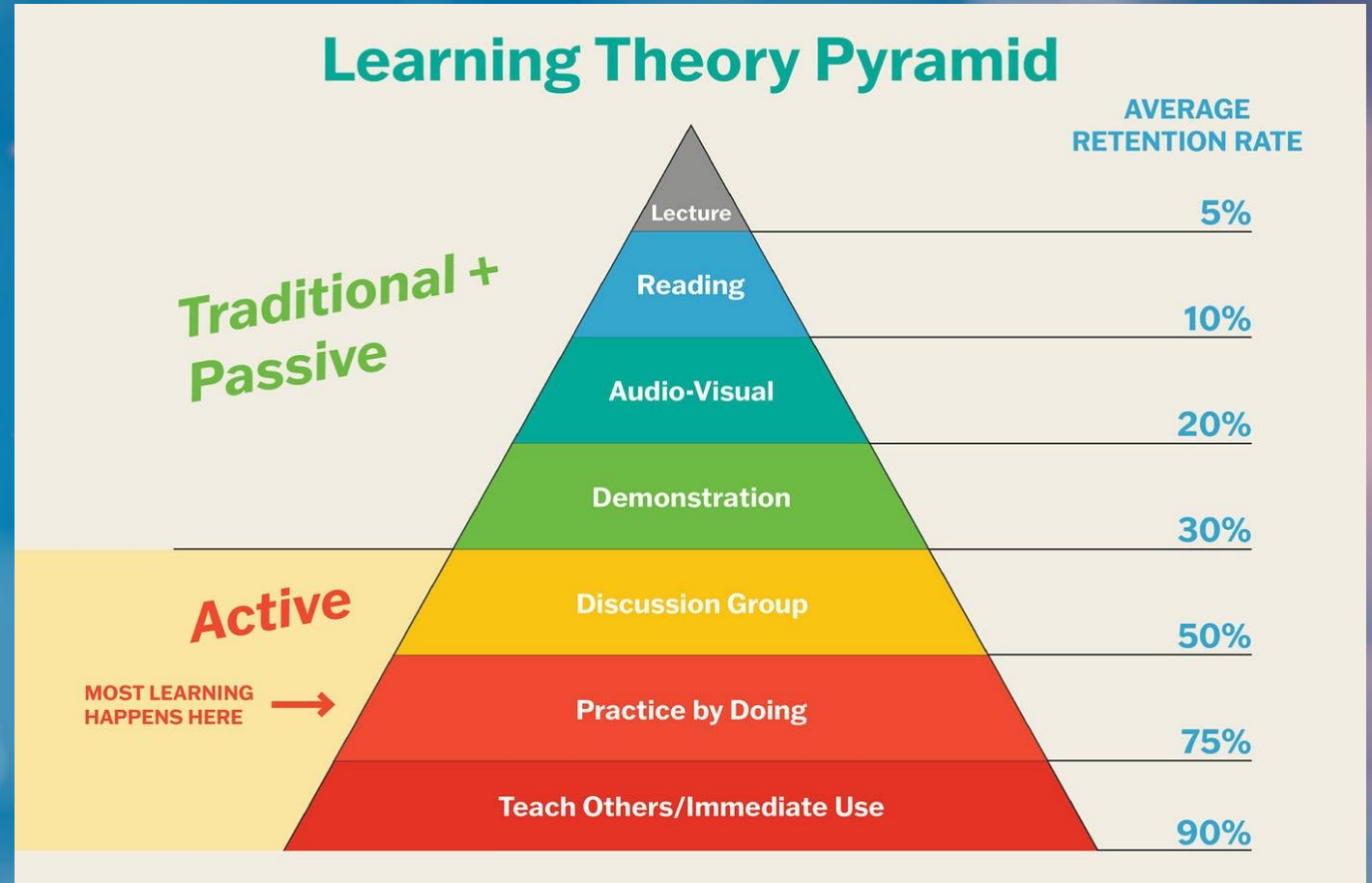


From zero to hero



What to expect ?

(when you're expecting..)





#7 ~~Component testing~~

#6 ~~Page component model~~

#5 Page object model

#4 Network requests

#3 Hooks & fixtures

#2 Custom commands

#1 Locators & selectors

#0 (setup & prep)





Preparations

1. Technical requirements:

- a. Node (18.X --> LTS)
- b. Git

2. SUT :

- a. <https://github.com/marcelblijleven/testrpg> (source code)
- b. <https://test-rpg.vercel.app/>

#0 (a) Setup local cypress project

1.Clone: github.com/EwaldVerhoeven/tc-cypress-workshop



```
~/repos$ clone  
git@github.com:EwaldVerhoeven/tc-  
cypress-workshop.git
```

2.Open your (cloned) project in your IDE



```
~/repos$ code tc-cypress-workshop
```

#0 (b) Setup local cypress project

1. Move into the project



```
~/repos$ cd tc-cypress-workshop
```


2. Install dependencies



```
~/repos/tc-cypress-workshop$ npm i
```


#0 (c) Setup local cypress project

1. Add 'E2E' object to config



```
import { defineConfig } from "cypress";  
export default defineConfig({  
  
  e2e: {  
  
  },  
});
```

#0 (d) Setup local cypress project

1. Add the following 7 settings:

- a. "supportFile"
- b. "specPattern"
- c. "fixturesFolder"
- d. "baseUrl"
- e. "watchForFileChanges"
- f. "screenshotsFolder"
- g. "videosFolder"

```
import { defineConfig } from "cypress";

export default defineConfig({

  e2e: {
    supportFile: "tests/support/e2e.ts",
    specPattern: "tests/e2e/*.cy.{js,jsx,ts,tsx}",
    baseUrl: "https://test-rpg.vercel.app",
    watchForFileChanges: false,
    screenshotsFolder: "tests/screenshots",
    videosFolder: "tests/videos",
    fixturesFolder: "tests/fixtures",

  },
});
```


#0 (e) Setup local cypress project

1. Add "chromeWebSecurity" and set value to 'false'
2. Add the following (e2e)settings:
 - A. "viewportHeight"
 - B. "viewportWidth"

```
import { defineConfig } from "cypress";

export default defineConfig({
  chromeWebSecurity: false,

  e2e: {
    supportFile: "tests/support/e2e.ts",
    specPattern: "tests/e2e/*.cy.{js,jsx,ts,tsx}",
    baseUrl: "https://test-rpg.vercel.app",
    watchForFileChanges: false,
    screenshotsFolder: "tests/screenshots",
    videosFolder: "tests/videos",
    fixturesFolder: "tests/fixtures",

    viewportHeight: 960, // like macbook-16
    viewportWidth: 1650, // bigger than macbook-16. To avoid
    horizontal scrolling
  },
});
```

Locators and selectors

- Are you (familiar with the) DOM?
- Locators in Cypres:
 1. `cy.get('myselector')`



```
cy.get(selector)
cy.get(alias)
cy.get(selector, options)
cy.get(alias, options)
```


Get the input element

```
cy.get('input').should('be.disabled')
```

Find the first `li` descendent within a `ul`

```
cy.get('ul li:first').should('have.class', 'active')
```

Find the dropdown-menu and click it

```
cy.get('.dropdown-menu').click()
```

source: <https://docs.cypress.io/api/commands/get>

Find 5 elements with the given data attribute

```
cy.get('[data-test-id="test-example"]').should('have.length', 5)
```

Find the link with an href attribute containing the word "questions" and click it

```
cy.get('a[href*="questions"]').click()
```

Find the element with id that starts with "local-"

```
cy.get('[id^=local-]')
```

source: <https://docs.cypress.io/api/commands/get>



```
<button  
  id="main"  
  class="btn btn-large"  
  name="submission"  
  role="button"  
  data-cy="submit"  
>  
  Submit  
</button>
```


(Warmup) exercise:

Write a selector query to yield the `<button>` element above

Selector	Recommended	Notes
<code>cy.get('button').click()</code>	⚠ Never	Worst - too generic, no context.
<code>cy.get('.btn.btn-large').click()</code>	⚠ Never	Bad. Coupled to styling. Highly subject to change.
<code>cy.get('#main').click()</code>	⚠ Sparingly	Better. But still coupled to styling or JS event listeners.
<code>cy.get('[name="submission"]').click()</code>	⚠ Sparingly	Coupled to the <code>name</code> attribute which has HTML semantics.
<code>cy.contains('Submit').click()</code>	✅ Depends	Much better. But still coupled to text content that may change.
<code>cy.get('[data-cy="submit"]').click()</code>	✅ Always	Best. Isolated from all changes.

#1(a) Write your first test

- Add: (test)file to your e2e folder called *rpgManualLogin.cy.ts*



```
describe("My first test", function () => {  
  it("Successful manual login", function () {  
    // automation & test code  
  })  
})
```

Running your tests

- *Command line (HEADLESS)*

```
npm  Yarn  pnpm  
npx cypress run
```

- *Cypress test runner (HEADED) --> DEMO!*

```
npm  Yarn  pnpm  
npx cypress open
```


#1(b) Write your first test

- Automate the steps below

*Hint: api's you'll need are **cy.visit()** / **cy.get()** / **.click()** / **.type()***

Step	actions
1	Click on the 'login' button
2	Enter a valid email
3	Enter a password
4	Press 'enter' key OR Click on 'login' button in modal

#1(c) Write your first test

- **Add: *assertions*** (according to 'results' column below) for **step 1 and 4**

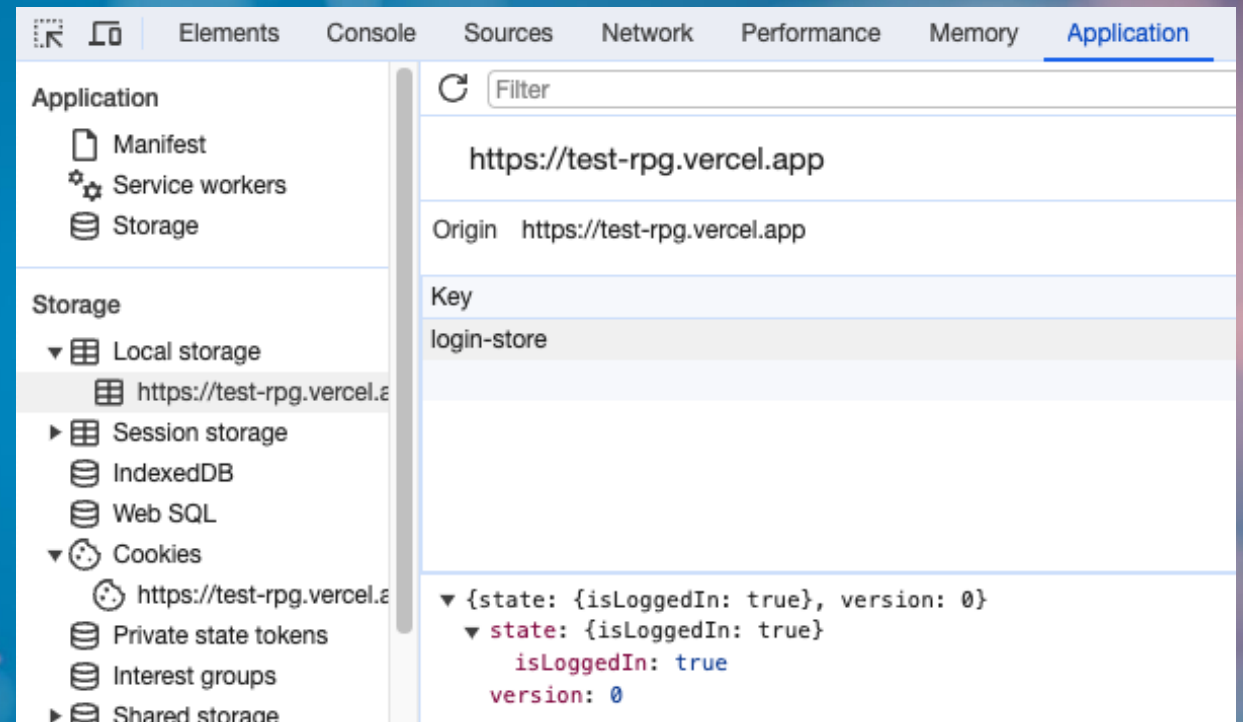
Step	actions	result
1	Click on the 'login' button	Modal with form appears
2	Enter a valid email	No validation messages
3	Enter a password	
4	Press 'enter' key OR Click on 'login' button in modal	Modal exits and button (step 1) now displays 'Logout'

Hint: api you'll need **.should()**

#1(d) Write your first test

- **Add:** Assertions to check that you are (also technically) logged in

Hint:
*What is in your browsers
local storage?*



#2 Custom commands

Exercises:

1. Add a file called 'commands.ts' in *./tests/support*
2. Write a **custom command** named '**login**' to automate/re-use the 'manual' login procedure (as done in previous exercise).
3. Add your custom command the Cypress namespace and import the commands module in your support file
4. Refactor *rpgManualLogin.cy.ts* and by using your 'login' custom command.
5. **BONUS:** Make use of the `.session()` api

#3 Hooks and fixtures

1. Automate the following script in a new test file called '*playTheGame.cy.ts*'

Step	Actions	Result
0	Navigate to baseUrl & Login	Browser at baseUrl
1	Click on <button> with text: "Click here to play"	Browser at 'play'
2	Enter Character name	
3	Select option from "Build"	Desired option is selected
4	Click on <button> with text: "Start!"	
5	Click (5 times) on the <button> with tekst: " Click me ... times"	Blue text appears that says 'Great job! You levelled up'
6	Upload a file by clicking 'choose file'	Blue text appears that says 'File selected, level up!'
7	Enter 'Lorem Ipsum' in the 'Type it' field	Blue text appears that says 'Dolar sit amet'
8	Slide the slider all the way to the right	Blue text appears that says 'Slid to the next level!'

Hooks syntax



```
before(function () => {
```

```
  cy.visit('/')
```

```
})
```

```
it('test #1', function () => {
```

```
  // rest of your test
```

```
})
```


#3 Hooks and fixtures

1. Refactor the test by putting step 0 into a 'hook'
2. Add an 'after' or 'afterAll' hook in which you 'end the game' and bring back the application to the home-page
(BONUS: or perform (simple) POST request to the available end-point)
3. Add a file under `./tests/fixtures` named 'testdata.json' and add an object with the following key/value pair2

```
{
  "role_1": {
    "name": "Henk",
    "build": "Mage"
  }
}
```

#3 (b) Hooks and fixtures

1. Add / load the fixture (*using `cy.fixture()`*) in the appropriate hook

Hint: use `cy.fixture()`

2. Refactor your test to use the fixture instead of 'hardcoded' data.

#4 Page object model (POM)

1. Add a subfolder 'pages' at ./tests/e2e and add an *index.ts* file
2. Add your first pageObject file and name it '
(*BONUS: you could add/define a base class in 'base.po.ts' for other page objects to inherit from*)
3. Define a page object (class) and a (readonly) attributes that represent html element and have the (cypress) "locator/code" as its value.
4. Add methods to the class that represent 'Actions' on the html elements. PAY ATTENTION TO READABLE NAMING!!!
5. Refactor both test files Implement/use these methods in your test (*.cy.ts)

#5 Network requests

1. Add a POST request to the available endpoint (using `.request()`). Put it in a after-hook.
Hint: First make a successful (test) request using a tool like postman
2. Let your test wait (and test) for the respons of the GET request (by the app) each time you 'level up'. Also assert the reponse code.
hint: Use the `.wait()` and `.intercept()`

[adult swim]

**DID YOU LEARN ANYTHING
FROM THIS EXPERIENCE?**



ewald@testcoders.nl



ewaldverhoeven.nl



<https://github.com/EwaldVerhoeven>



TESTCODERS

Technical Test Experts Nederland

📍 Utrecht, Nederland

👥 1.051 leden · Openbare groep ⓘ

👤 Georganiseerd door **Huub Jansen** and **3 others**

meetup