Jonathan Linat

How can I theoretically predict

what my screen reader will tell me? 🤔

April 2019 - September 2020

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Thanks to **Alexander Farrell**. *Translation review*

First of all.

What we know

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Web application

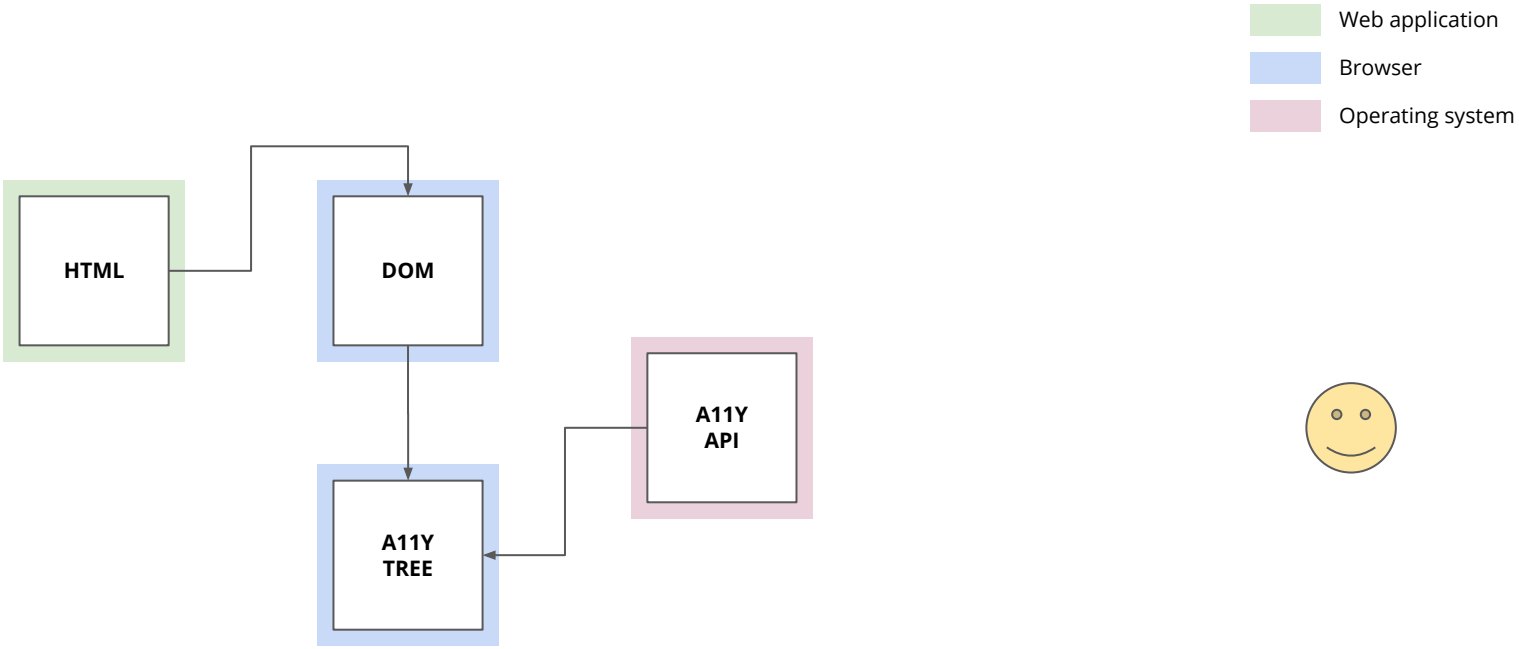
Nowadays, we are able to create **semantic** Web applications using **HyperText Markup Language**.

Semantic is related to **meaning** in language or logic.

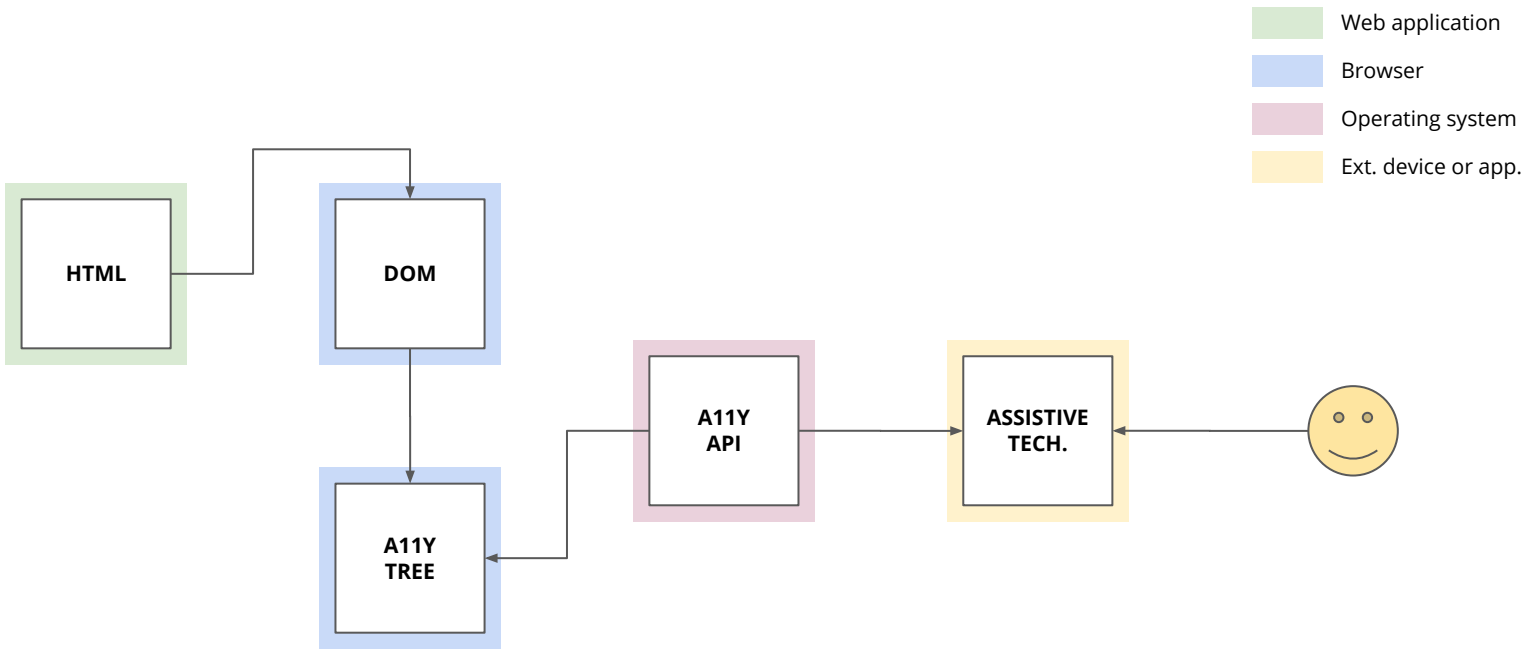
https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

The browser analises the content of the HTML document and generates an **object model** called DOM.

It also generates a parallel object model called **Accessibility Tree** based on the DOM.



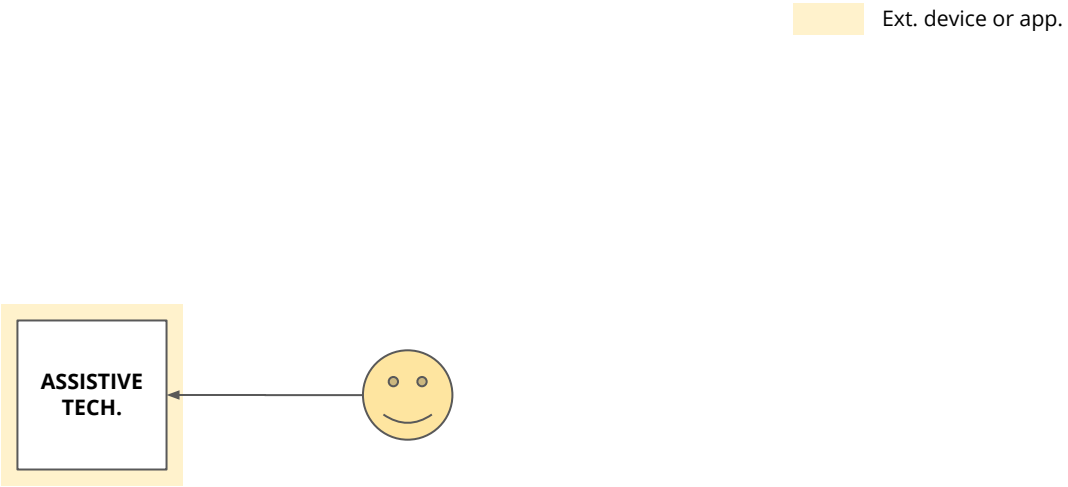
The operating system's **Accessibility API** reads and parses this tree.



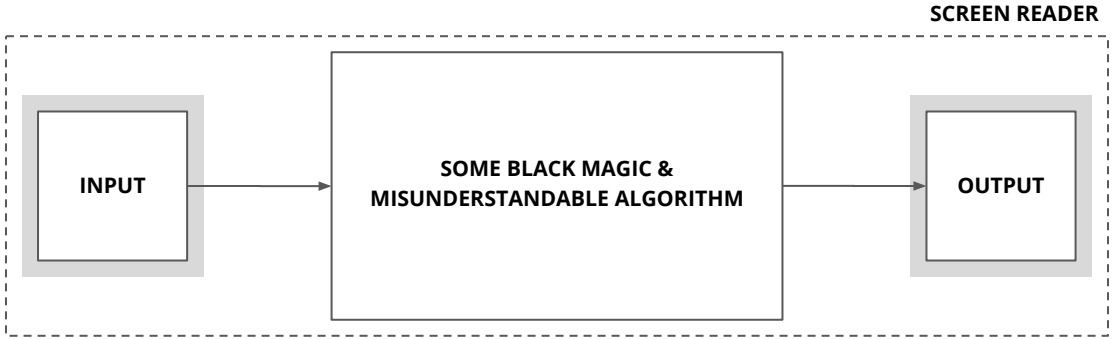
And through **assistive technology**, the user can access what the **Accessibility API** returns.

Mmmh, okay.

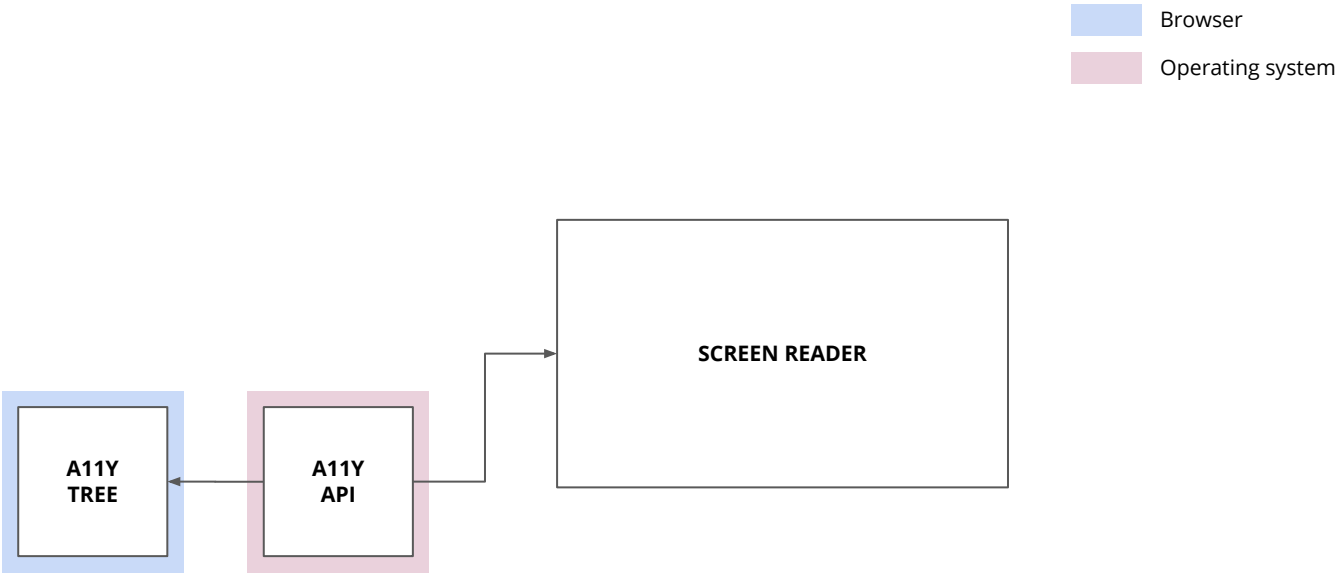
https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept



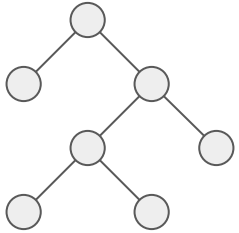
We also learned that **assistive technology** could be an external device or an application such as a **screen reader**.



A **screen reader** is nothing more than an **algorithm** that **receives** data and **returns** another.



What it receives is **processed data** from the operating system's **Accessibility API**.

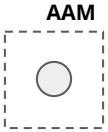


These data are **properties** present in nodes called **Accessibility nodes**.

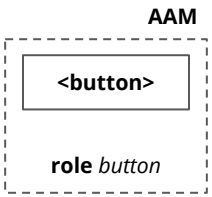
They are part of the browser's **Accessibility Tree**.

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

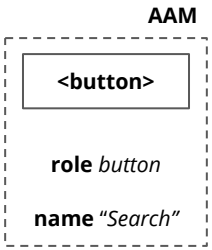
An **Accessibility node** is a different representation of a DOM element.



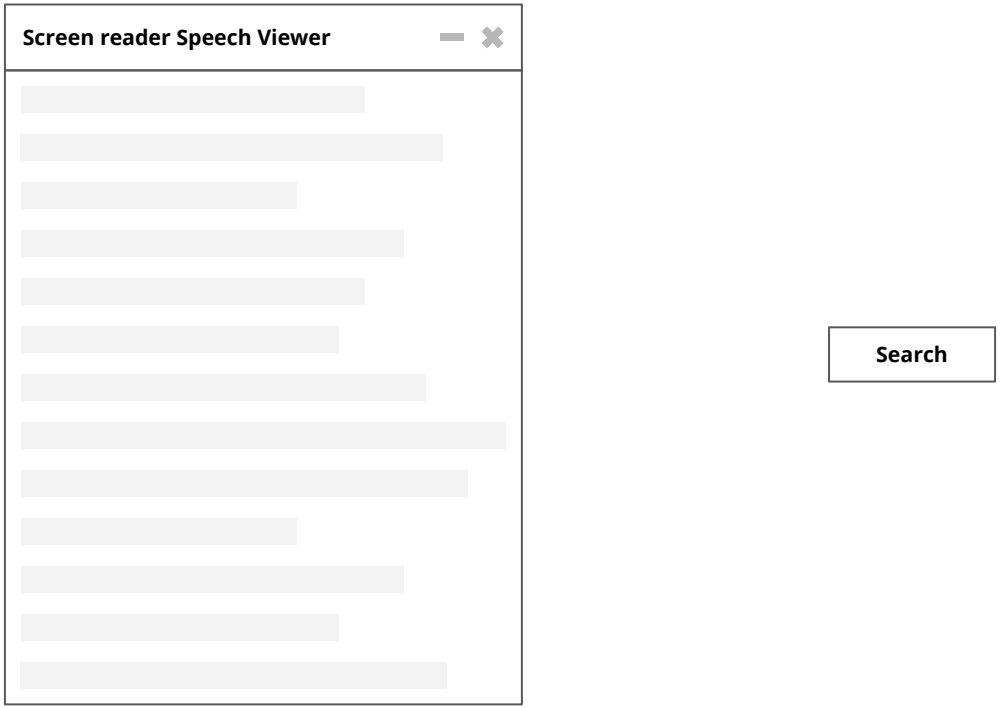
It has an associated **role** defined by the **HTML Accessibility API Mappings** specification.



In the case of a button, it has a **role** of *button*.



It also has several **properties** such as *name*, whose related value is its **label**.



So basically, what we do **expect** from a screen reader is that...

...it can **identify** the **meaning** of the **current active element**: mainly its *name* and its *role*.

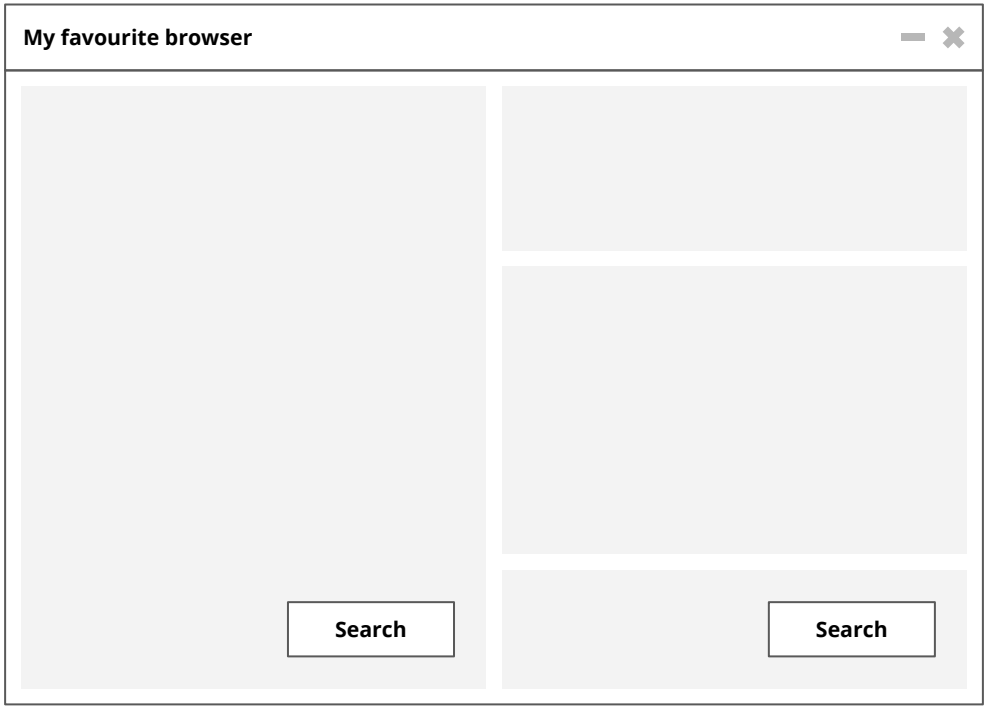
So logical.

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

But, yeah.

There is a problem

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept



Imagine having a **Web application** with **two visually equal** buttons.

Same *colour*, same *size*, some *typography*, same *behaviour*... You know, **equal**.

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Well, not really...

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

One button is created based on the HTML **button** tag...

...and the other one based on HTML **div** tag.

Search

```
<div tabindex="0">Search</div>
```

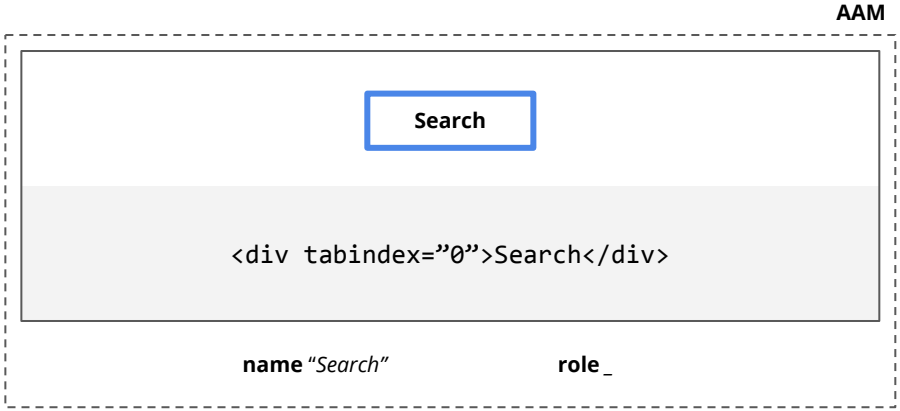
Search

```
<button type="submit">Search</button>
```

They have the **exact same behaviour** because of **JavaScript**.

But this button doesn't have the **same meaning**...

...as this one.



The **screen reader** will only return the **name** *Search*.

So...

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

How do you expect users to guess that
this element is a button and they can interact with it?

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

...

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

It’s just impossible!

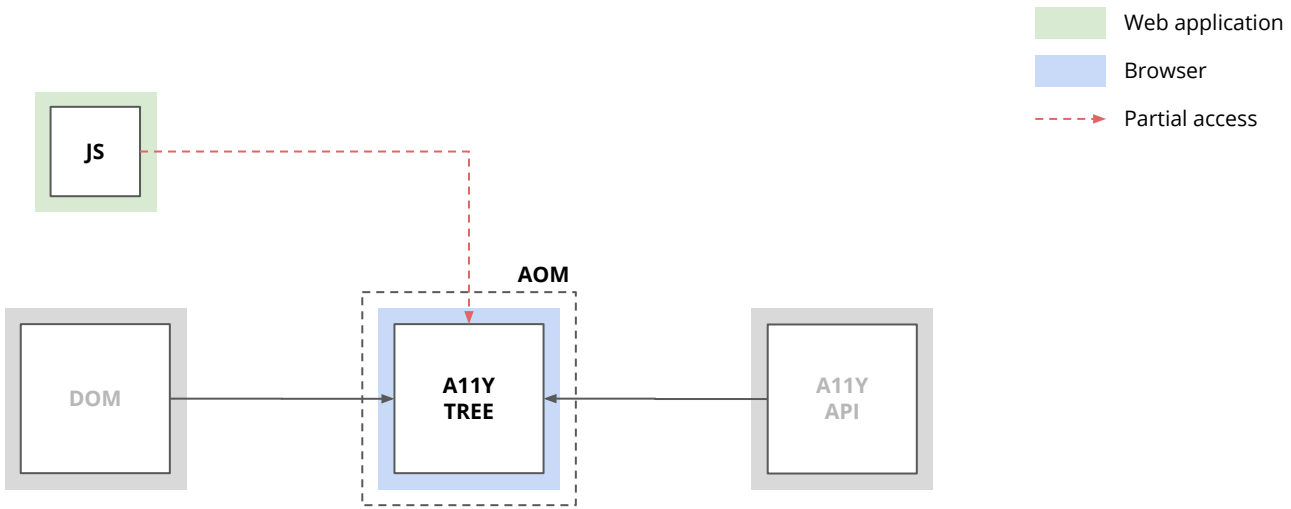
https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

This is why **you should always be aware** of the **meaning** of your **components**.

Wait a minute.

We may test their meaning

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept



We also know that it is **partially** possible to **access and consult** the browser's **Accessibility Tree**.

Accessibility Object Model

Unofficial Draft 16 August 2019

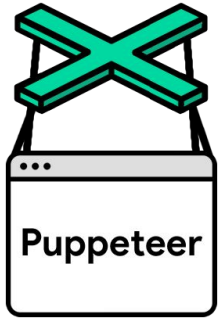
Editors:

Alice Boxhall (Google)
James Craig (Apple)
Dominic Mazzoni (Google)
Alexander Surkov (Mozilla)

This feature is part of an **unofficial specification** called **Accessibility Object Model**.

This specification is **currently only available** in **Google Chrome** as an experimental feature.

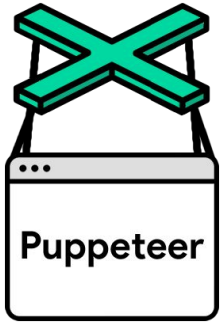
Google Chrome is based on Chromium.



And **Puppeteer** uses Chromium as its default browser.

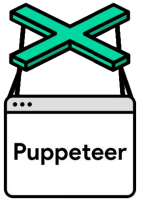
Wait, what? Puppeteer?

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept



Puppeteer is a Node library which provides a high-level API to control Chrome or Chromium over the DevTools Protocol.

It creates an up-to-date, automated testing environment and run tests directly in the latest version of Chrome using the latest JavaScript and browser features.



jest-puppeteer

That said, it can be used to run tests against Chromium, using the community-driven project called **jest-puppeteer**.

So cooool!

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

A great experiment.

Test and predict

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

```
window.getComputedAccessibleNode(Element el);
```

getComputedAccessibleNode() is the *Window* method we need to use.

```
> getComputedAccessibleNode(document.querySelector('#basicButtonSemantic[data-case="text"']))
<· ▼Promise {<pending>} ℹ
    ▶__proto__: Promise
     [[PromiseStatus]]: "resolved"
    ▼[[PromiseValue]]: ComputedAccessibleNode
       atomic: null
       autocomplete: null
       busy: null
       checked: "none"
       colCount: null
       colIndex: null
       colSpan: null
       disabled: false
       expanded: false
       firstChild: null
       keyShortcuts: null
       lastChild: null
       level: null
       modal: null
       multiline: false
       multiselectable: false
       name: "Visible label"
       nextSibling: null
      ▶parent: ComputedAccessibleNode {atomic: null, busy: null, disabled: false, expanded: false, modal: null, …
       placeholder: null
       posInSet: null
       previousSibling: null
       readOnly: false
       required: false
       role: "button"
       roleDescription: null
       rowCount: null
       rowIndex: null
       rowSpan: null
       selected: null
       setSize: null
       valueMax: null
       valueMin: null
       valueNow: null
       valueText: ""
      ▶__proto__: ComputedAccessibleNode
```

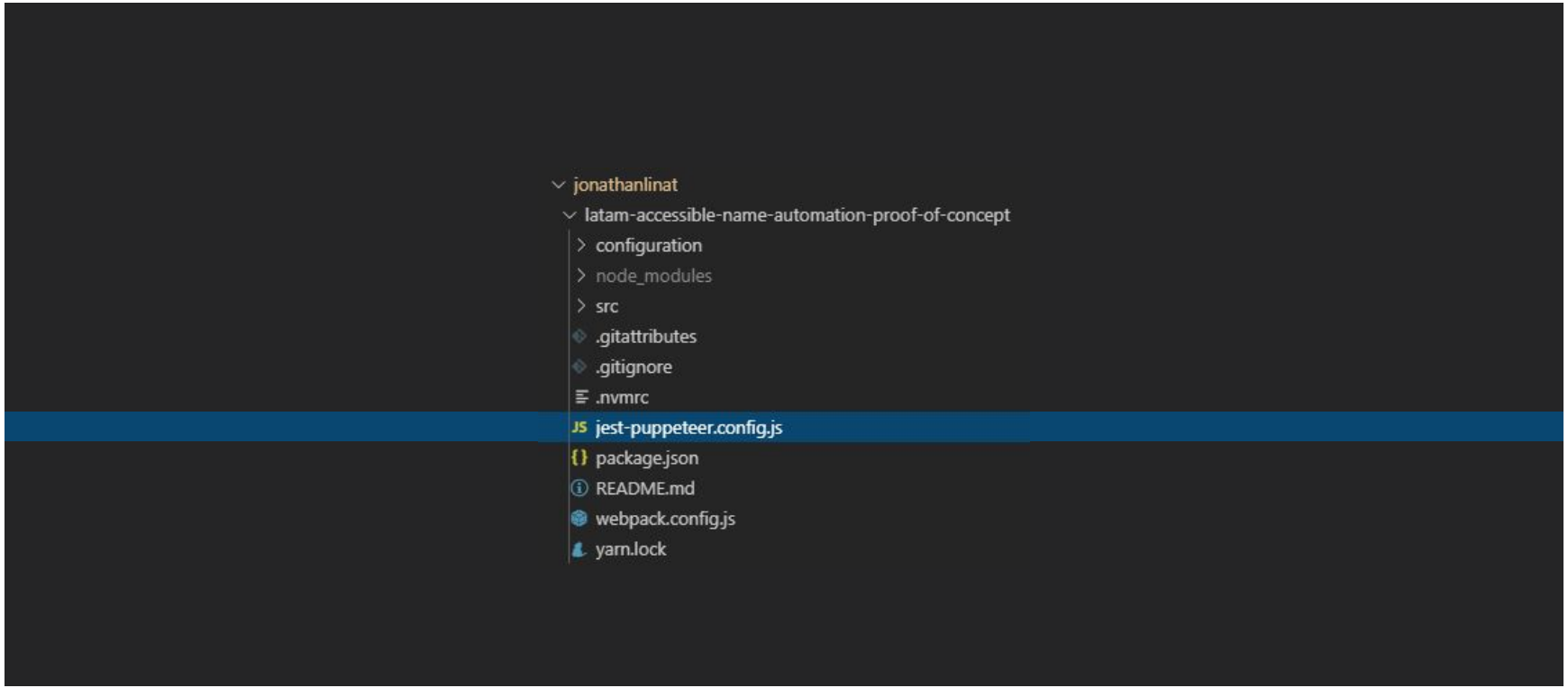
This method returns **all available properties** defined by **HTML Accessibility API Mappings**.

Okay. Let’s continue!

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

```
npm install --save-dev jest-puppeteer jest puppeteer
```

Install **jest-puppeteer** and its dependencies.



Create a file called **jest-puppeteer.config.js** located at the root of the project.

```
module.exports = {
  launch: {
    args: ['--enable-accessibility-object-model']
  }
}
```

Enable an experimental feature called **Accessibility Object Model,** pasting this piece of code into the file just created.

Create a test.

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

```
beforeAll (async () => {
  try {
    ({ name, role } = await page.evaluate(() => {
      return getComputedAccessibleNode(document.querySelector('button'))
        .then(data => {
          let object = {}
          for (key in data) object[key] = data[key]
          return object
      })
    }))
  } catch (err) {
    console.log(err)
  }
})
```

Before all expectation tests, **extract** *name* property and associated *role* from the **selected element**.

```
it('should have its accessible name equal to "Search"', async () => {
  await expect(name).toBe('Search')
})

it('should have its accessible role equal to "button"', async () => {
  await expect(role).toBe('button')
})
```

Finally, test and **expect** that the **selected element** has the **correct name property and role**.

```
npm run test
```

Execute.

```
● Accessible Name Automation Proof of Concept › Non semantic elements › Buttons › Inlined › Text › should have its accessible role equal to "button"

  expect(received).toBe(expected) // Object.is equality

  Expected: "button"
  Received: "genericContainer"

    482 |
    483 |           it('should have its accessible role equal to "button"', async () => {
  > 484 |             await expect(role).toBe('button')
        |                                ^
    485 |           })
    486 |         })
    487 |

    at Object.toBe (src/app/wrapper/app.accessiblename.spec.js:484:32)
```

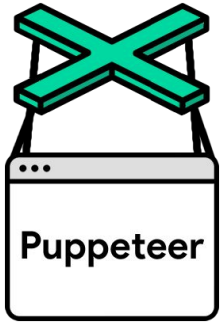
Aaaaaand, that’s the **final result**!

It failed because the tested component is a container, **not a button**.

Puppeteer.

Accessibility native class

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept



Puppeteer has its own Accessibility class.

The Accessibility class provides methods for inspecting Chromium's Accessibility Tree which is used by assistive technologies such as screen readers or switches.

Blink - Chrome's rendering engine - has a concept of "accessibility tree", which is then translated into different platform-specific APIs. Accessibility namespace gives users access to the Blink Accessibility Tree.

Most of the Accessibility Tree gets filtered out when converting from Blink AX Tree to Platform-specific AX-Tree or by assistive technologies themselves. By default, Puppeteer tries to approximate this filtering, exposing only the "interesting" nodes of the tree.

```
await page.accessibility.snapshot();
```

snapshot() captures the current state of the browser's Accessibility Tree.

```
console.log src/app/wrapper/app.accessiblename.spec.js:6

{
  role: 'WebArea',
  name: 'Accessible Name Automation Proof of Concept',
  children: [
    {
      role: 'heading',
      name: 'Accessible Name Automation Proof of Concept',
      level: 1
    },
    { role: 'heading', name: 'Semantic elements', level: 2 },
    { role: 'heading', name: 'Buttons', level: 3 },
    { role: 'heading', name: 'Basic', level: 4 },
    { role: 'button', name: 'Visible label' },
    { role: 'button', name: '' },
    { role: 'heading', name: 'Labelled', level: 4 },
    { role: 'button', name: 'Accessible label' },
    { role: 'button', name: 'Accessible label' },
    { role: 'GenericContainer', name: '' },
    { role: 'heading', name: 'Labelled', level: 4 },
    { role: 'GenericContainer', name: 'Accessible label' },
    { role: 'GenericContainer', name: 'Accessible label' },
    { role: 'heading', name: 'Inlined', level: 4 },
    { role: 'GenericContainer', name: 'Accessible label' },
    { role: 'GenericContainer', name: 'Accessible label' }
  ]
}
```

The returned object represents the **root Accessible node** of the page.

So, we also could **compare** this returned object with an expected mock. 😉

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Clone or download the proof of concept.

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Thanks!

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Extras

Used references

https://github.com/jonathanlinat/accessible-name-automation-proof-of-concept

Git at Google
Google Chrome Developers - *April 15, 2019*
https://chromium.googlesource.com/chromium/src.git/...

HTML Accessibility API Mappings 1.0
W3C - *July 15, 2019*
https://www.w3.org/TR/html-aam-1.0/

HTML Element test file index
The Paciello Group - *March 1, 2019*
https://thepaciellogroup.github.io/AT-browser-tests/

jest-puppeteer
Smooth Code - *July 23, 2019*
https://github.com/smooth-code/jest-puppeteer

The Accessibility Object Model (AOM)
WICG - *June 23, 2019*
https://wicg.github.io/aom/

HTML Standard
WHATWG - *August 14, 2019*
https://html.spec.whatwg.org/dev/

Automating Accessibility and Performance

Gildas Garcia - *July 18, 2018*

https://marmelab.com/blog/2018/07/18/accessibility-performance...

Puppeteer

Google Chrome Developers - *August 15, 2019*

https://github.com/GoogleChrome/puppeteer