
Web Accessibility

Plan

- Introduction
  - What is Web Accessibility ?
  - Ethics and deontology
  - Legal obligations
  - How to use the Web ?
  - Presentation of the various technical aids (NVDA, VoiceOver, etc.)
- Semantic Web
  - HTML
  - Accessiblity Tree
  - ARIA attributes
  - Microformats
  - Best Practices

Plan

- Standards
  - Section 508
  - WCAG 2.0
  - RGAA
  - WAI-ARIA
  - Accessiweb
- Technical Usecases
  - General design
  - Images
  - Links
  - Tables
  - Forms

Plan

- Technical Usecases
  - Modals
  - Complex components
  - Accessibility Best Practices with Vue.js
- Auditing
  - Manual evaluation
  - Automatic evaluation

Introduction

What is Web Accessibility ?

Ethics and deontology

Legal obligations

How to use the Web ?

Presentation of the various technical aids

ARIA attributes

- You must use first HTML Element
- When this it nos possible, and **only** when it is not possible
  - You can create your own HTML structure based on unsemantic elements
  - And add extra metadatas to give the missing semantic
- These metadats can also be used for non native HTML element (**tabs**, **modal**, ...)
- These attributes are defined by the **WAI-ARIA** specification

ARIA Specification

- These ARIA attributes are usefull if
  - you need to make your HTML code more semantic
  - you need to improve the support of Browser and AT
  - you need to use unsemantic HTML code if you need to style them more easily

ARIA Specification

- These metadats will impact only the Accessibility Tree
- They won’t have an impct
  - on the look and feel of your HTML
  - on the behavior
  - on the focus managment
  - on the navigation via the keyboard
- That’s the reason you must use semantic HTML element first.

ARIA Specification

- Aria Design Guide(https://www.w3.org/TR/wai-aria-practices-1.1/)

Example with a Checkbox

```
<div tabindex="0" role="checkbox" aria-checked="true">
    suivre la formation accessibilité
</div>
```

Roles

- A role will be added on any HTML element thank to the **role** attribute.

```
<tr role="...">
    ...
</tr>
```

Roles

- Les rôles sont découpés en six catégories :
  - Abstract Role
  - Document Structure Role (toolbar, …)
  - Landmark Role
  - Live Region Role
  - Widget Role
  - Window Role

Roles list

- alert
- banner
- checkbox
- complementary
- contentinfo
- combobox
- link
- search

Roles list

- switch
- radio
- radiogroup
- tab
- tablist
- tabpanel

Roles list

- toolbar
- tree
- treeitem

Landmark Role

- You can add roles to an existing semantinc HTML element
  - in order to provide a full support on browser / AT
  - in order to override the basic semantic

```
<nav role="navigation">
  <ul>
    <li><a href="#a">Dexter</a></li>
    <li><a href="#b">Doctor Who</a></li>
    <li><a href="#c">Futurama</a></li>
  </ul>
</nav>
```

Landmark Role

```
<form role="search">
  <label for="search-input">Search this site</label>
  <input type="text" id="search-input" name="search">
  <input type="submit" name="submit-btn" value="Search" />
</form>
```

Landmark Role

```
<footer role="contentinfo">
  <p>&copy; 2020 Small Business Ltd. All rights reserved.</p>
</footer>
```

State

- A `state` can be added thanks to the **aria-** prefix on any HTML elements.
- Theses states will be updated via JavaScript

```
<span aria-busy="true">
    ...
</span>
```

State

- Here is a short list of available ARIA states :
  - `aria-busy`
  - `aria-checked`
  - `aria-current`
  - `aria-disabled`
  - `aria-expanded`
  - `aria-hidden`
  - `aria-invalid`
  - `aria-pressed`
  - `aria-selected`

State

```
<form>
    <label>
        Name
        <input aria-invalid="true"/>
    </label>
    <button type="submit">
        <em class="fa fa-home" aria-hidden="true" />
        Valider
    </button>
</form>
```

Properties

- The **ARIA** specification provide also properties, that won't be updated via JavaScript
  - `aria-controls`
  - `aria-label`
  - `aria-labelledby`
  - `aria-live`
  - `aria-required`
  - ...

Custom Button

```
<button role="switch" aria-checked="true">
    Enable
</button>
```

Integration

- You must include Accessibility behavior inside your reusable components
- **But** you have to make configurable, in order to make fully reusable on all your page.

```
class SwitchButton extends HTMLElement {
  connectedCallback(){
    this.setAttribute('role', 'switch');
    this.setAttribute('aria-checked', 'false');
    this.setAttribute('tabindex', '0');
  }
}

window.customElements.define('button-switch', SwitchButton)
```

Integration

- Here is an example of how `Angular Material` *package* accessibility in their own components

```
<a mat-list-item routerLink cdkFocusRegionStart>
    Focus region start
</a>
<a mat-list-item routerLink>Link</a>
<a mat-list-item routerLink cdkFocusInitial>
    Initially focused
</a>
<a mat-list-item routerLink cdkFocusRegionEnd>
    Focus region end
</a>
```

Checkbox Custom

```
<p id="question">Question</p>
<ul aria-labelledby="question" role="group">
  <li role="checkbox" aria-checked="false" tabindex="0">
    <img aria-hidden="true" src="checkbox.svg" alt="Non sélectionné : " />
    Choix 1
  </li>
  <li role="checkbox" aria-checked="true" tabindex="0">
    <img aria-hidden="true" src="checkbox-checked.svg" alt="Sélectionné : " />
    Choix 2
  </li>
  <li role="checkbox" aria-checked="false" tabindex="0">
    <img aria-hidden="true" src="checkbox.svg" alt="Non sélectionné : " />
    Choix 3
  </li>
</ul>
```

Radio Custom

```
<p id="question">Question</p>
<ul aria-labelledby="question" role="radiogroup">
  <li role="radio" aria-checked="false" tabindex="-1">
    <img aria-hidden="true" src="radio.svg" alt="Non sélectionné : " />
    Choix 1
  </li>
  <li role="radio" aria-checked="true" tabindex="0">
    <img aria-hidden="true" src="radio-checked.svg" alt="Sélectionné : " />
    Choix 2
  </li>
  <li role="radio" aria-checked="false" tabindex="-1">
    <img aria-hidden="true" src="radio.svg" alt="Non sélectionné : " />
    Choix 3
  </li>
</ul>
```

Collapsible Panel

- When you need to develop your own `Collapsible Panel`, you need to add these attributes :
  - `aria-controls`
  - `aria-expanded`
- These attributes need to be updated based on the interaction of the user.

```
<button aria-controls="list" aria-expanded="true">Open</button>
<ul id="list" hidden>
    ...
</ul>
```

Write semantic CSS declarations

- You can use this ARIA roles, properties and states on your CSS selector
- Your CSS stylesheet will be more semantic
- You won’t have to create useless CSS class

```
input[aria-invalid='true']{
    background: red;
}
```

HTML

Accessiblity Tree

Screen Readers

- Used mostly by people with visual disabilities
- It converts a digital content to an audio or braille stream
- Provide some shortcust in order to reduce the reading of the page
- When the Screen reader reads the content of the page, the speed is at 350 word per minu

Screen Readers

- How a screen reader is able to communicate with a browser ?
  - The browser will generate :
  - DOM: Document Object Model
  - CSSOM: CSS Object Model
  - Accessibility Tree
- The Accessibility Tree will be exposed to native Accessibility *driver*
  - OSX Accessibility Platform (OSX)
  - UI Automation (Microsoft)
  - MS Active Accessibility (Microsoft)
  - Accessibility Toolkit (ATK)
- Screen Readers will also use these natives drivers
- Everytime an *event* is triggered on your web page, the browser will expose the new state of your tree

Screen Readers

```
<html>
    <head><title>Demo</title></head>
    <body>
        <label for="name">Name</label>
        <input id="name" value="Manu"/>
        <div>
            <button>OK</button>
        </div>
    </body>
</html>
```

- Here is an example of a generated **Accessibility tree**

```
id=1 role=WebArea name="Demo"
    id=2 role=Label name="Name"
    id=3 role=TextField value="Manu" labelledByIds=[2]
    id=4 role=Group
        id=5 role=Button name="OK"
```

- Chromium-based browser provide tools in order to visualize this tree.

Screen Readers

- NVDA (*)
- JAWS (*)
- Voice Over (iOS)
- TalkBack (Android)
- Narrator
- ChromeVox (Chrome plugin)

(*) More than 80% of the market

Screen Readers

- https://youtu.be/5R-6WvAihms[Assistive Tech - VoiceOver]
- https://youtu.be/bCHpdjvxBws[Assistive Tech - VoiceOver on iOS]
- https://youtu.be/0Zpzl4EKCco[Assistive Tech - Talkback]
- https://youtu.be/Jao3s_CwdRU[Assistive Tech - NVDA]

Rotor

- Each Screen Readers provide a **Rotor**
- Feature that allow to
- visualize the structure of the page
- browse easliy

Shortcusts

- Here are some example of shortcuts used on VoiceOver
  - VO+U : in order to open `rotor`
  - VO+ <-/-> : For browsing
  - VO+Fn+ -> : End of the page
  - VO+Fn+ <- : Start of the page
  - VO+Commande+L : Next link
  - VO+Commande+H : Next header
  - …

Demo

- Launch VoiceOver
- open the Rotor
- Navigate and Filter on the Rotor
- Navigate on the page
- Reading content
- Speed

PWX

- Take few minutes to use a screen reader based on a website you already know
- Select the right screen reader based what your operating system support.


Standards

Section 508

WCAG 2.0

RGAA

WAI-ARIA

Accessiweb


Technical Usecases

General design

Focus Managment

- On a web page, only these three categories are focusable by default
  - Form Controls
  - Links
  - Buttons
- If you need to make an unsemantic element focusable, you need to use the **tabindex** attribute.
- The attribute accepts three values
  - -1 : the element will only be focusable programmatically
  - 0 : the element will be focusable like a form control, link or button
  - higher than 0 : we will change the focus order

Focus Managment

- Here are three HTML elements using the **tabindex** attribute.

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

<span tabindex="-1"> ... </span>

<div tabindex="1"> ... </div>
```

- The **span** is now focusable programmatically.

```
document.querySelector('span').focus();
```

Focus Managment

- A visual indicator should be added on a focused element.
- Most of the time, same CSS rules as the one use for the **hover** state

```
:focus {
    outline: 2px dotted var(--link-color);
}
```

- Only visible element should be focusable
  - You need to implement **Focus Trap**
  - For example **Modal**, **Burger Menu**, …
- Make important behavior accessible as soon as possible
  - For example: Cookies banner

Focus Managment

- How would you improve a page with a cookies banner ?
- How would you make invisible elements unfocusale ?
- How would you make items on the background unfocusable ?

Skip Link

- Simple behavior you can implement in order to bypass all the header of your website.
- A **SkipLink** is composed of
  - A hidden link
  - this link will appear when the user navigate via the keyboard
  - We will redirect the user to the main part of the page (thanks to **anchor**)
- Here is a list of website using SkipLinks
  - Starbucks
  - Amazon
  - Github
  - Google
  - ...

Skip Link

- Here an example of implementation for a Skip Link.

```
<a class="skip-link" href="#main">Go to main content</a>

<main id="main" tabindex="-1">
    ...
</main>
```

```
.skip-link {
    position: absolute;
    top: -40px;
    left: 0;
    background: green;
    color: white;
    padding: 9px;''''
    z-index: 100;
}
.skip-link:focus {
    top: 0;
}
```

Langs

- Some Screen readers will select the right voice based on the lang of your page
- You must
  - define the default lang of the web pag
  - define everytime you have a part of the page that is not using the default lang

```
<html lang="en">
  <body>
    <article lang="fr">
      Ceci est un contenu en français
    </article>
  </body>
</html>
```

Titles

- All pages of the same web application should have a unique **title**

```
<head>
  <title> How to write accessible code | EmmanuelDemey.dev</title>
</head>
```

- An HTML page is like a Word document. The **outline** should be clear
  - h1 > h2 > h3 > … > h6

```
Array.from(document.querySelectorAll('h1, h2, h3, h4, h5, h6'))
```

Titles

- When you develop a **SPA**, you have give manually the focus everytime the user is redirected from one page to another.

```
const app = new Vue({
    watch: {
        $route: function(to) {
            this.$nextTick(function () {

          let focusTarget = this.$refs.routerView.$el;

        focusTarget.setAttribute('tabindex', '-1');

        focusTarget.focus();

        focusTarget.removeAttribute('tabindex');
            });
        }
    }
}).$mount('#app');
```

- A PR is opened in order to manage thie behavior internally in Vue.js : https://github.com/vuejs/vue-router/issues/2488

Images

- An image should contains a text alternative via the **alt** attribute.
  - if the image does not contain any information **alt=""**
  - if the image contain some information, the value should be accurate

```
<img src="error.svg" alt="" />

<img src="./conference.png" alt="Conference about Web Accessibility by Emmanuel Demey" />

<picture>
  <source srcset="img/kinepolis.avif" type="image/avif">
  <source srcset="img/kinepolis.webp" type="image/webp">
  <img loading="lazy" src="/img/kinepolis.jpg" alt="Une salle du Lille Grand Palais">
</picture>
```

- Be careful when you have multiple text alternatives

```
<a href="#" aria-label="Twitter">
  <img src="#" alt="Twitter Logo"/>
</a>
```

Links

Tables


Forms

Form Controls

- When developing a form, we have multiple HTML element available
  - inputs (text, date, number, …)
  - textarea
  - checkbox and radio
  - textarea
  - select
  - …

Labels

- The most important rule when managing form is to link a label to all inputs
- Multiple solutions are available
  - input inside the label
  - the label linked to the input thanks to the **for** attribute
  - using the **aria-labelledby** or **aria-label** attributes
  - a placeholder **is not** a solution
- When we click on the label, the corresponding input should become focused (*easy to test*)

Labels

- Here are some examples about defining a label to an input

```
<label> Name <input /></label>

<label for="firstName">Name</label> <input id="firstName" />

<div id="firstName-label">Name</div> <input aria-labelledBy="firstName-label" />

<input aria-label="Name"/>
```

Labels

- Browser are not able *for the moment* to translate strings defined in a attribute.
  - If you need to define an invisible label and the browser need to be able to translate it,
  - We will use a utility CSS class

```
.sr-only {
  border: 0 !important;
  clip: rect(1px, 1px, 1px, 1px) !important;
  -webkit-clip-path: inset(50%) !important;
  clip-path: inset(50%) !important;
  height: 1px !important;
  overflow: hidden !important;
  margin: -1px !important;
  padding: 0 !important;
  position: absolute !important;
  width: 1px !important;
  white-space: nowrap !important;
}
```

```
<div class="sr-only" id="firstName-label">Name</div> <input aria-labelledBy="firstName-label" />
```

Errors Managment

- When the form has errors, we should make the UX as smooth as possible
- Here is few rules
  - do not use only colors
  - do not use only icons
  - define the accepted value as soon as possible
  - linked the input to some interesting information via the **aria-describedby** attribute
  - after submitting the form, give the focus to the first element with an error

```
<label for="firstName">Name</label>
<input id="firstName" aria-describedby="firstName-error"/>
<div id="firstName-error" class="error"> Ce champ est obligatoire </div>
```

Errors Managment

- When an input is invalid, you should set the **aria-invalid** attribute
- You do not need extra CSS class like the **error** CSS class used previously
- You can now use more semantic CSS rules

```
<label for="firstName">Name</label>
<input id="firstName" aria-invalid="true" aria-describedby="firstName-error"/>
<div id="firstName-error" class="error"> Ce champ est obligatoire </div>
```

```
[aria-invalid='true']{
  background: red;
}
```

Global Errors

- Sometimes we have a global errors block
- You should add focus to this element in order link the user to the beggining of the form
- In order to be focusable, you should add the tabindex attribute to the main HTML element of this block.

```
<form>
  <div id="error-blocks" tabindex="-1"> ... </div>
  <button> Valider le formulaire </button>
</form>
```

```
document.querySelector('form').addEventListener('submit', () => {
  document.querySelector('#error-blocks').focus();
})
```

Autocomplete

- The **autocomplete** attribute make possible to provide an autocomplete behavior based on previous visited webpage

```
<label for="frmNameA">Name</label>
<input type="text" name="name" id="frmNameA"
  placeholder="Full name" required autocomplete="name">

<label for="frmEmailA">Email</label>
<input type="email" name="email" id="frmEmailA"
  placeholder="name@example.com" required autocomplete="email">

<label for="frmPhoneNumA">Phone</label>
<input type="tel" name="phone" id="frmPhoneNumA"
  placeholder="+1-555-555-1212" required autocomplete="tel">
```

DataList

- Component used to provide an easy autocomplete behavior
- Proposals are not based on the **user's data**

```
<label for="ice-cream-choice">Choose a flavor:</label>
<input list="ice-cream-flavors" id="ice-cream-choice" name="ice-cream-choice" />

<datalist id="ice-cream-flavors">
    <option value="Chocolate">
    <option value="Coconut">
    <option value="Mint">
    <option value="Strawberry">
    <option value="Vanilla">
</datalist>
```

Dynamic Autocomplete

- If you need to implement a dynamic autocomplete (with data coming from the server for example),
- you need to follow the **Combobox Pattern** from the WAI ARIA Authoring Practices Guide
- https://www.w3.org/WAI/ARIA/apg/patterns/combobox/

Complex components

Microdata

- API used to *business* metadata to an HTML page
- Useful in order make our website unserstandable by a device
- You can multiple syntaxes
  - Schema.org
  - JSON LD
  - RDF

Schema.org

- Supported by Bing, Google, Yahoo
- It provides a huge vocabulary
  - Person
  - Restaurant
  - Event
  - Product
  - …

Vocabulary

- Each vocabulary provides multiples properties you can add on your HTML
- Here is an example for the vocabulary `Restaurant` : ** name ** image ** geo ** events ** ...

Vocabulary

- In order to add these metadatas to your HTML, you need to define which node will be the root of your vocabulary.
- You will use `itemscope` and `itemtype` attributes.

```
<section itemscope itemtype="https://schema.org/Restaurant">
    <h1>Super Taco Bar</h1>
    <h2>Menu</h2>
    <p>
       Lille
    </p>
</section>
```

Vocabulary

- We can now add attributes to all the children of this root node
- We will use the `itemprop` attribute

```
<section itemscope itemtype="https://schema.org/Restaurant">
    <h1 itemprop="name">Super Taco Bar</h1>
    <h2>Menu</h2>
    <p itemprop="address">
       Lille
    </p>
</section>
```

Vocabulary

- We can have nested vocabularies

```
<section itemscope itemtype="https://schema.org/Restaurant">
    <h1 itemprop="name">Super Taco Bar</h1>
    <h2>Menu</h2>
    <div itemprop="address" itemscope itemtype="https://schema.org/PostalAddress">
        <p itemprop="addressLocality">Lille</p>
    </div>
</section>
```

LD+JSON

- If you prefer, you can use a JSON-like format when defining these matadats

```
<script type="application/ld+json">
{
  "@context": "http://schema.org/",
  "@type": "Person",
  "name": "Emmanuel Demey",
  "url": "http://twitter.com/EmmanuelDemey"
}
</script>-
```

Accessibility Best Practices with Vue.js


Auditing

Manual evaluation

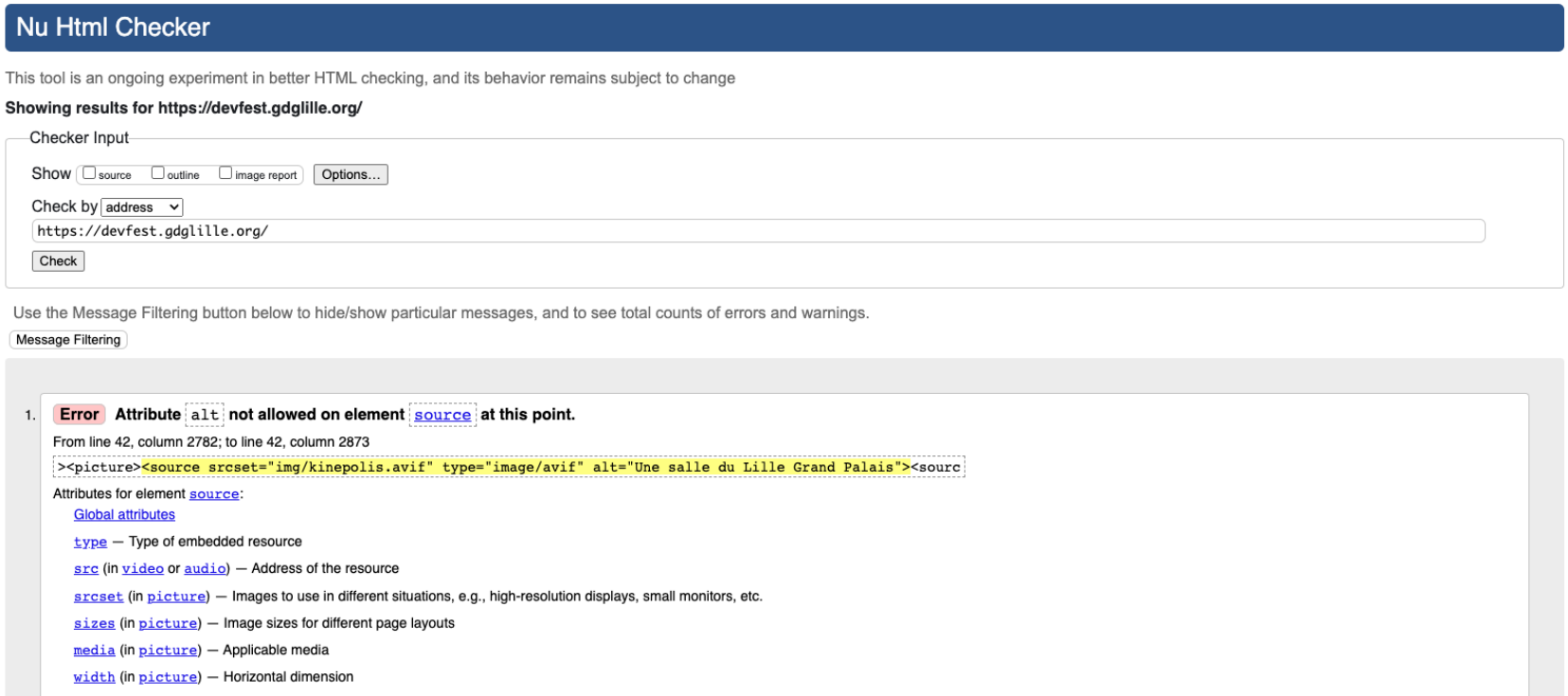
- Tools are not able to find all **quality** issues
  - accessibility
  - UX
  - Performance
- You need manual evaluation in order to be sure your application is
- Add **Is the feature fully accessible?** in your definition of done
- Work people with disabilities during the build of the application

Automatic evaluation

- We will talk about tools
  - tools usable via the browser
    - W3C Validators
    - Axe
    - Lighthouse
  - tools usable in your CI
    - ESLint
    - Pally
    - Testing Library
    - Playwright

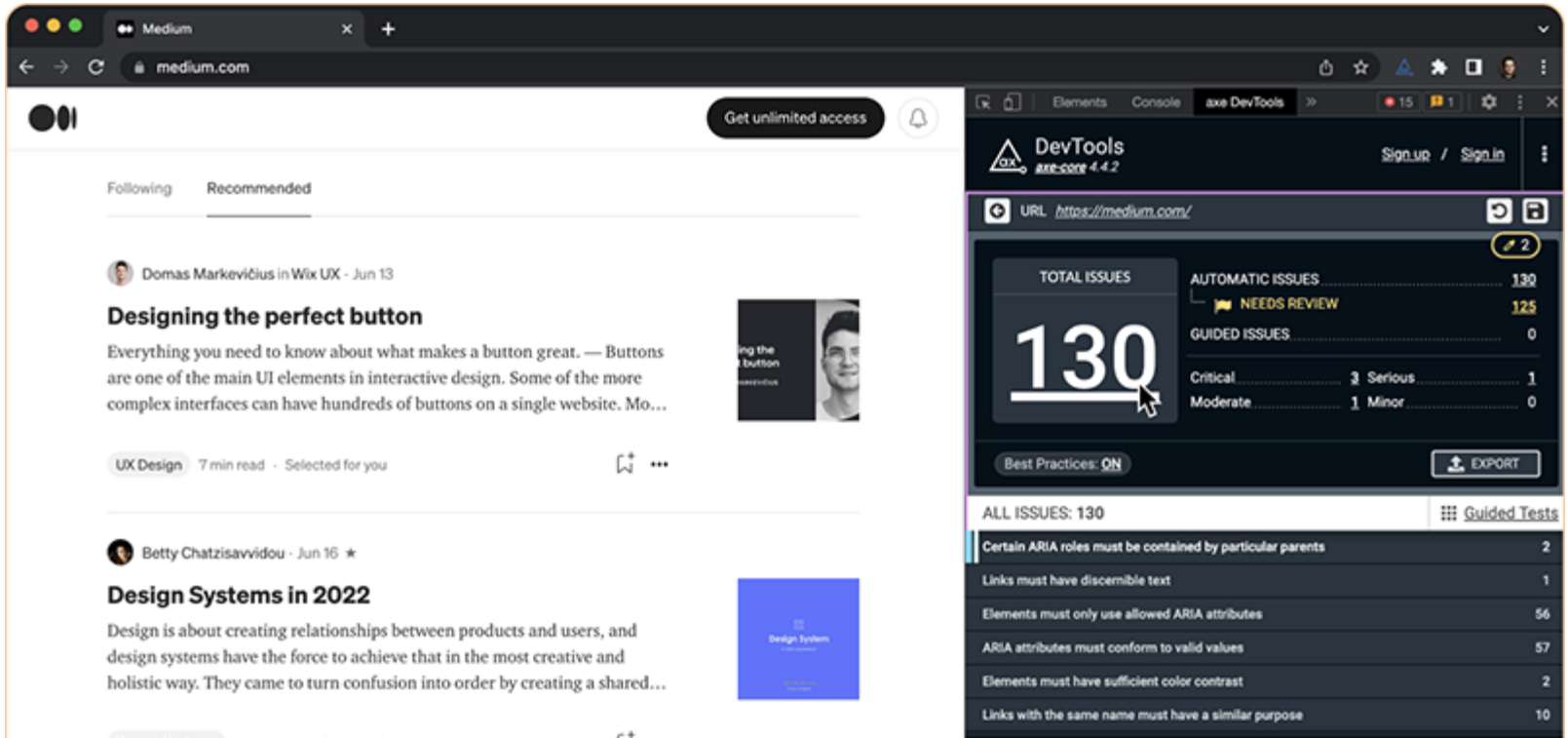
W3C Validators

- A valid HTML code is an HTML code following the semantic of the language
- A good HTML foundation is a good start for an accessible application.



Axe

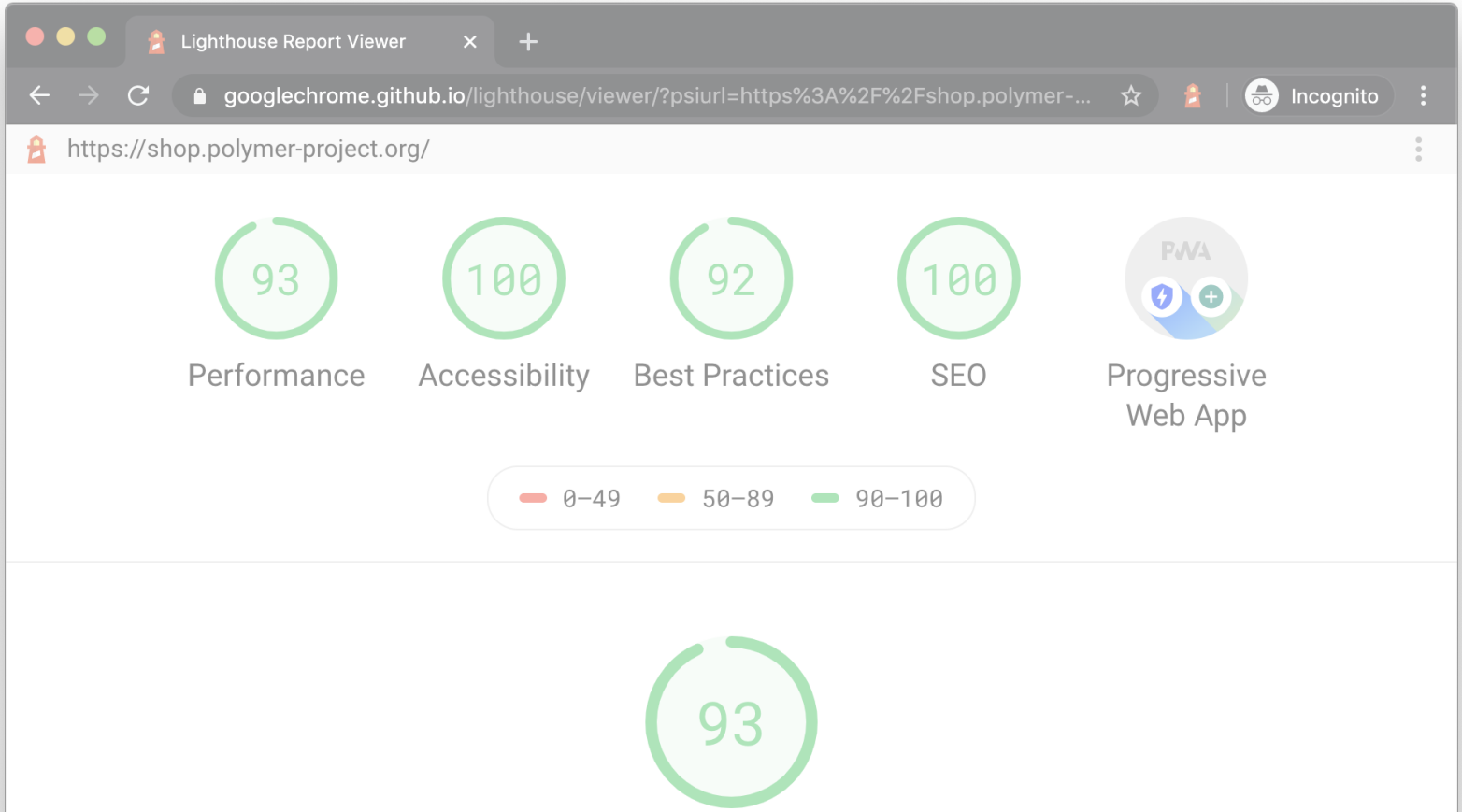
- Browser extension developed by the **Deque** company
- Usable via the terminal also



Lighthouse

- Available on all Chromium-based browser
- Usable via the terminal
- Define multiple checks categories
  - performance
  - accessibility
  - best practices
  - seo
- The **Accessibility** category is based on **Axe**

Lighthouse


Lighthouse Report Viewer
googlechrome.github.io/lighthouse/viewer/?psiurl=https%3A%2F%2Fshop.polymer-...
Incognito
https://shop.polymer-project.org/
93
Performance
100
Accessibility
92
Best Practices
100
SEO
PWA
Progressive Web App
0–49
50–89
90–100
93

Linting

- The **ESLint** exosystem provide a dedicated plugin for detected accessibility issues on a Vue.js application

```
npm i -D eslint-plugin-vuejs-accessibility
```

- You need to enable the plugin

```
{
  "plugins": ["vuejs-accessibility"]
}
```

- And finally enable specific rules or activated the recommanded ones

```
{
  "rules": {
    "vuejs-accessibility/rule-name": "error"
  }
  //"extends": ["plugin:vuejs-accessibility/recommended"]
}
```

Pally

- Another tool usable programmatically or on the terminal in order to audit a webpage

```
//pa11y https://example.com/

const pa11y = require('pa11y');

pa11y('https://example.com/').then((results) => {
    /*
    {
        documentTitle: 'The title of the page that was tested',
        pageUrl: 'The URL that Pa11y was run against',
        issues: [
            {
                code: 'WCAG2AA.Principle1.Guideline1_1.1_1_1.H30.2',
                context: '<a href="https://example.com/"><img src="example.jpg" alt=""/></a>',
                message: 'Img element is the only content of the link, but is missing alt text. The alt text should des
                selector: 'html > body > p:nth-child(1) > a',
                type: 'error',
                typeCode: 1
            }
        ]
    }
    */
});
```

Testing Library

- **Testing Library** well known utility package for testing React.js applicaiton
- Usable also with other frameworks like Vue.js
- Provide utility methods in order to interact with HTML elements by their roles
  - *ByRole
  - *ByLabelText
- Good practice in order to be sure that your HTML is semantic

```
npm i -D @testing-library/vue
```

Testing Library

```
import {render, fireEvent} from '@testing-library/vue'
import Component from './Component.vue'

test('properly handles v-model', async () => {
  const {getByLabelText, getByText} = render(Component)

  const usernameInput = getByLabelText(/username/i)

  await fireEvent.update(usernameInput, 'Bob')

  getByText('Hi, my name is Bob')
})
```

Playwright

- Solution used to write end-to-end tests
- Provide a package in order to run accessibility checks on tested pages

```
npm i -D @axe-core/playwrite
```

```
import { test, expect } from '@playwright/test';
import AxeBuilder from '@axe-core/playwright'; // 1

test('should not have any automatically detectable WCAG A or AA violations', async ({ page }) => {
  await page.goto('https://your-site.com/');

  const accessibilityScanResults = await new AxeBuilder({ page })
    .withTags(['wcag2a', 'wcag2aa', 'wcag21a', 'wcag21aa'])
    .analyze();

  expect(accessibilityScanResults.violations).toEqual([]);
});
```