network

MDPI

Article

Evaluating Progressive Web App Accessibility for People with Disabilities

Konstantinos I. Roumeliotis and Nikolaos D. Tselikas *

Department of Informatics and Telecommunications, University of Peloponnese, 22100 Tripoli, Greece; k.roumeliotis@uop.gr
* Correspondence: ntsel@uop.gr; Tel.: +30-2710372216

Abstract: App development is a steadily growing industry. Progressive web apps (PWAs) constitute a technology inspired by native and hybrid apps; they use web technologies to create web and mobile apps. Based on a service worker, a caching mechanism, and an app shell, PWAs aim to offer web apps with features and user interfaces similar to those of native apps. Furthermore, technological development has created a greater need for accessibility. An increasing number of websites, even government ones, are overlooking the need for equal access to new technologies among people with disabilities. This article presents, in a systematic review format, both PWAs and web accessibility and aims to evaluate PWAs' effectiveness as regards the corresponding accessibility provided.

Keywords: progressive web apps; pwa; web accessibility; wcag 2.2

Citation: Roumeliotis, K.I.; Tselikas, N.D. Evaluating Progressive Web App Accessibility for People with Disabilities. *Network* **2022**, *2*, 350–370. https://doi.org/10.3390/network2020022

Academic Editor: Andreas Kassler

Received: 16 April 2022
Accepted: 6 June 2022
Published: 8 June 2022

1. Introduction

An increasing number of users are interacting with websites and mobile apps daily, and mobile apps are experiencing tremendous growth, with users appreciating the usability and the user experience (UX) they offer. The growing users' need for mobile apps has led to the discovery of hybrid apps that combine web technologies and native functions. However, both hybrid and native apps have some limitations, such as a free space commitment on devices and more difficult multiplatform upgrades. In 2015, Google provided a solution to the above-mentioned restrictions with the introduction of a new technology called progressive web apps (PWAs). PWAs bring features we expect from native apps to the mobile browser [1]. They are built and enhanced with modern application programming interfaces (APIs) to deliver enhanced capabilities, reliability, and installability while reaching anyone, anywhere, and on any device with a single codebase [2].

At a time when web technologies are rapidly evolving, developers' interest in accessibility is fading. The need concerning people with disabilities for equal access to new technologies is growing. Even academic and government websites fail to offer real accessibility, thus making the lives of people with disabilities difficult [3].

Although much research focuses on the performance evaluation and search engine optimization of PWAs, their corresponding efficiency in terms of accessibility guidelines' conformance has not been investigated to date [1,2,4–6]. This article presents a review of the existing literature on PWA and accessibility technologies, and it is motivated by the need for accessibility solutions among platforms. The contributions of this article are summarized as follows:

- We include a discussion of PWAs and accessibility challenges as a guide for future research.
- We extensively review PWAs as a technology and articulate some drawbacks that arise.

- We critically investigate the accessibility technologies to date, including the newly introduced as a working draft Web Content Accessibility Guidelines (WCAG) 2.2.
- We combine and evaluate PWAs and accessibility technologies to promote web accessibility for anyone.

2. Materials and Methods

The purpose of this paper is to evaluate the PWAs' effectiveness against the Accessibility Guidelines and to promote the necessity of a more accessible web.

We followed six research stages to achieve the desired outcomes:

1. Literature review. Both PWAs and Accessibility Guidelines have been reviewed summarizing the existing literature.
2. Critical review. Both PWAs and Accessibility Guidelines address certain limitations that complicate the overall user experience (UX). From a critical point of view, we summarize those limitations.
3. Sample retrieval. In this stage, a sample of 20 PWA and 20 non-PWA websites are discovered and placed on a table.
4. Data retrieval. Each of the websites was fully tested against 10 accessibility evaluation tools—auditing tools. The results from the accessibility evaluation were added to a table.
5. Descriptive analysis. In this stage, using the Jupyter Notebook software, we made the descriptive analysis of our dataset.
6. Presenting the results of the descriptive analysis.
    a. Evaluation tools' limitations. After the descriptive analysis, the assumption that each accessibility evaluation tool performs its own measurements was confirmed.
    b. PWAs' accessibility limitations. It was also confirmed that PWAs are not fully accessible by default and that web developers must make an effort to conduct both machine and manual audits to achieve actual accessibility.
    c. PWAs' overall performance. In this stage, we conclude that PWAs are more accessible than conventional websites, observing that they have a lead in performance and SEO.

Figure 1 summarizes the methodology that has been used to conclude valuable outcomes regarding the PWAs' accessibility performance.

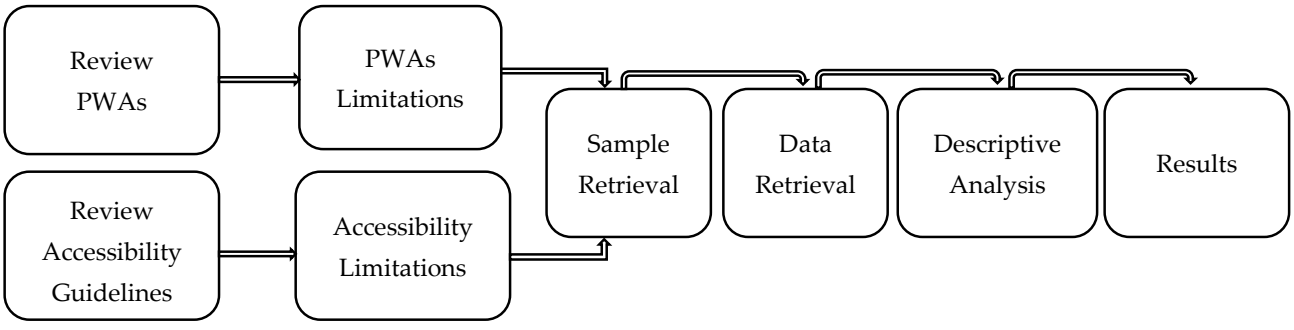

Figure 1. Overall step-by-step representation of the proposed methodology, reflecting the stages for evaluating PWAs' accessibility.

Research Limitations

Since PWAs are quite new as a technology, they also have a low adoption rate by web developers. To avoid inaccurate results from websites that have partially or wrongly adopted PWAs, for our analysis, we used the PWA websites presented as case studies on

the official Google page for PWAs. For each PWA website of our dataset, we chose a regular non-PWA website of the corresponding niche and industry (top-listed in Google search results).

The online tools used for the purposes of this paper are:

- Lighthouse by Google;
- Wave by WebAim;
- Web Accessibility by Level Access;
- Power Mapper;
- CSS Validation Service by W3;
- Nu Html Checker by W3;
- Axe DevTools Accessibility;
- Insights Chrome;
- Equal Access by IBM;
- Site Improve.

3. Progressive Web Apps (PWAs)

Prior research has found that almost four fifths (79%) of internet users access the internet on their mobile devices [7]. Statistical research agencies support the statistics that mobile internet traffic as a share of total global online traffic is greater than 55.64% and that, in 2020, mobile app downloads worldwide amounted to 218 billion [8].

Mobile devices are an indispensable part of human life nowadays. Mobile apps have been developed for every field and any need, from taking and editing photos to accessing social media, attending online meetings, and more. The incremental need for mobile apps led development companies to develop native or hybrid applications to overcome the limitations that the web as a platform imposes on mobile devices [1]. However, even mobile apps face some limitations, such as the following:

- Platform-specific applications;
- Devices' resources consumption;
- Multiplatform updates;
- Search engine optimization;
- Challenging app store optimization.

To overcome these limitations, in 2015, Google introduced a new technology, namely, PWAs, which are web applications built with modern APIs to deliver enhanced capabilities, reliability, and installability while reaching anyone, anywhere, and on any device with a single codebase [2]. As browsers become more modern, an increasing number of features become available to end users. This is known as progressive enhancement [5]. PWAs try to solve UX and user offline use. Moreover, they combine the advantages of apps and the web, such as being progressive, reliable, responsive, offline functional, interactive, installable, and notifiable [9]. PWAs are based on three main pillars:

- Capable: new and upcoming APIs are more capable than ever, introducing new capabilities, from file system access to app badging.
- Reliable: PWAs are reliable, fast, and dependable, regardless of network speed.
- Installable: PWAs run as apps on a user's home screen without any browser tab.

Furthermore, PWAs' main characteristics are as follows:

- Progressive: using service workers and a web app manifest, PWAs work on every modern browser.
- App-like: they use the app-shell model to provide app-style navigations and interactions [10].
- Installable: installed PWAs appear on the launch surfaces of any devices, such as the Mac OS X Applications folder and Spotlight function.
- Independently installed: they bypass App Store and Google Play installations.
- Responsive: PWAs are responsive and accessible from any platform and browser.

- Independent of connectivity: they are reliable even under unstable network conditions.
- Capable of delivering an offline experience: using service workers and the IndexedDB storage system, PWAs can provide offline experience.
- Safe: PWAs are served over Transport Layer Security (TLS) to ensure a secure connection.
- App badging: PWAs allow badging for app icons to subtly notify users of a new activity that might require their attention [2].
- Capable of supporting custom offline pages and splash screens: like native apps, PWAs support offline pages and splash screens.
- Fresh: PWAs are always kept up to date using the service worker update process.
- Discoverable: PWAs following W3C manifests are more likely to be discovered by search engines.

3.1. Service Workers

A service worker is a set of APIs introduced by PWA, running in its own thread and providing generic entry points for event-driven background processing that allows developers to programmatically cache and preload assets and data and manage push notifications, among other things [11]. More specifically, a service worker is a JavaScript script that runs in the background of the application, deploying in a separate thread from the UI, thereby avoiding app freezing. It acts as an intermediary between the app and the internet [4], and its main purpose is to execute functions as "promises," in a specific order, sending results back to the app. A JS Promise is an asynchronous managing mechanism that enables programmers to chain asynchronous computations while supporting proper error-handling methods [12]. Promises technically fix the gap between function execution order by telling the asynchronous method that it "promises" to call a given function as soon as the asynchronous (async) function is finished. An alternative method to promises is the callbacks. As regards the callbacks, the executing function knows in advance what has to be done when the asynchronous task has been completed. In promises, instead, the executing function returns an object (the promise), in which we describe what it has to do when the asynchronous task has been completed. A general approach is that the callbacks follow a more functional way of programming, instead of promises following a more object-oriented way.

Before service workers, the legacy technology called "AppCache" was the one that provided users an offline experience on the web. Although "AppCache API" was a straightforward solution to the offline experience, it faces a lot of issues that service workers avoid, such as caching all pages by default whether they have changed or not.

A service worker, as a middle service, is vulnerable to cross-site scripting (XSS) attacks, and a secure connection is, thus, required [13]. In addition, service workers have the power to provide async capabilities to an app, such as periodic sync, and act similarly to a cronjob performing specific actions at a certain date or time.

To set up a functional service worker, three main actions must be implemented: registration, installation, and activation. Through an on-page load, service workers must check for browser compatibility. Then, using the "serviceWorkerRegistration" function, the installation starts as a promise [14], where the install and activate "AddEventListener" functions follow a specific procedure to install and activate the service workers. The process followed by developers to set up a service worker is presented in Figure 2.

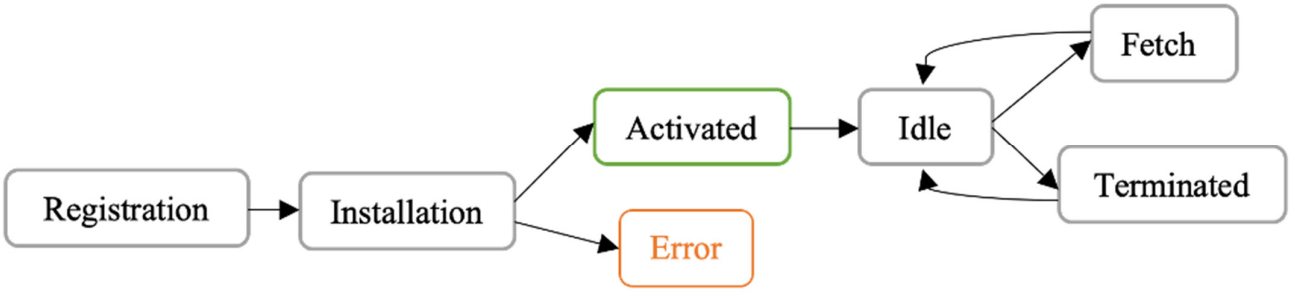

Figure 2. Service worker's lifecycle.

Modern web browsers have adopted the use of service workers, and others support them in their latest versions. Supported web browsers are Chrome, Firefox, Opera, Safari, IE, and Samsung Internet. These browsers also support features such as promises; navigator.serviceWorker; and fetch, install, and activate events. In contrast, background sync is currently only supported in Chrome [15]. According to Google's Web Updates in 2015, background sync is a web API that lets you defer actions until the user has stable connectivity. This is useful for ensuring that whatever the user wants to send is actually sent.

3.2. Caching Storage Application Programming Interface (API) and Offline Functionality

The service worker kernel is equipped with a caching interface; the cache can store static content that users can access without an internet connection. Depending on the caching settings that have been set, the service worker undertakes caching of the content that the user has already visited. If the user's device does not have an internet connection, all cached information is temporarily displayed until the connection is restored [16]. In that way, PWAs achieve the fastest loading of apps, making the apps usable even without an internet connection. The only limitation is the browser's storage limit. The process of creating and retrieving caching content is presented in Figure 3.

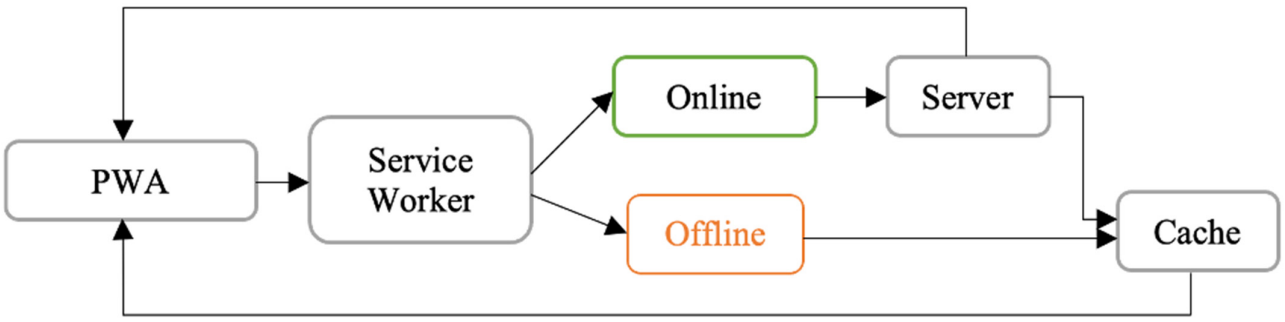

Figure 3. Caching mechanism.

To set up a caching mechanism, an "addEventListener" must be installed. Through the installation, event items are added on the cache's property as an array, and these properties can be used instead of the server's content when there is no internet connection. Moreover, even if an internet connection exists, the service worker sends the cached content back to the app, and only the missing content is requested by the webserver. To keep the cached content up to date, the service worker must regularly update its cached contents by deleting old caches and creating a new one. If the page is dynamic and its content changes frequently, refreshing the dynamic cached elements is possible with periodic JSON fetch calls.

Adopting techniques from native apps, PWAs suggest creating an offline fallback page with a custom service worker. On this page, the user is notified of the lack of internet

connection and, with an automatic reload based on the online event, is informed that a new reconnection attempt will be performed in x seconds [2].

3.3. Web App Manifest

The web app manifest is a JSON file that informs the browser about the PWA and how it should behave when installed on the user's desktop or mobile device [2]. This file contains the PWA's properties in JSON format. The properties required for a web app to be considered as a PWA are presented in Table 1, although many other optional properties exist that define more accurately the web app manifest [2].

Table 1. Web manifest properties.

Property	Description
short_name and name	These properties define the app's name shown under the app icon.
icons	These properties present the app icons in different sizes. For each icon, src, type, and sizes must be included.
start_url	This property describes the browser, which is the starting URL of the app.
background_color	This property sets the background color of the app.
display	This property is the display option. There are four display options: full-screen, standalone, minimal-ui, and browser. Each of these options displays the app differently.
scope	This property is a string that specifies which directories and files the web app manifest affects.
theme_color	This property defines the color of the toolbar.
shortcuts	This property is an array of app shortcut objects whose goal is to provide quick access to key tasks within the app. For each icon, name and URL values must be included.
description	This property is a short description of the app's scope.
screenshots	This property is an array of image objects, representing the app as a screenshot in common usage scenarios. For each image, src, sizes, and type must be defined.

Once the JSON file is created, it should be added as a link to the head of each page. A file validation using Chrome's dev tools is also deemed necessary.

Acting as native apps, PWAs can be added to users' home screens so that the users do not have to navigate to a URL every time they want to use an app [17]. Furthermore, through the launching process, instead of a blank white screen, a splash screen is displayed [5]. Chrome for Android automatically shows a custom splash screen as long as the PWA meets some basic requirements, such as name, background_color, and icons, in its web app manifest [2].

3.4. Push API and Notifications

PWAs have the ability to display re-engaging notifications as defined in the Push API [17]. On desktops, Chrome, Firefox, and Opera all support both the Push API and web notifications. While Safari supports web notifications, it has a custom implementation for push notifications too. Moreover, Edge supports web notifications, but has no Push API support. On the mobile side, iOS has no support for either feature, while Android supports only the Push API [5].

3.5. Application Shell (App Shell) Architecture

An application shell (app shell) refers to the local resources required by a web app to load the skeleton of the user interface (UI) [6]. After native app implementation, all of the views, fonts, and images are uploaded and rendered as a basic app skeleton on the app store. In contrast, PWAs are not uploaded to the native app store but are fetched at runtime when the app is opened [18]. The app shell contains HTML, JavaScript, and CSS files that change infrequently and can be cached so that they can be loaded instantly from the cache on repeat visits. If the app shell has been cached by the service worker, then, on repeat visits, the app shell allows users to rapidly receive meaningful pixels on their screens without a network. UI pieces across different pages, such as headers, toolbars, and footers, are commonly fetched and cached on the app shell. The main purpose of the app shell is to make a PWA's users feel as though they are using a "real" app [2].

The app shell should:

- Load fast;
- Use as little data as possible;
- Use static assets from a local cache;
- Separate content from navigation;
- Retrieve and display page-specific content (HTML, JSON, etc.).

3.6. PWA Critical View

This article identifies and evaluates, through testing, all of the advantages and disadvantages of PWAs that have emerged to date. These advantages and drawbacks are identified and thoroughly analyzed in Appendix A (Table A1).

Based on Table A1, we conclude that PWAs are fully compatible only with Android devices, thus depriving iOS users of many default features. However, PWAs are not responsible for the majority of these limitations; Apple's operating systems have many safety valves to enhance the security of their users, but set limits on innovation. Furthermore, PWAs constitute a new, rapid technology, and dealing with the limitations that arise takes time. Updates are frequent, making PWAs closer to native functions. For this technology to evolve, both developers and operating system must work together with browser companies.

4. Web Accessibility

Web accessibility allows everyone, including people with disabilities, to perceive, understand, navigate, and interact with the internet [19]. The same applies to mobile apps and PWAs. Web accessibility can be defined as making a website navigable and tractable through various user categories, especially users who have disabilities and normally face obstacles and limitations when interacting with the web via electronic devices [20]. Web accessibility is related to the practice of generating web pages accessed by people with all types of abilities and disabilities [21]. Despite technological growth and international regulations, the majority of websites and apps remain inaccessible to certain groups of people. Although accessibility seems to be obsolete, an increasing number of new technologies are relying on its guidelines to create user-accessible applications, such as voice search and PWAs.

4.1. Guidelines and Regulations

The growing number of national and international laws addressing the accessibility of information and communication technologies (ICT), including the web, has resulted in many different approaches in practice [20]. The first accessibility law adopted by the US in 1990 is the Americans with Disabilities Act (ADA). According to the law, websites and apps are deemed to be places of accommodation and have the duty to be accessible; those that are not accessible are considered to discriminate against people with disabilities [22].

There is a wide range of people with disabilities, including visual, auditory, physical, speech, cognitive, language, learning, and neurological disabilities, and combinations thereof [23]. While numerous guidelines and tools have been developed to help improve access to and understanding of websites' content, the most relevant ones are W3C WCAG, ISO, and Section 508 [24].

Although this paper focuses on W3C WCAG guidelines, it is imperative to mention shortly ISO 9241-151:2008 and Section 508 regulations. With the aim of increasing usability, ISO 9241-151:2008—Guidance on World Wide Web UIs provides guidance on the human-centered design of software web UIs [25]. ISO 9241-151 focuses on the design aspects and provides design guidance and recommendations in four major areas, which are [24,26]:

- High-level design decisions and design strategy;
- Content and functionality;
- Navigation, search, and interaction;
- Media design and presentation.

In 1998, US Congress amended the Rehabilitation Act of 1973 to require federal agencies to make their electronic and information technology (EIT) accessible to people with disabilities [27]. Section 508 of the Rehabilitation Act applies to the federal agencies, and it requires e-government websites to be accessible to people with disabilities [24]. Section 508 complies with the requirements of other guidelines and standards for both the US and the European Commission, as well as the W3C WCAG.

W3C Web Content Accessibility Guidelines (WCAG)

The World Wide Web Consortium (W3C) has developed Web Content Accessibility Guidelines (WCAG) to make the web accessible to people with disabilities [28]. Following these guidelines will make content more accessible to a wider range of people with disabilities. Making the web accessible benefits individuals, businesses, and society [23], and the WCAG's goal is to provide a single shared standard for web content accessibility that meets the needs of individuals, organizations, and governments internationally [28].

Four different WCAG versions exist: WCAG 1.0, WCAG 2.0, WCAG 2.1, and WCAG 2.2. Each version extends its predecessor and takes its requirements for granted.

WCAG 1.0, created by W3C on 5 May 1999, is the foundation stone of WCAG regulations. This first version consists of 14 guidelines or general principles of accessible design. Each guideline has a unique statement, a guideline number, and checkpoints. Respectively, the checkpoints have a unique statement, a checkpoint number, and a link to a section of the Techniques Document where implementations and examples of the checkpoint are discussed. To categorize the checkpoints depending on their importance each checkpoint has a priority level assigned by the Working Group based on the checkpoint's impact on accessibility [29]. In this process, there arose three levels of conformance:

- Conformance Level "A": all Priority 1 checkpoints are satisfied;
- Conformance Level "Double-A": all Priority 1 and 2 checkpoints are satisfied;
- Conformance Level "Triple-A": all Priority 1, 2, and 3 checkpoints are satisfied.

Although the conformance level was created to identify the priority of each checkpoint, at the same time, as the level increases, it is difficult for web developers to achieve the corresponding requirements, meaning that A-level checkpoints are easier to achieve than the Triple-A.

WCAG 2.0 came as an update to WCAG 1.0 in 2008. They improved the structure and consistency of the prior guidelines and, at the same time, added additional success criteria. The checkpoints were renamed to success criteria and the priority was assimilated from Level A to AAA. WCAG 2.0 is organized around four design principles that provide the foundation for web accessibility (perceivable, operable, understandable, and robust) [20].

- Level A is the minimum level that defines the foundations of web accessibility [30]. It consists of 12 guidelines, including 25 success criteria.
- Level AA includes all Level A and AA requirements [23]. In addition, it attaches 13 new success criteria to the existing six guidelines. These success criteria provide the basic objectives that authors must achieve to create more accessible content for users with different levels of disability [30].
- Level AAA includes all Level A, AA, and AAA requirements [23]. Moreover, it attaches 23 new success criteria to the existing nine guidelines. These success criteria allow for the evaluation of requirements and needs, such as design specifications, purchasing, regulation, or contractual agreements [30].

WCAG's 2.1 updated version came in June 2018. WCAG 2.1 came as an extension of the 2.0 version, adding five A, seven AA, and five AAA success criteria to the existing guidelines.

The newest WCAG 2.2 was introduced as a working draft in August 2020 and updated in May 2021. WCAG 2.2 extends its predecessor's guidelines, adding the following nine success criteria [31]:

- Accessible Authentication (3.3.7; Level A).
- If an authentication process relies on a cognitive function test, then at least one other method must also be available that does not rely on a cognitive function test [31]. Cognitive deficits usually occur in older people [32], who find it difficult to memorize passwords, make calculations, or even solve a puzzle. Since this success criterion is required for conformance to Level A, there must be an alternative method in case the website uses a cognitive authentication function.
- Redundant Entry (3.3.8; Level A).
- Redundant entry is a success criterion of Level A, which requires auto-populated or user-selected information in fields previously entered by the user. However, there is an exception when re-entering essential information [31].
- Consistent Help (3.2.6; Level A).
- When facing problems completing a task on a website, people with some types of disabilities may not be able to work through the issue without further help. Issues could include difficulty completing a form or finding a document or page that provides the information required to complete a task [31]. Therefore, up-to-date Frequently Asked Questions and human contact details, or a messaging mechanism is required.
- Page Break Navigation (2.4.13; Level A).
- The purpose of this success criterion is to allow people using assistive technology or screen readers to find references to content based on the page break locators found in the default view or printed version of a publication [31].
- Dragging Movements (2.5.7; Level AA).
- All functionality that uses a dragging movement for operation can be operated by a single pointer without dragging, unless dragging is essential [31]. Dragging movements are observed in services such as Google Maps on mobile phones, where more pointers are needed to drag a map.
- Target Size (2.5.8; Minimum; Level AA).
- This success criterion belongs to the AA conformance level and aims to help users with hand tremors and those who have difficulty with fine motor movement to activate interactive areas, such as in-line links and pop-over content accurately. Visible Controls (3.2.7; Level AA).
- The controls needed to progress or complete a process are visible at the time they are needed without requiring pointer hover or keyboard focus, or a mechanism is available to make them persistently visible [31]. Tasks such as on-mouseover user interactions can make it challenging for people with impaired memory and other cognitive and learning disabilities.

- Focus Appearance (2.4.11; Minimum; Level AA).
- The purpose of this success criterion is to ensure that a keyboard focus indicator is clearly visible and discernible following the appropriate minimum area and contrast [31].
- Focus Appearance (2.4.12; Enhanced; Level AAA).
- This success criterion is an extension of the previous one, extending the minimum area, increasing contrast, and excluding obscured elements.

All the updated WCAG's design principles, guidelines, and success criteria are presented in depth in Table 2.

Table 2. WCAG design principles, guidelines, and success criteria.

1. Perceivable	2. Operable	3. Understandable	4. Robust
1.1 Text Alternatives	2.1 Keyboard Accessible	3.1 Readable	4.1 Compatible
(A) 1.1.1 Non-text Content	(A) 2.1.1 Keyboard	(A) 3.1.1 Language of Page	(A) 4.1.1 Parsing
1.2 Time-based Media	(A) 2.1.2 No Keyboard Trap	(AA) 3.1.2 Language of Parts	(A) 4.1.2 Name, Role, Value
(A) 1.2.1 Audio-only and Video-only (Prerecorded)	(AAA) 2.1.3 Keyboard (No Exception)	(AAA) 3.1.3 Unusual Words	(AA) 4.1.3 Status Messages
(A) 1.2.2 Captions (Prerecorded)	(A) 2.1.4 Character Key Shortcuts	(AAA) 3.1.4 Abbreviations	
(A) 1.2.3 Audio Description or Media Alternative (Prerecorded)	2.2 Enough Time	(AAA) 3.1.5 Reading Level	
(AA) 1.2.4 Captions (Live)	(A) 2.2.1 Timing Adjustable	(AAA) 3.1.6 Pronunciation	
(AA) 1.2.5 Audio Description (Prerecorded)	(A) 2.2.2 Pause, Stop, Hide	3.2 Predictable	
(AAA) 1.2.6 Sign Language (Prerecorded)	(AAA) 2.2.3 No Timing	(A) 3.2.1 On Focus	
(AAA) 1.2.7 Extended Audio Description (Prerecorded)	(AAA) 2.2.4 Interruptions	(A) 3.2.2 On Input	
(AAA) 1.2.8 Media Alternative (Prerecorded)	(AAA) 2.2.5 Re-authenticating	(AA) 3.2.3 Consistent Navigation	
(AAA) 1.2.9 Audio-only (Live)	(AAA) 2.2.6 Timeouts	(AA) 3.2.4 Consistent Identification	
1.3 Adaptable	2.3 Seizures and Physical Reactions	(AAA) 3.2.5 Change on Request	
(A) 1.3.1 Info and Relationships	(A) 2.3.1 Three Flashes or Below Threshold	(A) 3.2.6 Consistent Help (NEW)	
(A) 1.3.2 Meaningful Sequence	(AAA) 2.3.2 Three Flashes	(AA) 3.2.7 Visible Controls (NEW)	
(A) 1.3.3 Sensory Characteristics	(AAA) 2.3.3 Animation from Interactions	3.3 Input Assistance	
(AA) 1.3.4 Orientation	2.4 Navigable	(A) 3.3.1 Error Identification	
(AA) 1.3.5 Identify Input Purpose	(A) 2.4.1 Bypass Blocks	(A) 3.3.2 Labels or Instructions	
(AAA) 1.3.6 Identify Purpose	(A) 2.4.2 Page Titled	(AA) 3.3.3 Error Suggestion	
1.4 Distinguishable	(A) 2.4.3 Focus Order	(AA) 3.3.4 Error Prevention (Legal, Financial, Data)	
(A) 1.4.1 Use of Color	(A) 2.4.4 Link Purpose (In Context)	(AAA) 3.3.5 Help	

(A) 1.4.2 Audio Control	(AA) 2.4.5 Multiple Ways	(AAA) 3.3.6 Error Prevention (All)
(AA) 1.4.3 Contrast (Minimum)	(AA) 2.4.6 Headings and Labels	(A) 3.3.7 Accessible Authentication (NEW)
(AA) 1.4.4 Resize text	(A) 2.4.7 Focus Visible	(A) 3.3.8 Redundant entry (NEW)
(AA) 1.4.5 Images of Text	(AAA) 2.4.8 Location	
(AAA) 1.4.6 Contrast (Enhanced)	(AAA) 2.4.9 Link Purpose (Link Only)	
(AAA) 1.4.7 Low or No Background Audio	(AAA) 2.4.10 Section Headings	
(AAA) 1.4.8 Visual Presentation	(AA) 2.4.11 Focus Appearance (Minimum) [NEW]	
(AAA) 1.4.9 Images of Text (No Exception)	(AAA) 2.4.12 Focus Appearance (Enhanced) (NEW)	
(AA) 1.4.10 Reflow	(A) 2.4.13 Page Break Navigation (NEW)	
(AA) 1.4.11 Non-text Contrast	2.5 Input Modalities	
(AA) 1.4.12 Text Spacing	(A) 2.5.1 Pointer Gestures	
(AA) 1.4.13 Content on Hover or Focus	(A) 2.5.2 Pointer Cancellation	
	(A) 2.5.3 Label in Name	
	(A) 2.5.4 Motion Actuation	
	(AAA) 2.5.5 Target Size (Enhanced)	
	(AAA) 2.5.6 Concurrent Input Mechanisms	
	(AA) 2.5.7 Dragging Movements (NEW)	
	(AA) 2.5.8 Target Size (Minimum) (NEW)	

Depth 0: design principles, Depth 1: guidelines, Depth 2: success criteria. (NEW) introduced in WCAG 2.2.

4.2. Guidelines and Regulations

Prior research has queried diverse approaches to address accessibility guidelines. This article not only identifies those guidelines, but also presents the problems that web developers face when trying to follow them, as well as the limitations of online tools that check for accessibility.

The success of web-based applications depends on how well they are perceived by the end users [28]. Making websites and apps accessible is mandatory. Thus, for example, in every new technology that Google presents, it emphasizes the need to create human-centered web applications. Furthermore, web developers, motivated by the best search engine rankings, follow some of the accessibility guidelines. However, the majority of web applications that are created are not fully accessible, thereby creating accessibility issues for people with disabilities. While some laws oblige website owners in both the public and private sectors to create websites that are accessible to everyone, these laws do not apply unless a person with disabilities complains about noncompliance.

4.2.1. Limitations of Accessibility Guidelines

Previous research has shown that, even though accessibility guidelines lay the foundations for a more accessible web, they also have some limitations. Evidence suggests that compliance with accessibility standards does not always guarantee a satisfying UX on the web [33]. Furthermore, guidelines are difficult to evaluate, even if a combination of human and machine audits are used [34]. In a study with intellectual disabilities participants, it was observed that, even if a website follows the W3C WCAG, users' satisfaction is not 100% [35].

4.2.2. Web Developers' Limitations

From a developer's point of view, dealing with accessibility is a demanding task that requires much time following each of the guidelines and success criteria. To implement real accessibility, every web development agency should hire a special team that will test the accessibility of the application, both with online tools and manually. A lack of awareness, education, and motivation leads developers to create inaccessible websites.

4.2.3. Limitations of Machine Auditing and Plugins

Online tools (machine audits) and plugins have been developed to help website owners and web developers to create more accessible websites. Using online evaluation tools, such as WAVE (by WebAim), the W3 validator (by the W3C), and Lighthouse (by Google), every website can be checked against the WCAG and success criteria. These tools scan a website's source code and highlight the corrections that must be made to make the website more accessible. Some of them use visual composers to highlight areas where errors occur, while others refer to the appropriate success criterion to understand exactly what needs to be corrected. Although machine evaluation tools offer a useful pathway, each of them presents different website errors compared to the others.

In addition, some plugins for popular open-source platforms promise full accessibility with a single click. These plugins make some changes to a website's source code without necessarily covering any guidelines or success criteria.

Based on the information presented in this section, we conclude that the need for accessibility is increasing; however, an increasing number of websites—even government websites, online libraries, and university websites—do not follow the guidelines. On the one hand, even though accessibility guidelines cannot cover every disability issue, they are improved day by day. On the other hand, web developers have a moral obligation to provide a website that is as accessible as possible. To achieve this, both manual and machine auditing must be performed.

5. Results

In Section 3, PWAs were extensively reviewed, and we conclude that the adoption of this new technology is accompanied by both positive and negative aspects. Then, in Section 4, accessibility guidelines that have been created to date were analyzed in depth, along with the limitations they face. In this section, we combine and evaluate the efficiency of PWAs in relation to accessibility.

Google's website states that PWAs are fully accessible and, on the same page, web developers are encouraged to perform manual and machine auditing to achieve accessibility [36]. The term "fully accessible" is used by a growing number of technologies to indicate that the core of that technology is accessible. Accessibility, however, cannot be applied to the core, but to areas to which people with disabilities have access (i.e., the front end). As a result, PWAs could be described as a technology that promotes accessibility.

The ultimate goal of this research is to compare and evaluate the PWA as a technology against other (non-PWA) web technologies regarding their compatibility with the existing accessibility guidelines in order to conclude whether the former is more accessible than the latter or not. In this stage, a sample of 20 PWA and 20 non-PWA websites are

selected as depicted in the first column of Table 3 and Table 4, respectively. Our dataset consists of two groups of websites. The first group (first column in Table 3) contains 20 websites that were built based on the PWA architecture. These websites are also listed as case studies by Google [37]. The second group (first column Table 4) contains 20 conventional non-PWA websites. To achieve uniformity in our results, for each PWA website, a non-PWA website was found in the same niche. The non-PWA websites examined in this paper are top-listed in Google's search results based on their niche.

The dataset of 20 PWAs and 20 conventional non-PWA websites was manually fully tested against 10 accessibility evaluation tools—auditing tools. For each website of the dataset, a manual procedure was performed, taking the website's URL and placing it to the auditing tool's input area. Auditing tools, in turn, accept the URL, sending a bot to the given website to perform accessibility inspections. The results returned by the auditing tools for each website were documented in excel sheets to be easily manageable by the program that will perform later the descriptive analysis. Each tool has its own metrics for measuring and displaying the results—total errors. Two of them, Google Lighthouse and PowerMapper, show the errors as a percentage of total checks. The remaining eight tools show the results as the number of the errors found. For the sake of simplicity, a unique ID key has been assigned to each auditing tool presented in the following tables: (T1) Lighthouse by Google, (T2) Wave by WebAim, (T3) Web Accessibility by Level Access, (T4) Power Mapper, (T5) CSS Validation Service by W3, (T6) Nu Html Checker by W3, (T7) AXE DevTools, (T8) Accessibility Insights Chrome, (T9) Equal Access by IBM, and (T10) Site Improve. The evaluation results for PWA websites are presented in Table 3 and for non-PWA in Table 4, respectively.

Table 3. Accessibility compliance errors for PWA websites.

Dataset/Auditing Tools	T1 (%)	T2	T3	T4 (%)	T5	T6	T7	T8	T9	T10	Niche
Nikkei.com	18	102	88	15	5	1000	450	56	55	115	News Blog
George.com	11	2	10	17	30	63	49	10	18	62	Clothing Brand
Ele.me	12	1	2	19	0	5	23	7	10	30	Food Ordering
BookMyShow.com	4	6	127	26	11	851	252	233	242	32	Ticketing
Forbes.com	11	6	38	26	12	69	33	12	7	30	News Blog
Infobae.com	3	1	6	17	11	21	123	7	257	214	Digital-only news
Lancome-usa.com	8	1	13	14	72	154	166	0	85	190	Luxury cosmetics
Makemytrip.com	14	4	1	23	5	7	161	22	22	199	Travel Booking
Mynet.com	8	3	8	17	0	54	353	23	290	257	News Blog
Olacabs.com	22	166	1	33	1	50	150	83	163	139	Car Rental
Olx.in	33	13	1	30	5	65	50	40	78	52	Free Classifieds
Twitter Lite	26	11	2	0	1	24	53	38	34	40	Social Network
Wego.com	11	2	19	12	0	3	386	90	162	190	Travel Booking
Housing.com	41	12	9	12	4	26	117	26	22	54	Property Website
Alibaba.com	51	153	171	4	23	2	286	185	270	371	Marketplace
Weather.com	12	2	6	26	2	129	21	4	9	162	Weather Platform
Carnival.com	25	9	19	35	54	185	130	28	39	80	Travel Booking
Washingtonpost.com	13	39	5	0	17	254	65	47	58	64	News Blog
Aliexpress.com	58	26	46	2	0	16	211	83	215	183	Marketplace
Extra.com	25	64	143	9	15	129	152	85	50	113	Marketplace

Table 4. Accessibility compliance errors for **non**-PWA websites.

Dataset/Auditing Tools	T1 (%)	T2	T3	T4 (%)	T5	T6	T7	T8	T9	T10	Niche
Bbc.com	7	3	4	2	59	47	127	10	17	82	News Blog
Nike.com	19	14	139	31	70	224	109	31	62	116	Clothing Brand
Doordash.com	22	3	4	33	0	6	14	1	10	19	Food Ordering
Booking.com	12	2	41	14	59	15	729	5	198	99	Ticketing
Dw.com	9	4	15	19	16	25	192	36	288	308	News Blog
Usatoday.com	18	9	33	22	12	99	483	126	175	44	News Blog
Harrods.com	11	1	0	33	10	739	61	16	28	46	Luxury cosmetics
Hopper.com	89	4	8	8	0	32	131	48	57	131	Travel Website
Nbcnews.com	43	30	64	22	454	33	472	29	72	106	News Blog
Americacarrental.com	28	27	19	21	22	79	41	15	25	19	Car Rental
Gumtree.com	5	6	64	17	14	11	237	47	159	178	Free Classifieds
Facebook.com	11	3	1	0	4	22	79	27	70	92	Social Network
Expedia.com	9	3	0	22	43	108	30	7	62	57	Travel Booking
Zillow.com	8	2	2	24	11	82	20	1	7	14	Property Website
Snapdeal.com	35	227	87	7	27	309	566	431	683	276	Marketplace
Weather-forecast.com	13	4	5	5	11	21	62	7	149	125	Weather Platform
Royalcaribbean.com	21	13	3	12	113	397	133	22	91	218	Travel Booking
Dailymail.co.uk	28	50	145	20	45	348	2134	217	1067	530	Newspaper
Ebay.com	8	26	27	12	23	319	142	43	80	115	Marketplace
Amazon.com	12	39	79	14	38	199	191	54	159	168	Marketplace

To analyze our dataset to achieve valuable results, Jupyter software has been used. Jupyter Notebook is the most widely used system for interactive literate programming for data science purposes. It was designed to make data analysis easier to document, share, and reproduce [38]. In our analysis, for simplicity purposes, we have used the terms "older technology" for non-PWA websites and "new technology" for PWA websites, respectively. Jupyter software, although based on Python language, can support many other languages, such as Java and R. In our analysis, we have used Python 3 programming language, which is highly supported by Jupyter's core.

Figure 4 presents the Python libraries we have used for our data analysis and Figure 5 presents the way we imported our dataset into Pandas Data Frame.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('ggplot')
from matplotlib import rcParams
rcParams['figure.figsize'] = 15, 5

import seaborn as sns; sns.set_theme()
```

Figure 4. Import required in Python Libraries.

```
old_tech = pd.read_excel('OLD-TECHNOLOGY.xlsx')
new_tech = pd.read_excel('NEW-TECHNOLOGY.xlsx')
new_tech.head()
```

Figure 5. Read data into Pandas Data Frame.

The results after the descriptive analysis, both for new and older technology, are presented in Table 5 and Table 6, respectively.

Table 5. Descriptive analysis for new technology (PWA)—new_tech.describe().

	T1 (%)	T2	T3	T4 (%)	T5	T6	T7	T8	T9	T10
count	20	20	20	20	20	20	20	20	20	20
mean	20.30	31.15	35.75	16.85	13.40	155.35	161.55	53.95	104.30	128.85
std	15.21	50.72	52.72	10.59	19.10	273.16	126.23	60.97	99.86	91.38
min	3	1	1	0	0	2	21	0	7	30
25%	11	2	4.25	11.25	1	19.75	52.25	11.50	22	53.50
50%	13.50	7.50	9.50	17.00	5	58.50	140	33	56.50	114
75%	25.25	29.25	40	26	15.50	135.25	221.25	83	176	190
max	58	166	171	35	72	1000	450	233	290	371

Table 6. Descriptive analysis for older technology (non-PWA)—old_tech.describe().

	T1 (%)	T2	T3	T4 (%)	T5	T6	T7	T8	T9	T10
count	20	20	20	20	20	20	20	20	20	20
mean	20.40	23.50	37	16.90	51.55	155.75	297.65	58.65	172.95	137.15
std	19.07	49.94	45.42	9.65	98.84	187.42	476.80	101.11	258.52	122.92
min	5	1	0	0	0	6	14	1	7	14
25%	9	3	3.75	11	11	24.25	61.75	9.25	49.75	54.25
50%	12.50	5	17	18	22.50	80.50	132	28	76	110.50
75%	23.50	26.25	64	22	48.50	245.25	295.75	47.25	163	170.50
max	89	227	145	33	454	739	2134	431	1067	530

The x-axis scale in both Tables 5 and 6 presents the auditing tools undertaken to perform the accessibility checks on sample websites, while the y-axis scale presents the metrics used by the tool to perform the descriptive analysis. To clarify the results shown in Tables 5 and 6 we provide some additional information bellow:

- The count() function used to count the sum of the data.
- The mean() function used to count the average value.
- The std() function used to compute the standard deviation along the specified axis.
- The 25%, 50%, and 75% are three quartile values (Q1, Q2, and Q3, respectively). The second quartile (Q2) is the median of the whole data, the first quartile (Q1) is the median of the upper half of the data, while the third quartile (Q3) is the median of the lower half of the data.
- The max() and min() functions used to present the max and min values of errors per auditing tool.

A way to start the descriptive analysis is by looking at what kind of ranges each tool has and how much they vary around their average values (mean). We can observe from the above plots that CSS Validation Service by W3 (T5) error value reduced to a significant level in new technology. The average error value in new technology is 13.40, while it is 51.55 in older technology. Similarly, in new technology, the maximum axe DevTool (T7) error value is 450, while it is 2134 in older technology, showing that PWAs are operating significantly better and that error rates in websites have been lowered to a larger extent. Both Tables 5 and 6 demonstrate that conventional non-PWA websites retain more errors than PWA websites in checks performed by accessibility auditing tools. As a result, the PWA websites of the dataset follow the accessibility guidelines more strictly than conventional non-PWA websites. Examining the mean values in Tables 5 and 6, we notice that nine out of ten auditing tools return less errors on average for PWAs than non-PWA websites, concluding that websites that use the PWA technology are more compatible with the existing accessibility guidelines.

To further support our findings, we graphically present the results for each tool into plots in Figures 6, 7 and 8.

```
fig,axs=plt.subplots(nrows=2)
fig.set_size_inches(15,10)

sns.barplot(y="Equal Access by IBM", x="Dataset/Auditing Tools", data=new_tech,ax=axs[0], dodge=False)
sns.barplot(y="Equal Access by IBM", x="Dataset/Auditing Tools", data=old_tech,ax=axs[1], dodge=False)

axs[0].set_title('Figure 1: Access by IBM in New-Technology')
axs[1].set_title('Figure 2: Access by IBM in Older-Technology')
axs[0].tick_params(axis="x", labelsize=12, rotation = 45)
axs[1].tick_params(axis="x", labelsize=12, rotation = 45)
plt.tight_layout()
plt.show()
```

Figure 6. Presenting outcomes into plots—Equal Access by IBM (T9).

Figure 7. Equal Access by IBM (T9) errors for new technology (PWA) websites.

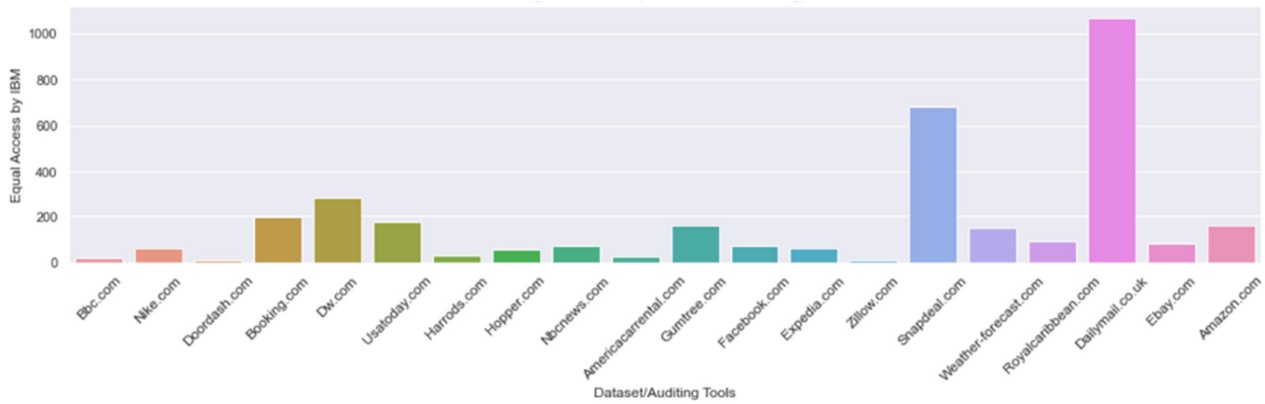

Figure 8. Equal Access by IBM (T9) errors for Older technology (non-PWA) websites.

Figures 7 and 8 compare errors/bugs in the IBM Equal Access toolkit (T9). The y-axis scale shows that error values in modern technologies are substantially lower than in older technologies, as calculated by several auditing tools.

The last column in both Tables 3 and 4 presents the niche—industry for each website checked, grouping for extension of our datasets. The calculation of errors per industry is depicted in Figure 9, while the corresponding plot is presented in Figure 10, based on Axe DevTools (T7).

```
# Using plotly.express
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots


fig = make_subplots(rows=1, cols=2,subplot_titles=("New_technology","Older_technology"))

fig.append_trace(go.Bar(
   x=new_tech['Niche'],
    y=new_tech["axe DevTools"],
), row=1, col=1)

fig.append_trace(go.Bar(
     x=old_tech['Niche'],
    y=old_tech["axe DevTools"],
), row=1, col=2)



fig.update_layout(height=600, width=1000, title_text="Axe DevTools")
fig.show()
```

Figure 9. PWA vs. non-PWA industry plots—Axe DevTools (T7).

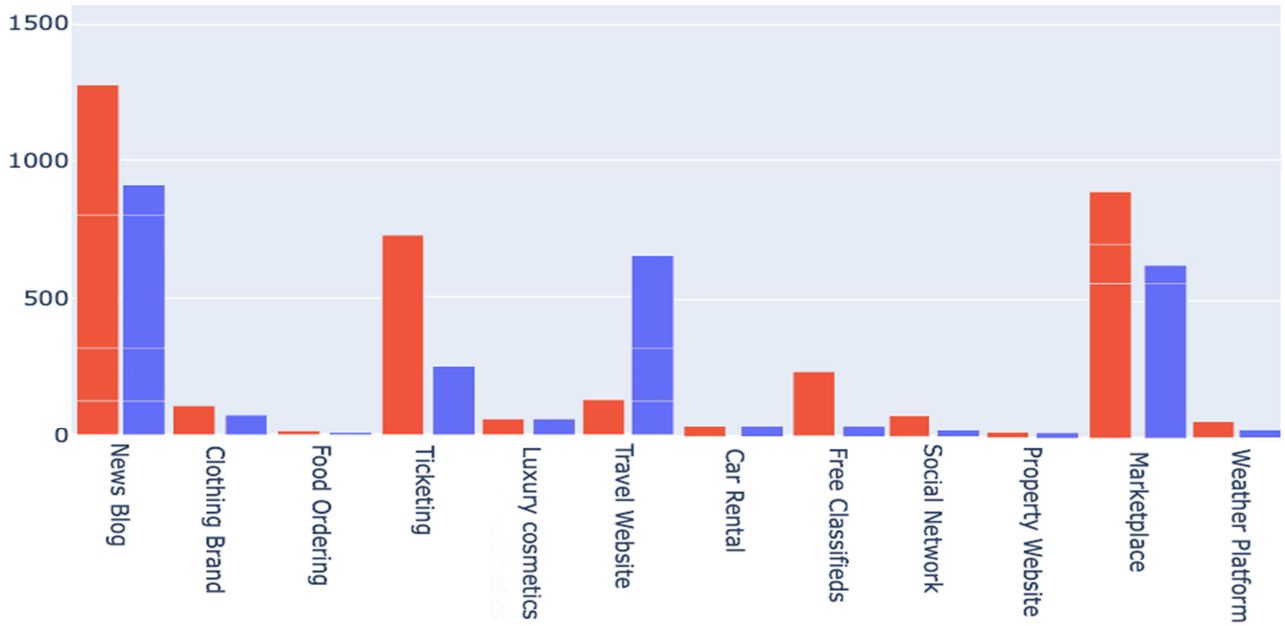

Figure 10. Presenting outcomes per industry into plots (red: older technology, blue: new technology)—Axe DevTools (T7).

The plot in Figure 10 illustrates that the Axes Devtool (T7) error rate is reduced in PWA when compared to the corresponding non-PWA performance. As a result, we can conclude that the general performance of websites in new technology is far better than the older ones with regards to accessibility conformance to the respective guidelines. However, many of the websites still need to be improved. It is crucial to mention that the websites that theoretically have the most traffic—visitors, such as news blogs and marketplaces, are the ones that mainly ignore the accessibility guidelines, either on PWA or non-PWA websites.

To conclude with, the results confirm that each auditing tool performs its own measurements and displays its own errors based on its assessment. Measurements in Google

Lighthouse (T1) indicate that PWAs emphasize performance and search engine optimization. Although, the majority of PWAs present less accessibility errors than the non-PWAs, and only few of the former ignore in a great scale the accessibility guidelines. This observation confirms that PWAs are not fully accessible by default and that web developers must make an effort to conduct both machine and manual audits to achieve actual accessibility.

To expand our research, five ready-made, premium PWA templates from ThemeForest were selected and analyzed using the same accessibility tools. ThemeForest is a ready-made template repository where web designers and developers sell their templates. These templates state on their purchase page that they fully adopt PWA techniques, including accessibility. However, the results from the measurements revealed that none of them follow the accessibility guidelines. The sales of these five templates have reached 2000, meaning that 2000 new websites can be added to the list of non-accessible websites on the World Wide Web. Template repositories are the only place where most website owners buy their templates. If these sources ignore the need for accessibility, then the majority of websites will ignore this need as well.

6. Conclusions

Web accessibility aims to provide usable web information and services to as many people as possible [39]. A growing number of technologies are characterized as accessible without actually following the accessibility guidelines. An essential need exists to adapt accessibility across to the World Wide Web, covering each platform and each technology. In this article, we conducted an in-depth review of both PWAs and web accessibility as technologies, and we combined them to evaluate PWAs' effectiveness as regards the accessibility they offer. Following specific methodology, a representative sample of 20 PWAs and 20 conventional non-PWA websites in corresponding niches were collected and analyzed against 10 accessibility evaluation tools. The results have shown a great lead on accessibility guidelines' conformance for the websites that have adopted the PWA architecture. We conclude that PWAs constitute a new technology with many limitations, which it exceeds daily. Despite these limitations, PWAs offer early adopters a lead in performance, search engine optimization, and accessibility. Web developers' awareness is key to achieving the upgrade of the World Wide Web to a place where every user, regardless of his/her ability, can have equal and trouble-free access. PWAs as a technology, accessibility guidelines, and accessibility auditing tools are living organisms that evolve according to the needs of people with disabilities. Human-centered future research could create tools that will not only check websites based on a list of accessibility guidelines, but incorporate suggestions from the individuals who face a problem highlighting the website's area that is difficult for them to read, understand, or access.

Author Contributions: Conceptualization, K.I.R. and N.D.T.; methodology, K.I.R. and N.D.T.; formal analysis, K.I.R. and N.D.T.; investigation, K.I.R. and N.D.T.; resources, K.I.R.; data curation, K.I.R.; writing—original draft preparation, K.I.R. and N.D.T.; writing—review and editing, K.I.R. and N.D.T.; visualization, K.I.R.; supervision, N.D.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. Advantages and disadvantages of progressive web apps (PWAs).

Advantages	Disadvantages
PWAs are cross-platform solutions. They are created once for all platforms.	PWAs cannot be deployed to the App Store or Google Play. Nonetheless, third-party technologies can convert PWAs into native apps using a protocol based on Custom Tabs.
PWAs allow for quick installations without waiting times.	PWAs are not fully supported by iOS and OsX devices. Many features, such as push notifications for iOS, cannot be achieved without a developer registering an app in the Apple developer portal.
The cost to develop a PWA is much lower than a native or hybrid app.	The add-to-home-screen feature is not available for iOS. Users must manually add the app to the home screen through settings options.
The cost to maintain a PWA is much lower than a native or hybrid app.	PWAs cannot access native iOS components, such as Face ID and Bluetooth.
PWAs are installable. Installing a PWA allows it to look, feel, and behave like all other installed apps.	PWAs are not allowed to access Apple's iBeacons, depriving them of using phone battery and altimeter features.
PWAs can be updated on the fly without a user's interaction.	PWAs have no access to the iOS local filesystem, contact book, and current location.
PWAs provide a custom offline page.	Background sync, which is a core web API, is only supported by the Chrome browser.
PWAs are reliable, fast, and dependable regardless of the network speed.	Web developers must calculate the caching limits for each browser to offer a decent offline caching experience.
Chromium-based PWAs can access the hardware features on Android devices in the same way as native mobile applications.	Face and Speech Recognition is only available through third-party APIs.
PWAs are equally usable with a mouse, a keyboard, a stylus, or touch [2].	Only in-app purchases can be used, not payments for each app download.
	PWAs are incompatible with old devices' obsolete browsers.
Well-known platforms, such as WordPress and Magneto, have already implemented plugins and templates to easily develop PWAs.	PWAs escape the app approval process, and low-quality web apps will eventually emerge.
A PWA can be used as a light version of an app for devices with fewer resources.	Ratings, reviews, and responses on app stores are indispensable in helping users to select the most appropriate app depending on their needs. Unfortunately, PWAs are not a part of app stores.
PWAs can be accessible by meeting WCAG standards.	The mobile-first approach that PWAs often promote will end with mobile-only apps. PWAs' templates are not as desktop-friendly.
PWAs can interact flawlessly with powerful APIs.	Using more internet resources compared with native apps, PWAs consume more battery.

A PWA can be properly indexed on search engines, promoting itself using search engine optimization techniques.	As web applications, PWAs are more likely to be hacked.
PWAs can expand the number of returning visitors, increase conversion rates and engagement, and reduce data usage. Google's case studies have shown that AliExpress increases the conversion rate of new users by 104%, with new PWAs and the Twitter Lite PWA significantly increasing pages per session by 65%, and exhibiting a 75% increase in Tweets sent and a 20% decrease in bounce rate [37].	PWA developers use pop-ups to alert users that can add the app to their home screen. However, some pop-up notifications may be blocked by modern browsers.

References

1. Tandel, S.S.; Jamadar, A. Impact of Progressive Web Apps on Web App Development. *Int. J. Innov. Res. Sci. Eng. Technol.* **2018**, *7*, 9439–9444. https://doi.org/10.15680/IJIRSET.2018.0709021.
2. Progressive Web Apps. Available online: https://web.dev/progressive-web-apps/ (accessed on 11 April 2022).
3. Anderson, S.; Bohman, R.P.; Burmeister, K.O.; Sampson-Wild, G. User Needs and e-Government Accessibility: The Future Impact of WCAG 2.0. In *User-Centered Interaction Paradigms for Universal Access in the Information Society*; Springer: Berlin/Heidelberg, Germany, 2004. https://doi.org/10.1007/978-3-540-30111-0_25.
4. Sheppard, D. *Beginning Progressive Web App Development. Creating a Native App Experience on the Web*; Apress: Berkeley, CA, USA, 2017. https://doi.org/10.1007/978-1-4842-3090-9.
5. Sheppard, D. Introduction to Progressive Web Apps. In *Beginning Progressive Web App Development*; Apress: Berkeley, CA, USA, 2017. https://doi.org/10.1007/978-1-4842-3090-9_1.
6. Rojas, C. Making Your First Progressive Web App. In *Building Progressive Web Applications with Vue.js*; Apress: Berkeley, CA, USA, 2020. https://doi.org/10.1007/978-1-4842-5334-2_1.
7. Shimray, R.S.; Ramaiah, C. Use of Internet through Mobile Devices: A Survey. *SRELS J. Inf. Manag.* **2019**, *56*, 100–105. https://doi.org/10.17821/srels/2019/v56i2/141631.
8. Mobile Internet & Apps. Available online: https://www.statista.com/markets/424/topic/538/mobile-internet-apps/#overview (accessed on 11 April 2022).
9. Yeh, C.; Chen, T.; Chen, M. A Progressive Web App for Evaluating Occupation Indicator. *J. Internet Technol.* **2019**, *20*, 2217–2223.
10. Mhaske, A.; Bhattad, A.; Khamkar, P.; More, R. Progressive Web App for Educational System. *Int. Res. J. Eng. Technol.* **2018**, *5*, 310–312.
11. Malavolta, I.; Procaccianti, G.; Noorland, P.; Vukmirovic, P. Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps. In Proceedings of the IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Buenos Aires, Argentina, 22–23 May 2017. https://doi.org/10.1109/MOBILESoft.2017.7.
12. Madsen, M.; Lhoták, O.; Tip, F. A model for reasoning about JavaScript promises. In Proceedings of the ACM on Programming Languages, OOPSLA, Vancouver, Canada, 12 October 2017. https://doi.org/10.1145/3133910.
13. Chinprutthiwong, P.; Vardhan, R.; Yang, G.; Gu, G. Security Study of Service Worker Cross-Site Scripting. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 8 December 2020. https://doi.org/10.1145/3427228.3427290.
14. ServiceWorkerContainer.Register. Available online: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorker Container/register (accessed on 11 April 2022).
15. Is Service WORKER ready? Available online: https://jakearchibald.github.io/isserviceworkerready/ (accessed on 11 April 2022).
16. Gambhir, A.; Raj, G. Analysis of Cache in Service Worker and Performance Scoring of Progressive Web Application. In Proceedings of the International Conference on Advances in Computing and Communication Engineering (ICACCE), Paris, France, 22–23 June 2018.. https://doi.org/10.1109/ICACCE.2018.8441715.
17. Adetunji, O.; Ajaegbu, C.; Otuneme, N.; Omotosho, J.O. Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development. *Am. Sci. Res. J. Eng. Technol. Sci.* **2020**, *68*, 85–99.

18. What Is an Application Shell? Available online: https://developers.google.com/web/ilt/pwa/introduction-to-progressive-web-app-architectures#what (accessed on 11 April 2022).
19. European Commission Web Accessibility Policy. Available online: https://ec.europa.eu/digital-single-market/en/web-accessibility (accessed on 11 April 2022).
20. Abuaddous, Y.H.; Jali, Z.M.; Basir, N. Web Accessibility Challenges. *Int. J. Adv. Comput. Sci. Appl.* **2016**, *7*, 172–181. https://doi.org/10.14569/IJACSA.2016.071023.
21. Hilera, R.J.; Fernandez-Sanz, L.; Misra, S. Present and future of web content accessibility: An analysis. *Teh. Vjesn.* **2013**, *20*, 35–42.
22. Americans with Disabilities Act (ADA). Available online: https://adata.org/learn-about-ada (accessed on 11 April 2022).
23. Kirkpatrick, A.; Connor, O.J.; Campbell, A.; Cooper, M. Web Content Accessibility Guidelines (WCAG) 2.1. World Wide Web Consortium. Available online: https://www.w3.org/TR/WCAG21/ (accessed on 11 April 2022).
24. Sohaib, O.; Hussain, W.; Badini, K.M. User experience (UX) and the Web accessibility standards. *IJCSI Int. J. Comput. Sci. Issues* **2011**, *82*, 584–587.
25. *ISO 9241-151:2008*; Ergonomics of Human-System Interaction. ISO: Geneva, Switzerland, 2008. Available online: https://www.iso.org/standard/37031.html (accessed on 11 April 2022).
26. Çağıltay, K.; Alacam, O.; Ocak, N.; Erdal, F. Developing ISO 9241-151 Product Certification Process: Challenges. In *International Conference of Design, User Experience, and Usability*; Springer: Berlin/Heidelberg, Germany, 2013. https://doi.org/10.1007/978-3-642-39253-5_36.
27. IT Accessibility Laws and Policies. Available online: https://www.section508.gov/manage/laws-and-policies (accessed on 11 April 2022).
28. Campoverde-Molina, M.; Luján-Mora, S.; García, V.L. Empirical Studies on Web Accessibility of Educational Websites: A Systematic Literature Review. *IEEE Access* **2020**, *8*, 91676–91700. https://doi.org/10.1109/ACCESS.2020.2994288.
29. Web Content Accessibility Guidelines 1.0. Available online: https://www.w3.org/TR/WAI-WEBCONTENT/ (accessed on 11 April 2022).
30. Rodríguez, G.; Pérez, J.; Cueva, S.; Torres, R. A framework for improving web accessibility and usability of Open Course Ware sites. *Comput. Educ.* **2017**, *109*, 197–215. https://doi.org/10.1016/j.compedu.2017.02.013.
31. WCAG 2.2. Available online: https://www.w3.org/WAI/WCAG22/Understanding/ (accessed on 11 April 2022).
32. McEwen, S.B.; Sapolsky, M.R. Stress and cognitive function. *Curr. Opin. Neurobiol.* **1995**, *5*, 205–216. https://doi.org/10.1016/0959-4388(95)80028-X.
33. Aizpurua, A.; Arrue, M.; Vigo, M. Prejudices, memories, expectations and confidence influence experienced accessibility on the Web. *Comput. Hum. Behav.* **2015**, *51*, 152–160. https://doi.org/10.1016/j.chb.2015.04.035.
34. Brajnik, G.; Yesilada, Y.; Harper, S. Is accessibility conformance an elusive property? A study of validity and reliability of WCAG 2.0. *ACM Trans. Access. Comput.* **2012**, *4*, 8–28. https://doi.org/10.1145/2141943. 2141946.
35. Karreman, J.; Van Der Geest, T.; Buursink, E. Accessible Website Content Guidelines for Users with Intellectual Disabilities. *J. Appl. Res. Intellect. Disabil.* **2007**, *20*, 510–518. https://doi.org/10.1111/j.1468-3148.2006.00353.x.
36. What Makes a Good Progressive Web App? Available online: https://web.dev/pwa-checklist/ (accessed on 11 April 2022).
37. PWA Case Studies. Available online: https://developers.google.com/web/showcase (accessed on 11 April 2022).
38. Pimentel, J.F.; Murta, L.; Braganholo, V.; Freire, J. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 25–31 May 2019. https://doi.org/10.1109/MSR.2019.00077.
39. Vollenwyder, B.; Iten, H.G.; Brühlmann, F.; Opwis, K.; Mekler, D.E. Salient beliefs influencing the intention to consider Web Accessibility. *Comput. Hum. Behav.* **2019**, *92*, 352–360. https://doi.org/10.1016/j.chb.2018.11.016.