data

MDPI

Data Descriptor

The Tabular Accessibility Dataset: A Benchmark for LLM-Based Web Accessibility Auditing

Manuel Andruccioli, Barry Bassi, Giovanni Delnevo * and Paola Salomoni

Department of Computer Science and Engineering, University of Bologna, 47522 Cesena, Italy; manuel.andruccioli@unibo.it (M.A.); barry.bassi@unibo.it (B.B.); paola.salomoni@unibo.it (P.S.)
* Correspondence: giovanni.delnevo@unibo.it

Abstract

This dataset was developed to support research at the intersection of web accessibility and Artificial Intelligence, with a focus on evaluating how Large Language Models (LLMs) can detect and remediate accessibility issues in source code. It consists of code examples written in PHP, Angular, React, and Vue.js, organized into accessible and non-accessible versions of tabular components. A substantial portion of the dataset was collected from student-developed Vue components, implemented using both the Options and Composition APIs. The dataset is structured to enable both a static analysis of source code and a dynamic analysis of rendered outputs, supporting a range of accessibility research tasks. All files are in plain text and adhere to the FAIR principles, with open licensing (CC BY 4.0) and long-term hosting via Zenodo. This resource is intended for researchers and practitioners working on LLM-based accessibility validation, inclusive software engineering, and AI-assisted frontend development.

Dataset: https://www.doi.org/10.5281/zenodo.17062188.

Keywords: web accessibility; large language models; open data; dynamic tables; vue components; human–AI interaction; digital sustainability

Academic Editor: Davide Martinenghi

Received: 7 August 2025
Revised: 5 September 2025
Accepted: 18 September 2025
Published: 19 September 2025

Citation: Andruccioli, M.; Bassi, B.; Delnevo, G.; Salomoni, P. The Tabular Accessibility Dataset: A Benchmark for LLM-Based Web Accessibility Auditing. *Data* **2025**, *10*, 149. https://doi.org/10.3390/data10090149

1. Summary

Web accessibility is a fundamental pillar of an inclusive digital society [1]. It ensures that individuals with disabilities can perceive, understand, navigate, and interact with online content effectively, thus supporting equitable participation in economic, educational, and civic life [2]. With the increasing reliance on the web as a primary interface for services and information, accessibility is no longer a niche concern but a mainstream requirement [3]. In particular, adherence to global standards such as the Web Content Accessibility Guidelines (WCAG) has become a legal and ethical imperative across many sectors and jurisdictions [4].

Despite the formalization of accessibility standards and the proliferation of automated tools, such as WAVE [5], Axe [6], and Lighthouse [7], web accessibility remains a persistent challenge [8]. Existing tools are often limited to detecting syntactic issues and fail to capture semantic or context-dependent violations [9], especially in modern web applications that rely heavily on JavaScript frameworks, dynamic content rendering, and component-based architectures [10]. These limitations contribute to widespread accessibility gaps, particularly in websites developed without expert oversight or adequate accessibility training [11].

Artificial Intelligence (AI) and Machine Learning (ML) have recently emerged as promising avenues for addressing these limitations by enabling more nuanced and scalable evaluations of web accessibility [12]. For example, Makati [13] presented DocTTTTTQuery, a method that employs a sequence-to-sequence model (i.e., T5) to generate a set of anticipated queries for a given document. These generated queries are then appended to the original document, creating an enhanced version. The underlying principle is that by enriching the document with potential queries, the overall retrieval performance is improved. The author demonstrated that using these enhanced documents leads to superior retrieval results. Instead, Ara and Sik-Lanyi [14] evaluated website accessibility using a decision tree. The model is trained on data generated by the Mauve tool, which assesses websites against various checkpoints from the WCAG 2.1. The authors aimed to identify accessibility barriers and demonstrate the need for stronger web accessibility policies. Their experiments, validated through a confusion matrix and classification report, found that none of the tested webpages were free from accessibility errors, highlighting a significant need for improved directives to enhance social inclusiveness.

In such a context, Large Language Models (LLMs) such as GPT-4, Claude, and LLaMA have shown considerable potential due to their ability to interpret source code, understand natural language requirements, and reason about content structure and intent [15]. Rather than relying solely on rigid rule-based systems, LLMs can offer more flexible and context-aware assessments. This represents a paradigm shift in accessibility validation from conventional DOM-level checks and static analyzers to AI-powered agents that can analyze raw HTML/CSS/JS code, interpret it semantically, and suggest or apply fixes [16]. Several recent studies illustrate this emerging capability. For instance, Othman et al. [17] investigated how ChatGPT can remediate WCAG 2.1 violations identified by the WAVE tool, showing significant improvement in compliance through automated fixes. Similarly, Delnevo et al. [18] assessed LLMs' ability to validate and correct HTML forms and tables, finding that ChatGPT could successfully detect and fix structural issues such as missing labels and incorrect semantic tags. López-Gil and Pereira [19] further explored LLMs for automating manual WCAG criteria evaluation, such as interpreting link purposes or identifying language mismatches, reaching detection accuracies beyond 85%. However, these studies also highlight the limitations of current models, especially in dealing with missing structural elements and generating consistent remediations.

The dataset described in this article was created to systematically investigate and benchmark the capabilities of LLMs in detecting and remediating accessibility issues directly at the source code level. It is motivated by the need for a structured, reproducible, and scalable resource that allows researchers to explore how LLMs perform in realistic, code-centric accessibility validation scenarios. The dataset includes a diverse set of code snippets, some intentionally injected with accessibility violations and others designed to meet best practices.

The dataset is structured into two primary directories, each targeting distinct facets of accessibility evaluation within modern web development. The first directory, titled "Dynamic Generated Content", comprises a collection of code snippets developed using four widely adopted frontend and backend technologies, Angular, React, Vue, and PHP. Each snippet contains two functionally equivalent versions of a tabular component, with one that adheres to established WCAG and another that deliberately incorporates common accessibility violations. These violations reflect typical real-world pitfalls, such as missing ARIA attributes, poor semantic structure, or inaccessible keyboard interactions, thereby simulating realistic development scenarios where accessibility might be overlooked.

The second directory, named "Vue Table Components", includes 25 distinct Vue-based implementations of table components. These examples were carefully selected and constructed to represent a diverse range of development patterns and accessibility practices. They vary in several dimensions, including the programming paradigm (with examples written using both the Composition API and the Options API), the degree and type of

accessibility support provided, and the complexity of the internal structure (e.g., use of slots, scoped components, or dynamic data binding). This portion of the dataset is particularly valuable for studying how different architectural choices in Vue development influence the interpretability and accessibility of the resulting components.

Together, these two parts provide a rich and heterogeneous source of annotated web components for training, evaluating, and benchmarking LLMs and accessibility auditing tools, particularly in contexts involving dynamically generated content, which remains a significant challenge for traditional static analysis methods.

This dataset supports a broader line of research aimed at exploring the responsible integration of AI into web development workflows, with an emphasis on equity, sustainability, and inclusivity. The dataset is intended to serve multiple audiences, including AI researchers developing accessibility-aware models, Human–Computer Interaction (HCI) practitioners evaluating usability, accessibility auditors seeking AI assistance, and educators looking to train students in inclusive web development. By releasing this dataset publicly, we aim to facilitate a deeper understanding of the strengths and limitations of LLMs in real-world accessibility use cases.

The initial results based on this dataset have been published in [20], where GPT-4o mini, o3-mini, and Gemini 2.0 Flash were employed to systematically evaluate the capabilities of LLMs to identify and assess accessibility issues.

2. Data Description

The dataset is structured into two primary directories, each targeting distinct facets of accessibility evaluation within modern web development. The first directory, titled "Dynamic Generated Content", comprises a collection of code snippets developed using four widely adopted frontend and backend technologies, Angular, React, Vue, and PHP. This directory is further organized into two subfolders. The "data" subfolder contains the source code for each snippet, representing the underlying implementations of the accessible and non-accessible tabular components. The "eval" subfolder includes resources for benchmarking and analysis, including a "labels.csv" file that enumerates the accessibility violations present in each snippet based on source code inspection and an "accessibility-reports" directory containing the reports automatically generated by the Mauve accessibility tool from the rendered HTML output. Together, these artifacts enable both a code-level and rendered-output evaluation of accessibility, facilitating a comparative analysis of LLM predictions against ground truth labels and tool-generated diagnostics. Each snippet contains two functionally equivalent versions of a tabular component, with one that adheres to the established WCAG and another that deliberately incorporates common accessibility violations. These violations reflect typical real-world pitfalls, such as missing ARIA attributes, poor semantic structure, or inaccessible keyboard interactions, thereby simulating realistic development scenarios where accessibility might be overlooked.

The name of files is defined as "*$framework* -table-*$validity*", where

- *$framework* indicates the Javascript framework (i.e., "angular", "react", and "vue") or the "php" language.
- *$validity* indicates whether the generated table will be valid according to the WCAG ("accessible") or not ("invalid").

Hence, the full list of files of the "Dynamic Generated Content" folder is

- angular-table-accessible.js;
- angular-table-invalid.js;
- php-table-accessible.php;
- php-table-invalid.php;
- react-table-accessible.js;

- react-table-invalid.js;
- vue-table-accessible.html;
- vue-table-invalid.html.

The second directory, named "Vue Table Components", includes 25 distinct Vue-based implementations of table components. The former one is structured in the "data" and "eval" subfolders. The table components were carefully constructed to represent a diverse range of development patterns and accessibility practices. They vary in several dimensions, including the programming paradigm (with examples written using both the Composition API and the Options API), the degree and type of accessibility support provided, and the complexity of internal structure (e.g., use of slots, scoped components, or dynamic data binding). This portion of the dataset is particularly valuable for studying how different architectural choices in Vue development influence the interpretability and accessibility of the resulting components.

Each distinct Vue-based implementation is structured in a separate folder named "delivery-*$n*", where "*$n*" ranges from "01" to "25". Each folder contains the following files and folders:

```
Vue Table Components/data
└─ delivery-$n
   ├─ dataset The folder used to store data.
   │  └─ *.(json|csv) The JSON or CSV file that contains the dataset that is visualized
   │     in the table.
   ├─ src The folder that contains the source code of the various components.
   │  ├─ accessible-composition-api/ The folder contains an accessible version of the
   │  │  table showing the dataset using the Composition API.
   │  ├─ accessible-options-api/ The folder contains an accessible version of the table
   │  │  showing the dataset using the Options API.
   │  ├─ minimal-composition-api/ The folder contains a minimal version of the table.
   │  │  The implementation does not consider possible accessibility issues and shows the
   │  │  dataset using the Composition API.
   │  ├─ minimal-options-api/ The folder contains a minimal version of the table.
   │  │  The implementation does not consider possible accessibility issues and shows the
   │  │  dataset using the Options API.
   │  └─ App.vue This is the root component of the Vue application. All other components
   │     are rendered within this component. It contains the template for the main
   │     application layout that include the script and style.
   ├─ index.html The entry point of the application. This is a basic HTML file where
   │  the Vue application is mounted. It also includes the necessary script tags to load
   │  the compiled JavaScript.
   ├─ jsconfig.json A configuration file used to specify path aliases and indicate the
   │  compiler which folders to ignore.
   ├─ main.js The main JavaScript file serves as the entry point. It imports the
   │  Vue framework and the main App.vue component and mounts the application to the
   │  index.html file.
   ├─ package.json This file contains all the metadata for the project including its
   │  name version scripts such as dev and build and a list of all project dependencies
   │  and dev dependencies.
   ├─ package-lock.json This file is automatically generated by npm and locks down
   │  the exact versions of all installed dependencies and their sub-dependencies. This
   │  ensures that everyone working on the project has the exact same package versions
   │  and prevents potential inconsistencies.
   └─ vite.config.js The configuration file for Vite that is the build tool used for
      projects. This file defines how the project is built and served including settings
      for plugins server behavior and build output.
```

3. Methods

3.1. Languages and Frameworks Covered

The dataset has been designed to reflect real-world patterns in contemporary web development, with a particular focus on technologies that dominate the current frontend and backend ecosystem. In selecting the technologies to be included, priority was given to JavaScript frameworks and PHP based on their widespread adoption and relevance in both legacy and modern web applications.

JavaScript remains the cornerstone of interactive web development. Within the JavaScript ecosystem, the dataset includes examples written in Vue, React, and Angular, three of the most prominent frontend frameworks. These frameworks offer declarative syntax, component-based architecture, and varying levels of support for accessibility, making them particularly relevant for studying how accessibility violations can manifest and be remediated in different environments. Their differences in templating systems, rendering strategies, and reactive paradigms also provide a useful contrast for evaluating the reasoning capabilities of LLMs when analyzing source code.

Vue.js was selected for special attention due to its flexibility in allowing developers to write components using two distinct paradigms, namely the Options API and the newer Composition API. The Options API relies on a traditional object-based syntax, promoting readability and clarity for beginners. In contrast, the Composition API uses the `setup()` function and encourages function-based organization, improving code reuse and scalability in complex applications. Including both paradigms in the dataset enables a fine-grained analysis of how code structure and abstraction style may affect accessibility practices and their interpretability by LLMs.

In parallel, PHP was included as a representative of server-side technologies that still play a significant role in generating dynamic HTML content. Though often overlooked in modern single-page applications, PHP remains widely used in content management systems and traditional websites. Including PHP in the dataset allows for the evaluation of LLMs in contexts where accessibility violations arise from backend generated content, such as dynamic tables or forms rendered from server-side scripts.

Together, these frameworks and languages ensure that the dataset captures both client-side and server-side sources of accessibility issues, supporting comprehensive analysis and benchmarking.

3.2. Details on Development

The dataset was constructed through a two-phase process involving both professor-authored and student-authored code. The first phase involved manually crafting dynamic web snippets in Angular, React, Vue, and PHP. For each technology, two versions of a table component were created, with one being accessible, adhering to WCAG 2.1 guidelines (e.g., semantic markup, ARIA roles, keyboard support), and one being non-accessible, featuring common violations such as missing labels, improper heading structures, or lack of alternative text. These components were written to reflect typical developer practices and errors, ensuring realistic input for LLM evaluation.

The second phase involved collecting contributions from a cohort of computer science students participating in a web development course. Each student was instructed to design and implement four independent Vue table components as Single-File Components (SFCs) using `.vue` files. The deliverables included

- A minimal table using the Options API;
- A minimal table using the Composition API;
- An accessible table using the Options API;
- An accessible table using the Composition API.

To guide development and promote consistency, the following constraints were imposed:

- Each student was required to select a unique dataset of at least 50 rows and 5 attributes, representing real-world data such as songs, movies, environmental measurements, etc.
- Students were encouraged to modularize their code using multiple components where appropriate and to leverage native Vue features (e.g., props, slots, reactivity).
- All components had to remain autonomous, meaning no shared logic or global imports were allowed across implementations. Code duplication was explicitly permitted to ensure isolation.
- Deliverables were submitted as compressed `.zip` archives, with the source code and data included, but build artifacts (e.g., `node_modules`, `dist`) were excluded for portability and reproducibility.

To ensure methodological rigor and reproducibility, the dataset creation process was structured into a multi-step workflow. The activities were carried out iteratively by students and the professor as follows (Figure 1 provides a visual overview of the process):

1. A set of rules and consistency guidelines for code submission was defined.
2. Each student proposed a unique dataset through a dedicated online form, ensuring diversity and avoiding duplication across the cohort.
3. Students implemented their Vue components according to the predefined rules and submitted their initial code.
4. The professor performed a first review of all submissions, providing individual feedback.
5. Students revised and corrected their code based on the professor's comments and submitted the final version.
6. The professor collected all the submitted works and performed manual quality checks, including
    - Normalization of directory structures;
    - Removal of unnecessary or temporary files (e.g., `node_modules`,`dist`, editor configuration);
    - Verification of code readability, indentation, and consistency;
    - Ensuring presence of both accessible and non-accessible versions.
7. The entire dataset was anonymized to remove any personally identifiable information or metadata.

In parallel to the student contributions, the professor developed the dynamically generated content dataset, which has gone through its own validation phase to integrate manually created components. Once both parts were refined and validated, they were carefully integrated into a single collection that constituted the final dataset. Finally, the dataset was then uploaded to Zenodo to ensure open access, reproducibility, and long-term preservation. This workflow guaranteed both the quality and the FAIR compliance of the resulting dataset.

This rigorous protocol ensured that each submission represented a self-contained, fully functional Vue project capable of being evaluated in isolation. Importantly, the dual-paradigm structure (Options vs. Composition API) allows researchers to study whether the underlying code organization influences the likelihood of accessibility errors or the ease with which LLMs can interpret and remediate them.

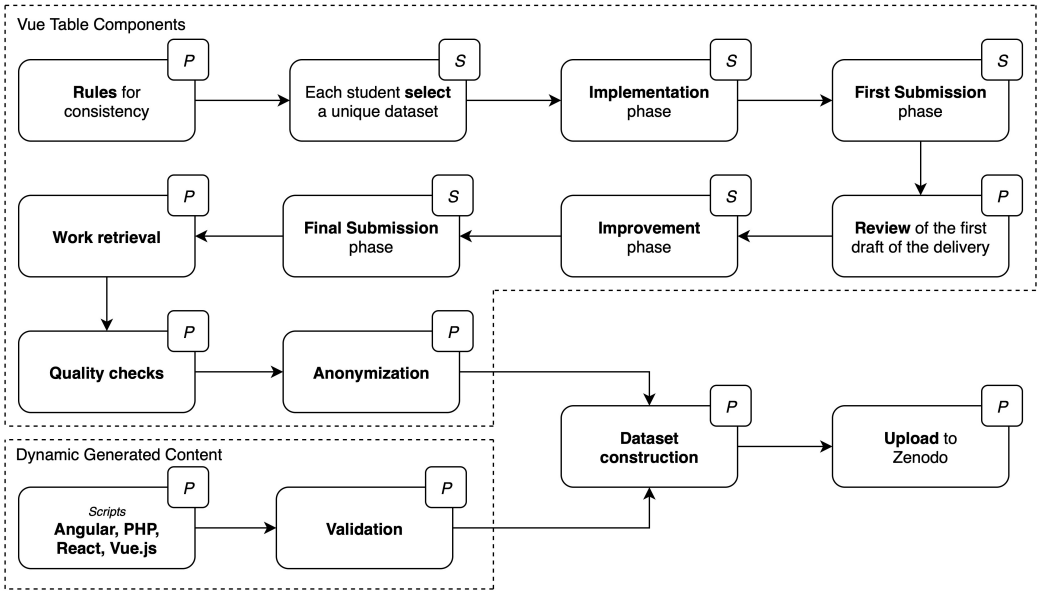

Figure 1. Methodology for the creation of the dataset. Tasks were performed accordingly by professor (P) or students (S).

3.3. Implementation Diversity Within Table Structures

A critical aspect of our benchmarking dataset is the diversity of table implementations. To effectively evaluate an LLM's ability to navigate and interpret the DOM of complex web pages, we first categorized our collected table examples based on their underlying Vue component architecture. The results of this categorization are presented in Table 1.

Table 1. Categorization of table implementations based on Vue component splitting and their corresponding delivery examples.

Split Approach	Deliveries
Single file	3, 4, 10, 13, 17, 18, 19, 23
Body-only	1, 8, 12
Header and body	2, 5, 6, 7, 9, 11, 14, 15, 16, 20, 21, 22, 24, 25

As shown in the table, we identified three primary approaches to table component splitting. The first, labeled "Single file", represents the simplest and most common implementation, where the entire table structure, including the <table>, <thead>, and <tbody> elements, is contained within a single Vue component. This approach was observed in a significant portion of our dataset, including eight deliveries.

A more modular approach is the "Body-only" split. In this configuration, only the <tbody> element is encapsulated within a dedicated child component, with the parent component managing the <table> and <thead> elements. This method is often used to optimize the rendering performance for tables with dynamically updating rows. We observed this pattern in three deliveries.

Finally, the most granular architectural split is the "Header and Body" approach. This methodology separates both the <thead> and the <tbody> into distinct, self-contained child components. This is typically performed to allow for independent manipulation of the header and body content, such as sorting or filtering, without impacting the other part of the table. This complex yet common pattern was found in the largest subset of our deliveries, specifically 14 deliveries.

While the number of samples is limited, the chosen deliveries are abstracted from the datasets that will populate the tables. This abstraction ensures that our sample set represents a wide variety of implementation patterns, providing a robust benchmark for

evaluating LLMs on their ability to handle varying levels of DOM complexity, a crucial skill for accurate accessibility analysis.

3.4. Selected Datasets

As part of the data collection process, students were assigned the task of implementing Vue-based table components using a dataset of their choice. Each student selected a unique dataset, ensuring diversity in content while adhering to a common structure requirement, namely a minimum of 50 records and 5 features per dataset. The topics ranged from entertainment and technology to science and public health, providing a rich variety of use cases for web interface implementations.

Table 2 summarizes the datasets selected by each student. It is worth noticing that in the first delivery, the student employed a self-constructed dataset.

Table 2. List of datasets used by students in the various deliveries.

Delivery	Name	Reference
delivery-01	Cinema anomalies	N/A
delivery-02	Amazon Musical Instruments Reviews	[21]
delivery-03	NASA - Near-Earth Asteroids and Comets	[22]
delivery-04	NVIDIA-STOCK-DATA	[23]
delivery-05	Real World Smartphone's	[24]
delivery-06	Used Car	[25]
delivery-07	Achieved Frames per Second (FPS) in Video Games	[26]
delivery-08	Pokemon	[27]
delivery-09	Footballers with 50+ International Goals [men]	[28]
delivery-10	EPL Dataset 2022/2023	[29]
delivery-11	Global Heath Statistics	[30]
delivery-12	World Population by Country	[31]
delivery-13	Air Quality and Pollution Assessment	[32]
delivery-14	Crime Rate by Country 2024	[33]
delivery-15	NBA Players	[34]
delivery-16	Fortnite Players Stats	[35]
delivery-17	League of Legends Master+ Players	[36]
delivery-18	Top 100 Most Streamed Songs on Spotify	[37]
delivery-19	The Office Dataset	[38]
delivery-20	Food Recipes	[39]
delivery-21	NASA - Earth Meteorite Landings	[40]
delivery-22	Google Playstore	[41]
delivery-23	Erasmus mobility statistics 2014_2019	[42]
delivery-24	Bank marketing	[43]
delivery-25	Game Recommendations on Steam	[44]

3.5. Usage and Application

This dataset is designed to support a wide range of research and practical applications in the field of AI-assisted web accessibility auditing and remediation. It can be used in isolation (via source code analysis) or in combination with rendered output (i.e., the DOM produced by each component) to investigate accessibility behaviors from both a static and dynamic perspective.

There are several potential use cases, as follows:

- Source Code Analysis Only: Researchers may focus solely on the raw HTML, Vue, React, Angular, or PHP source code to test the ability of LLMs or static analyzers

to detect accessibility issues without relying on runtime context. This is especially relevant for pre-deployment code review tools.
- Rendered Output Analysis: By executing the components and inspecting the resulting DOM, researchers can simulate what a user's browser would process. This method is suitable for testing tools that analyze accessibility post-rendering (e.g., using Axe-core or Lighthouse).
- Combined Static and Dynamic Analysis: The most comprehensive use involves combining code-level insights with runtime behaviors, allowing for a comparison of LLM predictions and traditional tool outputs. This approach is particularly useful for validating whether LLMs can correctly infer the final structure and accessibility state of dynamic components.

Additionally, the dataset provides an excellent testbed for fine-tuning LLMs, evaluating prompt engineering strategies, or developing hybrid accessibility checkers that integrate rule-based systems with generative AI models. Its diverse inclusion of languages, paradigms, and component architectures makes it uniquely suited for advancing research at the intersection of web development, accessibility, and AI.

The first set of experiments conducted that take advantage of this dataset are presented in [20]. The authors evaluated three different LLMs, for instance, GPT-4o mini, o3-mini, and Gemini 2.0 Flash, to evaluate the accessibility of dynamically generated web content on all scripts and languages available.

4. User Notes

4.1. Compliance with FAIR Principles

The dataset adheres to the FAIR principles—findability, accessibility, interoperability, and reusability [45]:

- Findability: The dataset is uniquely identified by a Digital Object Identifier (DOI). This permanent reference acts like a digital fingerprint, ensuring it can always be found and is cited accurately.
- Accessibility: The dataset is hosted on Zenodo, a repository dedicated to open science that supports long-term preservation.
- Interoperability: The included examples are composed of plain text files. This makes the data highly flexible and easy to work with, as the source code can be analyzed directly with any standard text editor or programming tool.
- Reusability: The content is published under a Creative Commons Attribution 4.0 International (CC BY 4.0) license. This a very permissive license that allows for the unrestricted use, modification, and distribution of the dataset as long as the original creators are credited.

4.2. Dataset Limitations

While the dataset offers a rich and diverse collection of web components designed to evaluate accessibility in dynamic web applications, several limitations must be acknowledged.

First, the dataset is inherently synthetic and educational in nature. A significant portion of the components, particularly those implemented in Vue, were developed by students as part of a university assignment. Although the submissions were reviewed for correctness, completeness, and adherence to accessibility principles, the code may not fully represent the complexity, scalability, or design patterns commonly found in large-scale, industry-grade applications. As such, the dataset may lack examples of deeply nested components, complex interactivity, or third-party library integrations.

Second, the dataset is currently limited to table-based components, both accessible and non-accessible. While tables are a critical element for accessibility, frequently subject to

violations and requiring semantic precision, they do not cover the full breadth of accessibility challenges faced in modern web interfaces, such as modal dialogs, custom widgets, form validation flows, or responsive navigation menus.

Third, the focus on Vue (version 3), React (version 18), Angular (version 18), and PHP (version 8.3) excludes other emerging or legacy technologies. Frameworks such as Svelte, Nuxt.js, or server-side technologies like ASP.NET and Python-based frameworks (e.g., Django, Flask) are not included. This may limit the generalizability of findings derived from this dataset to other technological contexts.

Lastly, while the Vue components were carefully designed to be diverse, the controlled structure of the assignment may have led to similarities in implementation approaches, code style, or problem-solving strategies. Furthermore, the requirement for autonomy across implementations—while methodologically useful—may introduce unnatural code duplication not typically seen in production systems.

Despite these limitations, the dataset serves as a valuable and novel resource for the community, particularly for studying how Large Language Models interpret and reason about accessibility in source code across different frameworks and paradigms.

5. Conclusions and Future Work

This paper presents a novel dataset of accessible and non-accessible web components, spanning multiple languages and frameworks with a particular focus on Vue.js implementations. The dataset was carefully curated to provide controlled variations in accessibility, code organization, and development paradigms. It enables diverse research directions, including automated accessibility auditing, LLM-based code reasoning, and the training of supervised or reinforcement learning systems.

The collection bridges a gap between synthetic benchmarks and real-world web code, offering a unique testbed for evaluating the ability of AI models to reason about dynamic and component-based content. By combining static and dynamic code representations, the dataset supports multi-faceted analysis pipelines involving source-level analysis and DOM-based audit tools.

In future studies, we plan to

- Extend the dataset with additional User Interface (UI) components such as modals, forms, and menus that pose more advanced accessibility challenges.
- Incorporate runtime accessibility audit results (e.g., from Axe-core, Lighthouse) as labels to enable supervised learning approaches.
- Develop benchmark tasks for evaluating the accessibility-awareness of LLMs through code completion, explanation, or repair prompts.
- Expand to additional frameworks such as Svelte Nuxt.js and potentially server-rendered environments such as Django or ASP.NET.

We believe this dataset will serve as a valuable foundation for the development of next-generation accessibility tools and AI-driven development environments that embed inclusive design by default.

Author Contributions: Conceptualization, G.D. and P.S.; methodology, B.B.; software, B.B.; validation, M.A., B.B., and G.D.; data curation, M.A.; writing—original draft preparation, M.A. and B.B.; writing—review and editing, G.D. and P.S.; visualization, M.A.; supervision, P.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data presented in the study are openly available in Zenodo at https://doi.org/10.5281/zenodo.17062188.

Acknowledgments: We thank Giulia Bonifazi for her precious support and all the students who developed a version of the Vue components. During the preparation of this manuscript/study, the authors used ChatGPT (GPT-4o) for the purposes of writing assistance. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

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

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
ML	Machine Learning
LLM	Large Language Model
SDGs	Sustainable Development Goals
WCAG	Web Content Accessibility Guidelines
HCI	Human–Computer Interaction
DOI	Digital Object Identifier
SFC	Single-File Component
UI	User Interface

References

1. Ferri, D.; Favalli, S. Web Accessibility for People with Disabilities in the European Union: Paving the Road to Social Inclusion. *Societies* **2018**, *8*, 40. [CrossRef]
2. Bricout, J.; Baker, P.M.A.; Moon, N.W.; Sharma, B. Exploring the Smart Future of Participation: Community, Inclusivity, and People With Disabilities. *Int. J.-E-Plan. Res.* **2021**, *10*, 94–108. [CrossRef]
3. Teixeira, P.; Eusébio, C.; Teixeira, L. Understanding the integration of accessibility requirements in the development process of information systems: A systematic literature review. *Requir. Eng.* **2024**, *29*, 143–176. [CrossRef]
4. Abou-Zahra, S.; Brewer, J., Standards, Guidelines, and Trends. In *Web Accessibility*; Springer: London, UK, 2019; pp. 225–246. [CrossRef]
5. WebAIM-Web Accessibility in Mind. WAVE Web Accessibility Evaluation Tools. 2025. Available online: https://wave.webaim.org/ (accessed on 6 August 2025).
6. Deque Systems Inc. Axe: The Accessibility Engine. 2024. Available online: https://www.deque.com/axe/ (accessed on 6 August 2025).
7. Chrome for Developers. Introduzione a Lighthouse. 2025. Available online: https://developer.chrome.com/docs/lighthouse/overview?hl=en (accessed on 6 August 2025).
8. Ara, J.; Sik-Lanyi, C. Automated evaluation of accessibility issues of webpage content: Tool and evaluation. *Sci. Rep.* **2025**, *15*, 9516 . [CrossRef] [PubMed]
9. Alarcon, R.; Moreno, L.; Martinez, P. Lexical Simplification System to Improve Web Accessibility. *IEEE Access* **2021**, *9*, 58755–58767. [CrossRef]
10. Roumeliotis, K.I.; Tselikas, N.D. Evaluating Progressive Web App Accessibility for People with Disabilities. *Network* **2022**, *2*, 350–369. [CrossRef]
11. Seixas Pereira, L.; Duarte, C. Evaluating and monitoring digital accessibility: Practitioners' perspectives on challenges and opportunities. *Univers. Access Inf. Soc.* **2025**, *24*, 2553–2571. [CrossRef]
12. Abou-Zahra, S.; Brewer, J.; Cooper, M. Artificial Intelligence (AI) for Web Accessibility: Is Conformance Evaluation a Way Forward? In Proceedings of the 15th International Web for All Conference, W4A '18, Lyon, France, 23–25 April 2018; ACM: New York, NY, USA, 2018; pp. 1–4. [CrossRef]
13. Makati, T. Machine learning for accessible web navigation. In Proceedings of the 19th International Web for All Conference, W4A'22, Lyon, France, 25–26 April 2022; ACM: New York, NY, USA, 2022; pp. 1–3. [CrossRef]
14. Ara, J.; Sik-Lanyi, C. Webpage Accessibility Evaluation Using Machine Learning Technique. In Proceedings of the 2023 14th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), Budapest, Hungary, 22–23 September 2023; IEEE: New York, NY, USA, 2023; Volume 9, pp. 000069–000074. [CrossRef]
15. Pedemonte, G.; Leotta, M.; Ribaudo, M. Improving Web Accessibility With an LLM-Based Tool: A Preliminary Evaluation for STEM Images. *IEEE Access* **2025**, *13*, 107566–107582. [CrossRef]

16. Oswal, S.K.; Oswal, H.K., Conversational AI for Accessible Website Design: Integrating LLM Assistants in Website Builders. In *New Frontiers for Inclusion*; Springer Nature: Cham, Switzerland, 2025; pp. 251–261. [CrossRef]
17. Othman, A.; Dhouib, A.; Nasser Al Jabor, A. Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation. In Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments, PETRA '23, Corfu Greece, 5–7 July 2023; ACM: New York, NY, USA, 2023; pp. 707–713. [CrossRef]
18. Delnevo, G.; Andruccioli, M.; Mirri, S. On the Interaction with Large Language Models for Web Accessibility: Implications and Challenges. In Proceedings of the 2024 IEEE 21st Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 6–9 January 2024; IEEE: New York, NY, USA, 2024; pp. 1–6. [CrossRef]
19. López-Gil, J.M.; Pereira, J. Turning manual web accessibility success criteria into automatic: An LLM-based approach. *Univers. Access Inf. Soc.* **2024**, *24*, 837–852. [CrossRef]
20. Andruccioli, M.; Bassi, B.; Delnevo, G.; Salomoni, P. Leveraging Large Language Models for Sustainable and Inclusive Web Accessibility. *Preprints* **2025**. [CrossRef]
21. Amazon Music Reviews Dataset. Available online: https://www.kaggle.com/datasets/eswarchandt/amazon-music-reviews (accessed on 15 May 2025).
22. NASA. Near-Earth Asteroids and Comets. Available online: https://data.nasa.gov/resource/2vr3-k9wn.json (accessed on 6 August 2025).
23. NVIDIA Stock Data. Available online: https://www.kaggle.com/datasets/muhammaddawood42/nvidia-stock-data (accessed on 15 May 2025).
24. Real World Smartphones Dataset. Available online: https://www.kaggle.com/datasets/abhijitdahatonde/real-world-smartphones-dataset (accessed on 15 May 2025).
25. Used Car Dataset. Available online: https://www.kaggle.com/datasets/mohitkumar282/used-car-dataset?select=used_car_dataset.csv (accessed on 15 May 2025).
26. FPS in Video Games Dataset. Available online: https://www.kaggle.com/datasets/kritikseth/achieved-frames-per-second-fps-in-video-games (accessed on 15 May 2025).
27. Pokémon Dataset. Available online: https://www.kaggle.com/datasets/jaidalmotra/pokemon-dataset (accessed on 15 May 2025).
28. Footballers with 50+ International Goals. Available online: https://www.kaggle.com/datasets/whisperingkahuna/footballers-with-50-international-goals-men (accessed on 15 May 2025).
29. EPL Dataset 2022-2023. Available online: https://www.kaggle.com/datasets/acothaha/epl-dataset-20222023-update-every-week (accessed on 15 May 2025).
30. Global Health Statistics. Available online: https://www.kaggle.com/datasets/malaiarasugraj/global-health-statistics (accessed on 15 May 2025).
31. 2023 World Population by Country. Available online: https://www.kaggle.com/datasets/rajkumarpandey02/2023-world-population-by-country?resource=download&select=countries-table.json (accessed on 15 May 2025).
32. Air Quality and Pollution Assessment. Available online: https://www.kaggle.com/datasets/mujtabamatin/air-quality-and-pollution-assessment (accessed on 15 May 2025).
33. Crime Rate by Country 2024. Available online: https://www.kaggle.com/datasets/shahriarkabir/crime-rate-by-country-2024 (accessed on 15 May 2025).
34. NBA Players Data. Available online: https://www.kaggle.com/datasets/justinas/nba-players-data (accessed on 15 May 2025).
35. Fortnite Players Stats. Available online: https://www.kaggle.com/datasets/iyadali/fortnite-players-stats?select=Fortnite_players_stats.csv (accessed on 15 May 2025).
36. League of Legends Master Players. Available online: https://www.kaggle.com/datasets/jasperan/league-of-legends-master-players (accessed on 15 May 2025).
37. Top 100 Most Streamed Songs on Spotify. Available online: https://www.kaggle.com/datasets/pavan9065/top-100-most-streamed-songs-on-spotify/data (accessed on 15 May 2025).
38. The Office Dataset. Available online: https://www.kaggle.com/datasets/nehaprabhavalkar/the-office-dataset (accessed on 15 May 2025).
39. Recipes3k Dataset. Available online: https://www.kaggle.com/datasets/crispen5gar/recipes3k (accessed on 15 May 2025).
40. NASA Meteorite Landings (Y77D-TH95). Available online: https://data.nasa.gov/resource/y77d-th95.json (accessed on 15 May 2025).
41. Google Play Store Dataset. Available online: https://www.kaggle.com/datasets/arnikaer/googleplaystore (accessed on 15 May 2025).
42. Erasmus Mobility Statistics (2014–2019). Available online: https://www.kaggle.com/datasets/donjoeml/erasmus-mobility-statistics-2014-2019 (accessed on 15 May 2025).
43. Bank Marketing Dataset. Available online: https://www.kaggle.com/datasets/mahdiehhajian/bank-marketing (accessed on 15 May 2025).

44. Steam Game Recommendations. Available online: https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam (accessed on 15 May 2025).
45. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **2016**, *3*, 160018. [CrossRef] [PubMed]