Implementation of a Webpage Editor for Pages with Minimum JavaScript Dependency

Runyao Fan
Computer Science, NYUAD
rf1888@nyu.edu

Advised by: Professor Yasir Zaki, Professor Riyadh Baghdadi

ABSTRACT

JavaScript loading and parsing have been a significant source of page loading time. To enable mobile phone users in regions with poor internet connections, it is necessary to create lightweight web pages devoid of the myriad of functionalities enabled by JavaScript. Potential solutions include optimizing the loading process of websites, replacing JavaScript with another more efficient way to implement webpages' interactive features, or limiting JavaScript libraries and functionalities supported by web editors or browsers. Over the course of our research, we have noticed that recompiling JavaScript into Go and WebAssembly does not bring improvement in page loading time. Pursuing the approach of simplifying webpage structure, this project designs and implements MAML Editor, a webpage editor that creates webpages with minimum JavaScript dependency. The editor consists of a React-based front end and a back end implemented by Flask. Webpages created by users are transported to the back end in the format of Mobile Application Mark-up Language (MAML). The back end in turn translates the MAML objects to HTML pages with minimum JavaScript dependency. We carried out a study of 17 existing pages by creating pages that are similar in content and style using the MAML Editor. The MAML pages on average reduce the time taken for the "load" event to occur by 93.5%.

KEYWORDS

page load time, web page simplification, website editor, web page interactive functionality

Reference Format:
Runyao Fan. 2022. Implementation of a Webpage Editor for Pages with Minimum JavaScript Dependency. In *NYUAD Capstone Project 2 Reports, Spring 2022, Abu Dhabi, UAE.* 7 pages.

1 INTRODUCTION

The Internet plays an important role in the life of people in less developed countries and regions. It offers an inexpensive way for the less privileged to acquire otherwise inaccessible information, such as best practices in farming, information on disease prevention and treatment, and education resources in many subjects and fields. Moreover, the internet has improved the standard of living for many by providing a platform to advocate for their product and business [3] [5]. Nevertheless, internet accessibility in underdeveloped regions is still heavily restrained due to the lack of infrastructure development and the poor processing power of low-end smartphones. Page load times in developing countries can be as high as 60 seconds, while more than half of users tend to stop visiting a page if the page load time exceeds 3 seconds [1]. Page load time in developing regions needs to be reduced so that people from these areas can have unhindered internet access that everyone deserves.

Research has shown that the complex structure of modern webpages is a significant reason for long page loading time. Modern webpages often request resources from many servers located across the globe [4]. A study by Butkiewicz et. al. showed that over 60 percent of webpages make requests from no less than five different non-origin sources, and these requested contents account for over 35 percent of the page size [2]. Moreover, the loading of a webpage is blocking and recursive. When the initial HTML document is parsed, it may make requests to more server locations for new resources, which may, in turn, make more internet connections [2]. Moreover, the evaluation of certain parts of an HTML document, such as inline JavaScript, blocks subsequent HTML parsing and the loading of a webpage becomes going through a complex dependency graph, where many operations await completion of other downloads and evaluations.

This project aims to reduce page loading time, especially for mobile users in regions with poor internet infrastructure. It is based on MAML, a mark-up language format that inhibits JavaScript and represents web page elements in a simple way. Nevertheless, interactivity has become an indispensable feature of many websites and this project aims to retain essential interactive features for users while using the least amount of JavaScript code. As MAML pages are created with the MAML editor, users simply need to be concerned with the type of components they want to add to the webpage. The concern of how different types of components are implemented lies in the backend, where user-created components represented by MAML objects are translated into HTML pages. As such, we can control the amount of JavaScript in the generated pages by carefully designing the translation process. While many interactive features can be implemented using JavaScript's rich collection of libraries, we found that components such as carousel and dropdown menus can be implemented with just HTML and CSS. When interactive components do not need JavaScript to function, the translation process creates interactive components in HTML pages without using JavaScript.

2 BACKGROUND RESEARCH

As we aim to limit the amount of JavaScript in the editor-generated websites, we first conducted research on the website features that may depend on JavaScript and are used the most frequently. In this way, we can allow the editor to support the most necessary JavaScript functionalities, making the generated websites as lightweight as possible without compromising the interactiveness essential for users.

2.1 Subject of Analysis

We used Amazon's Alexa Web Ranking service to inspect the world's top 100 websites by traffic. Alexa ranks the websites by a combined score of a website's visitors and pageviews, reflecting the popularity of websites in the entire world instead of just developed regions like the US. For example, the top 100 list consists of many websites catering to specific geographical communities. Web portals of various developing countries, such as qq.com and sohu.com from China and okezone.com from Indonesia, are included in the list. By having these websites less known in other countries but playing an important role in many regions with poor internet infrastructures, the list allows us to analyze website functionalities more comprehensively. We find that these web portals widely used in developing countries have layouts we were unfamiliar with. The analysis provided insights into the usage of interactive components by different websites and gave us ideas on the components the webpage editor should support.

2.2 Website Analysis Methodology

To find the most used interactive features, we open each website in the list, observe components that change automatically, hover on different parts of the page, and click different parts of the website. If we find any interactive event, we then decide if the visual effects depend on JavaScript. For example, many websites feature side menus and dropdown bars that only appear when a user hovers the mouse over or clicks specific components. These effects are often realized by using JavaScript event listeners.

2.3 Website Analysis Result

After inspecting the top 100 websites according to Amazon Alexa, we narrowed down the most used interactive features to 11, which are the following:

- drop-down menu
- loading of new content when the page reaches its bottom
- display video preview when the mouse hovers over the thumbnail
- video player
- carousel
- component that appears after scrolling below a certain point
- countdown timer
- animation triggered by scrolling
- auto-animation
- toggle button that changes page theme
- notification window

This finding indicates that although JavaScript supports a large number of libraries and has been a significant source of websites' loading time, different websites share many essential functionalities. As a result, our inspection shows that the webpage editor only needs to support a small set of JavaScript functionalities used by the most popular websites. Nevertheless, assessing more websites may allow us to discover other interactive features that the webpage editor should also support. A future task of the project may be to inspect more websites and categorize and rank the interactive features. This way, we will have more comprehensive information on the JavaScript-enabled functionalities that we should consider implementing for the webpage editor.

3 DESIGN

To allow content creators to create new webpages or convert existing webpages to ones compatible with the MAML specification, we need a webpage editor that gives the users a simple webpage creation interface while restricting the types of JavaScript functionalities supported. While several commercial webpage editors are in the market, they do not support the customization we need, such as saving the built website to a specific format or restricting the set of JavaScript supported. As such, we choose to develop a webpage editor on our own while referring to existing webpage editors for design and UI practices.

3.1 Expected Outcome

We aim to build a browser-based webpage editor that allows content creators to design and produce websites that resemble the look and functionality of websites we commonly browse. For example, the editor should support text editing, image upload, positioning, and the addition of interactive features such as buttons and side menus. Moreover, the editor should allow content creators to preview the website in a non-editable mode. After successfully implementing the webpage editor with the said functionalities, the editor should also export the completed website in files that follow the syntax specifications of pre-existing MAML implementations.

As there are existing webpage editors in the market, we used some to get ideas on the editor page layout and functionality we should implement for the MAML editor. For example, Wix is an editor with a large user base, and its designs have been tested by many. As Figure 1 below shows, the viewport contains a large editing area with grids, and a side panel on the left end of the page, allowing users to add different elements to the page. Ideally, our website editor should feature a similar side panel with component types such as text box, image, carousel, and dropdown menus. Content users will click on the buttons to add components to the editing area, where they can further customize the content and style of the elements and position them freely in the area. After finalizing the JavaScript functionalities that can be added to the product pages, each functionality can correspond to a button on the side panel. In this way, we can implement the functions of the editor incrementally. Each time, we can add a new button and implement the functionality with minimum JavaScript dependency.

3.2 Development Framework

3.2.1 React. Given the functionalities the editor should support, we decided to develop the editor using React. React is a JavaScript library and a popular solution to build user interfaces for web-based applications. React treats a webpage as different components that maintain their own state. Developers can define a component to specify how it should look on the page by HTML-like syntax and create and manage the component's state. When a component changes in its state, React re-renders the particular component without the need to reload the whole page. Components are also reusable, allowing developers to quickly create components of the same style but different content. These features of React are very useful for the implementation of a webpage editor.

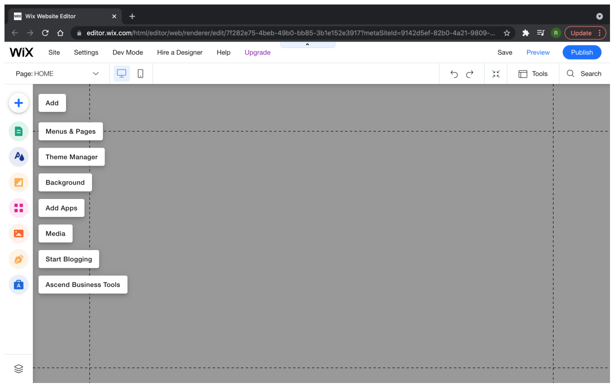
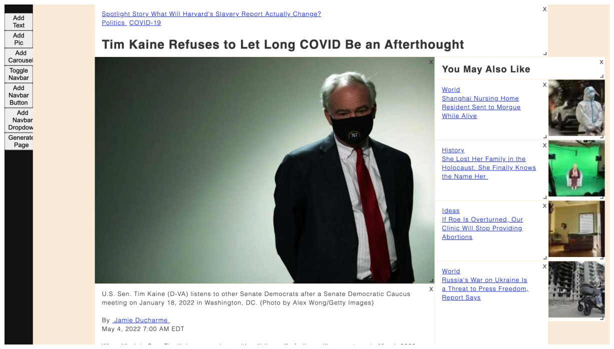


Figure 1: Layout of Wix Website Editor

First of all, the editor's page will be dynamic, as users should be able to add, delete, position, and resize different types of components. React's component-based approach to web development addresses the design needs of the editor. In React, developers can define reusable components that can be placed in different positions on a webpage. It acts as a template and an instance of a reusable component can have its unique content based on properties passed into it. React applications are also fast because React supports component-wise re-rendering. As users need to constantly edit the components they add to the editing area, React re-renders the parts of the page being edited by the content creator while reducing computational cost by leaving everything else intact. Another advantage of React is that the reusable component approach works well for the webpage editor components as they share a number of functional commonalities. For example, regardless of whether a component is a text box, a picture, or a dropdown menu, the content creator should be able to resize them and move them around. Preferably, the editing area should also assist creators in positioning the components by supporting features such as automatic alignment and automatic centering. The idea of abstraction in React allows us to create a grid layout system where users can add and remove rectangular components in it. To differentiate different types of components, we simply need to customize the UI on top of the shared layout implementation.

3.2.2 Node.js and NPM. Another set of tools the project depends on is Node.js and its Node Package Manager (NPM). Using Node.js to create and manage React applications is a widely-used practice, and NPM hosts a large number of packages and extensions that we can conveniently import into the project. For example, a grid layout system supporting dynamic resizing and positioning is a complex task, and we searched for existing solutions on NPM. As a result, we found that an NPM package named react-grid-layout is used by many people to manage pages with moveable, resizable components. It allows users to drag and drop components and implements a grid system where components snap into the pre-defined positions. Comparing the package with several other solutions such as react-moveable, another NPM package, we decided to use the react-grid-layout package as the backbone of the webpage editor, as it supports the aforementioned functionalities that are going to be useful when building a webpage editor.



Spotlight Story What Will Harvard's Slavery Report Actually Change?
Politics COVID-19

Tim Kaine Refuses to Let Long COVID Be an Afterthought

U.S. Sen. Tim Kaine (D-VA) listens to other Senate Democrats after a Senate Democratic Caucus meeting on January 18, 2022 in Washington, DC. (Photo by Alex Wong/Getty Images)

By Jamie Ducharme
May 4, 2022 7:00 AM EDT

You May Also Like

World
Shanghai Nursing Home Resident Sent to Morgue While Alive

History
She Lost Her Family in the Holocaust. She Finally Knows the Name Her

Ideas
If Roe Is Overturned, Our Clinic Will Stop Providing Abortions

World
Russia's War on Ukraine Is a Threat to Press Freedom, Report Says

Figure 2: Creating a Page Using the Editor

4 IMPLEMENTATION

The editor is a browser-based application that supports user registration, authentication, editor state persistence, and translation of MAML pages to HTML pages. Its front end is implemented by React and its backend is implemented by Flask.

When a user visits the register and log in page, they can either register as a new user or log in if they have registered before. After logging in, users will be redirected to Created Pages Gallery, where they can view and retrieve pages they have created before. This is realized by saving users' page information in the database when they send their created pages to the back end for translation. The user pages are grouped by their usernames.

4.1 Editor

The editor page consists of a sidebar and an editing area. The sidebar features buttons for users to add different components to the editing area. To have different buttons add different types of elements to the editing area, each button is bounded to a specific function. The particular function then adds an item of the corresponding type to the list in the state of the component managing the editing area, EditorWindow. When elements are rendered, the type of item is checked and the corresponding type of element is rendered on the page.

Once the components are created in the editing area, the user can edit, reposition, and resize them. After the user is satisfied with the created page, they can click "Generate Page" on the sidebar to send the page to the back end for translation and HTML page creation.

4.1.1 React-grid-layout. The editing area's most important features, such as dynamic component position and size, are realized by the NPM Package react-grid-layout. The entire editing area is managed by EditorWindow which relies on the underlying implementation of react-grid-layout. The component state stores position and index of each component instance. Each component instance is assigned a unique ID, which cannot be reassigned to new component instances even if an old component instance is removed from the editing area. This helps prevent errors that may arise due to duplicate component instance IDs.

The addition and deletion of components in the editing area are realized by changing the state of the EditorWindow component. When initialized, EditorWindow contains a list to store the key, position, and type of all moveable rectangles in the editing area. When a new rectangle is added, the button calls a function to append a new item to the list. The change in list content triggers EditorWindow to re-render itself. During the rendering, EditorWindow iterates through the items in the list and creates a rectangular component for each item. Removing elements works in a similar way. Instead of appending an item, the triggered function removes the corresponding item from the list, and EditorWindow is re-rendered to reflect the change.

The React component managing the editing area also stores information about component instances, such as images of image and carousel instances, in its state. This implementation allows the component to have direct access to information of each component instance so that it can send them conveniently to the back end when the user finishes creating the page. Subsequently, functions that modify individual component instances are defined in EditorWindow and passed to the instances as props.

4.1.2 Text Box. The text box content creators add to the editing area should support input and styling options, such as font type, size, and style. In other words, we need to implement a rich-text editor within a text box component. Again, a number of rich-text editors that can be imported to projects via NPM are available, and after considering the quality and code maintenance of different rich-text editor components, we decided on a package called react-quill, a React implementation of Quill.js. Quill.js is a rich text editor that natively works by its JavaScript library, and react-quill offers a convenient way for developers to import the editor to React applications.

We made modifications to Quill.js' original implementation to persist the content of text boxes. With the default implementation, whenever new component instances are added to the editing area, the editing area is re-rendered and text boxes lose their content. Although React only re-renders the relevant components in a page, adding new component instances causes positional information in EditorWindow's state to change, and adding or removing component instances triggers the re-rendering of all the elements in the editing area. Quill.js' original implementation does not store the content of text boxes in its state. While storing the content in Quill.js' own React component state addresses the issue, we eventually stored text information of text box instances under EditorWindow so that they are accessible when EditorWindow needs to send all page information to the back end.

4.1.3 Navigation Bar. When a user clicks "Toggle Navbar", a window pops up and allows them to choose a color for the navigation bar. The color picker is implemented using NPM's react-color package and allows users to either pick a color from the color wheel or specify a color by its hex code. To add regular or dropdown buttons to the navigation bar, users can click "Add Navbar Button" or "Add Navbar Dropdown". The clicking triggers a popup window, displaying the react component that takes users' input and passes the information to EditorWindow. All navigation bar button information is stored in an array in EditorWindow.

To allow users to move around and delete the created navigation bar buttons, the navigation bar is also based on react-grid-layout. EditorWindow sends all the button information to the navigation bar component as props, and the navigation bar component uses this information to generate dynamic, moveable buttons.

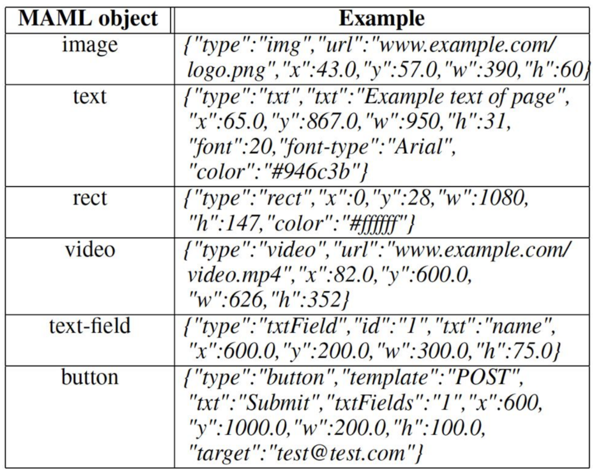
4.1.4 Images and Carousels. When users click "Add Pic", EditorWindow creates a new element of type "picture" in its state's item list and causes a re-render of the editing area. A component instance of UploadAndDisplayImage would appear in the space, featuring a button for image selection and upload. After the user uploads the image, the image file is stored in EditorWindow's state and passed to the UploadAndDisplayImage component for display.

When users click "Add Carousel", the CarouselEditor component appears in a popup window, allowing users to upload images to the editor. When the user is satisfied with uploaded images and clicks "create", CarouselEditor calls a function to store image files in EditorWindow's state and adds a carousel item to EditorWindow's state. When EditorWindow re-renders, this newly added carousel item is rendered with the image files already stored in the state.

4.2 Translation

When the user clicks "Generate Page", they key in a page name and EditorWindow calls the "generatePage" function to generate MAML objects based on the component's state as well as positional information provided by the DOM.

MAML stores the most essential information needed for generating a webpage. It divides a page into fundamental types of components such as text, image, and buttons, and records their information such as text content, position, width, and height.



MAML object	Example
image	*{"type":"img","url":"www.example.com/ logo.png","x":43.0,"y":57.0,"w":390,"h":60}*
text	*{"type":"txt","txt":"Example text of page", "x":65.0,"y":867.0,"w":950,"h":31, "font":20,"font-type":"Arial", "color":"#946c3b"}*
rect	*{"type":"rect","x":0,"y":28,"w":1080, "h":147,"color":"#ffffff"}*
video	*{"type":"video","url":"www.example.com/ video.mp4","x":82.0,"y":600.0, "w":626,"h":352}*
text-field	*{"type":"txtField","id":"1","txt":"name", "x":600.0,"y":200.0,"w":300.0,"h":75.0}*
button	*{"type":"button","template":"POST", "txt":"Submit","txtFields":"1","x":600, "y":1000.0,"w":200.0,"h":100.0, "target":"test@test.com"}*

Figure 3: Sample MAML Objects

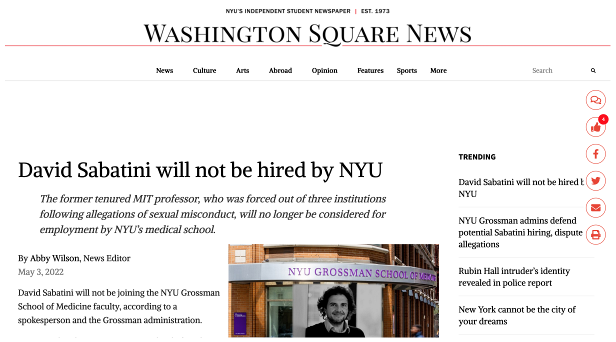
When the "generatePage" function is called, EditorWindow's state is inspected, as it stores a list of items in the editing area. While some information is stored in the state, some other information needs to be accessed through the DOM. When each element is created, they are given an HTML ID linked to their unique ID in EditorWindow's state. This allows us to use query selectors in JavaScript to pinpoint the item's corresponding HTML element and get relevant information such as position and style.

After the "generatePage" function acquires all the necessary information for an item, it generates the MAML representation in the form of a JSON object string and appends it to the data that needs to be sent to the back end. After all data is gathered, the front end sends the MAML objects JSON string and all necessary files such as images to the backend. Before the files are sent to the back end, they are renamed with the unique ID of their items so that items and their files can be easily matched in the back end.

Upon receiving a request from the front end, the back end store all files under the page name specified by the user. It then runs a Python script to parse the MAML objects JSON string. The script creates an HTML page by adding relevant elements to a string and adding more elements as it parses the MAML objects. It uses the position, style, and content information from the MAML objects to create corresponding HTML elements. After the parsing, the script writes the string into an HTML file which is ready to be displayed by browsers.

5 PERFORMANCE EVALUATION

To compare the loading time of the MAML pages with existing webpages, we chose 20 existing webpages to create MAML pages that have similar content and style. Given the limited functionalities supported by the MAML editor, we ended up creating 17 MAML pages that are able to produce most of the content of the original webpages.



NYU'S INDEPENDENT STUDENT NEWSPAPER | EST. 1973

WASHINGTON SQUARE NEWS

News Culture Arts Abroad Opinion Features Sports More Search

David Sabatini will not be hired by NYU

The former tenured MIT professor, who was forced out of three institutions following allegations of sexual misconduct, will no longer be considered for employment by NYU's medical school.

By Abby Wilson, News Editor
May 3, 2022

David Sabatini will not be joining the NYU Grossman School of Medicine faculty, according to a spokesperson and the Grossman administration.

TRENDING

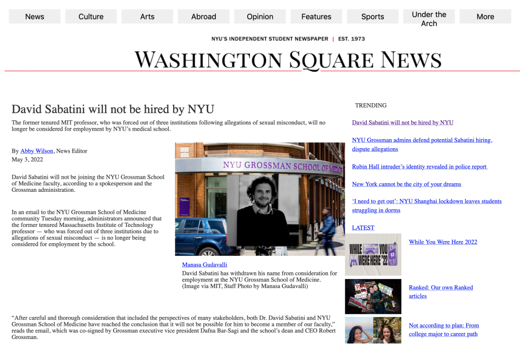
David Sabatini will not be hired b NYU

NYU Grossman admins defend potential Sabatini hiring, dispute allegations

Rubin Hall intruder's identity revealed in police report

New York cannot be the city of your dreams

Figure 4: An Existing Webpage



News Culture Arts Abroad Opinion Features Sports Under the Arch More

NYU'S INDEPENDENT STUDENT NEWSPAPER | EST. 1973

WASHINGTON SQUARE NEWS

David Sabatini will not be hired by NYU

The former tenured MIT professor, who was forced out of three institutions following allegations of sexual misconduct, will no longer be considered for employment by NYU's medical school.

By Abby Wilson, News Editor
May 3, 2022

David Sabatini will not be joining the NYU Grossman School of Medicine faculty, according to a spokesperson and the Grossman administration.

In an email to the NYU Grossman School of Medicine community Tuesday morning, administrators announced that the former tenured Massachusetts Institute of Technology professor — who was forced out of three institutions due to allegations of sexual misconduct — is no longer being considered for employment by the school.

Manasa Gudavalli
David Sabatini has withdrawn his name from consideration for employment at the NYU Grossman School of Medicine. (Image via MIT, Staff Photo by Manasa Gudavalli)

"After careful and thorough consideration that included the perspectives of many stakeholders, both Dr. David Sabatini and NYU Grossman School of Medicine have reached the conclusion that it will not be possible for him to become a member of our faculty," reads the email, which was co-signed by Grossman executive vice president Dafna Bar-Sagi and the school's dean and CEO Robert Grossman.

TRENDING

David Sabatini will not be hired by NYU

NYU Grossman admins defend potential Sabatini hiring, dispute allegations

Rubin Hall intruder's identity revealed in police report

New York cannot be the city of your dreams

'I need to get out': NYU Shanghai lockdown leaves students struggling in dorms

LATEST

While You Were Here 2022

Ranked: Our own Ranked articles

Not according to plan: From college major to career path

Figure 5: Corresponding MAML Page

To produce a fair comparison of loading time, as the MAML pages are served from a server in New York University Abu Dhabi, the original webpages are cloned and hosted on a proxy server at the university. Firefox browser was used to inspect the loading time of webpages, with caching disabled.

Time taken for both DOMContentLoaded and Load event to fire are measured. All measurements are done at the same time of the day with a Wi-Fi network connection, and each page is loaded five times to produce an average loading time. Given that the Load event takes into consideration the time to fetch resources like images, we believe that it is a better measurement for page loading time. Given that one of the 17 pages took extremely long to fire the Load event, we chose to use data of 16 pages. On average, MAML pages reduce page loading time by 93.5%.

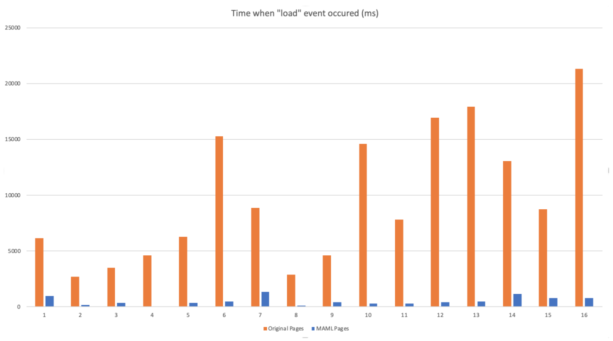


Figure 6: Loading Time Comparison

6 LIMITATIONS AND FUTURE PLAN

As the MAML editor supports limited webpage components, some elements cannot be recreated using the MAML editor at the moment. For example, webpages can have complex, multi-layered dropdown menus and this is not supported by the MAML editor. The editor also does not support elements such as icon buttons and Twitter post plug-ins. Moreover, many websites display images in webp format and this is not commonly used by users who may use the MAML Editor. The temporary solution during the webpage-making process was to convert webp images into jpeg format, which causes the file size to be different and introduces more variables to the page loading time comparison.

Given that the created MAML pages do not have all the functions of the original webpages, the significant reduction in page loading time is partly because the MAML pages are not fully functional. Nevertheless, the MAML pages achieve the essential function of displaying texts, images, and links for users while taking a significantly shorter amount of time to load. This shows that generating simplified webpages that sacrifice some functionality can be a way to provide accessible online resources to communities with poor internet infrastructure and limited mobile phone processing power. The next step will be to expand the range of components supported by the editor so that pages with more complete functionalities can be produced. In the future, the design of the user interface can be improved, and we also plan to implement back end functionalities such as storing editor states for users to resume editing when they log in again.

7 CONCLUSION

Continuing the effort to reduce page loading time by MAML, this project aims to create an editor that allows people to create lightweight MAML pages on their own. To find a balance between minimum webpage complexity and interactive feature compatibility, we researched the essential JavaScript-dependent web page features. The resulting editor supports a number of components and has produced webpages that have similar content and style as the original webpages while loading in a much shorter time. This shows us the viability of generating simplified webpages for people with limited internet access. With future expansion on the functionalities supported by MAML Editor, it will offer a way for people in developing regions to create and share webpages that are fast to load and easily accessible.

REFERENCES

[1] Daniel An. 2017. Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed. https://www.thinkwithgoogle.com/intl/en-ca/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/

[2] Michael Butkiewicz, Harsha V. Madhyastha, and Vyas Sekar. 2011. Understanding Website Complexity: Measurements, Metrics, and Implications. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (Berlin, Germany) *(IMC '11)*. Association for Computing Machinery, New York, NY, USA, 313–328. https://doi.org/10.1145/2068816.2068846

[3] Catalina M. Danis, Jason B. Ellis, Wendy A. Kellogg, Hajo van Beijma, Bas Hoefman, Steven D. Daniels, and Jan-Willem Loggers. 2010. Mobile Phones for Health Education in the Developing World: SMS as a User Interface. In *Proceedings of the First ACM Symposium on Computing for Development* (London, United Kingdom) *(ACM DEV '10)*. Association for Computing Machinery, New York, NY, USA, Article 13, 9 pages. https://doi.org/10.1145/1926180.1926197

[4] Rodérick Fanou, Gareth Tyson, Pierre Francois, and Arjuna Sathiaseelan. 2016. Pushing the Frontier: Exploring the African Web Ecosystem. In *Proceedings of the 25th International Conference on World Wide Web* (Montréal, Québec, Canada) *(WWW '16)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 435–445. https://doi.org/10.1145/2872427.2882997

[5] Fie Velghe. 2013. Literacy Acquisition, Informal Learning and Mobile Phones in a South African Township. In *Proceedings of the Sixth International Conference on Information and Communication Technologies and Development: Full Papers - Volume 1* (Cape Town, South Africa) *(ICTD '13)*. Association for Computing Machinery, New York, NY, USA, 89–99. https://doi.org/10.1145/2516604.2516615