MAML EDITOR: A Website Editor Supporting a Restricted Subset of JavaScript Functionalities

Gideon Akosah
Computer Science, NYUAD
gaa382@nyu.edu

Advised by: Yasir Zaki, Thomas Potsch

ABSTRACT

There has been a significant increase in web page complexities in recent decades. These complexities, even though are aimed at improving overall user experience, have incurred extra overhead data costs especially to users with low end mobile devices. As well as spending more money on data, users in developing countries with low end mobile devices are not able to enjoy elegant browsing experiences. It has been proven that JavaScript loading and parsing in web browsers have been a main cause of these overheads. Due to these reasons and a host of other reasons, there have been numerous researches and implementations of different tools aimed at improving the overall browsing experience. Potential solutions include optimizing websites' loading process, re-implementing interactive features using more efficient programming languages, and limiting JavaScript libraries and functionalities supported by web editors or browsers. An existing approach, Mobile Application Mark-up Language (MAML), represents web pages in a simple format at the cost of inhibiting all JavaScript. This project explores different approaches to bring interactive features to MAML pages. Previous research by my partner shows that simply recompiling JavaScript into Go and WebAssembly does not improve page loading time. As such, the aim now is to realize MAML's interactive features by supporting a restricted subset of JavaScript functionalities. The editor consists of a React-based front end and a back end implemented by Flask. Web pages created by users are transported to the back end in the format of MAML. The back end in turn translates the MAML objects to HTML pages with minimum JavaScript dependency. This report discusses the progress of the design and implementation of the MAML editor. The aim of this editor is to reduce the overhead costs associated with web pages by allowing content creators to incorporate only the most basic and important JavaScript functionalities of the websites. For this semester, I extended the functionalities of the editor by implementing a feature that allows users to save, retrieve and reedit previously created pages.

KEYWORDS

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

Reference Format:
Gideon Akosah. 2022. MAML EDITOR: A Website Editor Supporting a Restricted Subset of JavaScript Functionalities. In *NYUAD Capstone Project 1 Reports, Spring 2022, Abu Dhabi, UAE*. 6 pages.

1 INTRODUCTION

Web page performance has become a major headache to both developers and users in recent times. New technologies aimed at improving user web experiences have added different layers of complexities to our web pages. These complexities inadvertently affect the performance of the websites. Even though there has been significant improvement in the general internet infrastructures due to increases in bandwidth, highly improved link media such as fiber optics, 4G, 5G and a host of other solutions aimed at providing supersonic internet access, there are still barriers affecting web page performances. These barriers are not limited to internet infrastructure but may include the browsing media and developer's decisions with respect to web page design. The effects of these barriers are often prevalent in developing countries where people do not have access to mobile phones with high computing power, constant network connections etc [1]. Page load times in underdeveloped countries can be as long as 60 seconds, and more than half of users will abandon a page if it takes more than 3 seconds to load. Bad user experience can affect businesses tremendously. It is estimated that 53% of users leave the site if it takes more than 3 seconds

This report is submitted to NYUAD's capstone repository in fulfillment of NYUAD's Computer Science major graduation requirements.

جامعة نيويورك أبوظبي
NYU ABU DHABI

Capstone Project 1, Spring 2022, Abu Dhabi, UAE

for the website to load [2]. Also, research has found a strong correlation between website speed and revenue, viewability, bounce rate, session duration [3]. As a result, various researchers have designed and implemented different solutions aimed at improving the performance of web pages [4]. These solutions are often targeted to a different aspect of web page infrastructure. While some of these solutions are aimed at reducing the amount of JavaScript processed in client's browsers, other solutions are aimed at reducing the latency of client-server interactions. JavaScript parsing has been determined to be a major source of overhead costs in web page loading times. This is mainly due to the recursive nature of script loading inline HTML. The numerous network requests made to load all these scripts invariably add huge delays as well. This project aims to reduce this problem by creating an interactive web editor which supports only a few JavaScript functionalities. The editor is also based on MAML which is a lightweight markup language that inhibits JavaScript and represents web pages in a simplified manner. Since interactivity is still an indispensable feature of many web pages, this format ensures that our editor presents a significantly faster loading times while sacrificing only a minimal and often unimportant JavaScript functionalities for an improved browsing experience. This work is a collaboration between myself and a former capstone student, who worked mainly on the front end. He implemented the web editor window that allows creators to reproduce simpler versions of existing web pages. My job was mainly focused on the back end. I helped in converting the MAML objects created in the front end to their corresponding lightweight HTML in the back end as well as other server and database tasks. To extend the features of this editor and to get an end-to-end functioning product, I implemented a feature that allows users to reload older pages and resume editing those pages. Users can now save their works and retrieve them at anytime to resume editing. This feature will enhance user experience as they would not have to complete editing in one sitting. They can always save their works and come back to them later.

2 BACKGROUND RESEARCH

Because we want to keep the use of JavaScript in editor-generated websites to a minimum, the first part of the project, which was mainly performed by my peer before I joined him, was to look into the most prevalent website features that use JavaScript. The aim of this exercise was to determine the most prevalent JavaScript features that are absolutely required in most if not all of the top Alexa websites. Our web editor will then support only these features allowing the resulting web sites to be as light as feasible while maintaining the interactive elements that consumers require.

2.1 Preliminary 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 web page editor should support.

2.2 Preliminary 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 Preliminary Analysis Results

After exhaustive research on Alexa's top 100 websites, we came up with 11 top most JavaScript-dependent features that are necessary to implement in our web editor. The features are outlined below:

- Drop-down menu
- Loading of new content when the page reaches the 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.

Although JavaScript supports a vast variety of libraries and has been a significant cause of website loading time, this research demonstrates that most websites share many essential capabilities. As a result, this initial examination reveals that the website editor only requires support for a limited range of JavaScript features. However, reviewing more websites may lead to the discovery of additional interactive features that the website editor should 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 web page editor.

3 DESIGN

The product we are building is similar to other web editor tools like wix, WordPress etc. However the main difference is that ours support conversion of the web page components into MAML objects. Also, our web editor only supports a limited number of JavaScript features. Because of these differences, our design decisions, even though may slightly mimic those used in these products, have to incorporate other new aspects that supports our own expectations.

3.1 Expected Product

We are anticipating an end product that looks similar to all the other web editing tools on the market in terms of appearance but differ in functionalities. In particular, text editing, image upload, element positioning, and the inclusion of interactive elements like buttons and side menus should all be supported by the editor. Furthermore, the editor should permit content producers to see the website in non-editable mode. Additionally, the editor should export built web pages in a format that is compatible with existing MAML implementations. As already implemented by my partner, our web editor features a main editor window where the components are placed. It also features a side menu which contains buttons that are used to select the specific component a user wants to add to the editor window. Each button represents a different functionality that may be implemented or not depending on user's preference. 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. As stated earlier, my main job was to set up back end functionalities for this web editor as well as extend the functionalities of the front end. Last semester, I created features to automatically generate corresponding HTML scripts from MAML objects and images received from the front end. These pages can then be served to the user in case they would want to continue their page creation or editing. Also, I setup up user creation/login options as well as authentication to ensure that users get personalized experiences when using the product. I created endpoints to receive images and maml documents while creating personalized paths on the local disk of the server to store contents for users separately. Also, I designed and created all the database tables, models and corresponding queries related to the project. Lastly, I implemented a separate page where everyone can find all previously generated pages and visit these pages accordingly. This semester, the goal was to keep building and expanding the functionalities of the web page editor. I designed and created a feature that allows users to save unfinished work and return back to them later. This feature will enhance user experience as it will no longer require users to complete their work in one sitting.

3.2 Development Framework

3.2.1 React. Given the functionalities the editor should support, my partner decided to develop the front end of the editor using React. React is a JavaScript library and a popular solution to build user interfaces for web-based applications. React treats a web page 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 web page editor.

3.2.2 NodeJs/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 web page editor, as it supports the aforementioned functionalities that are going to be useful when building a web page editor.

3.2.3 Python/Flask. The product is expected to be used by significant number of people. As such I needed to choose a

back end framework that is robust, fast and hugely supported. For this purpose, I chose python with Flask as my main development framework. Flask also has an extended rest API library called restapix. This library supports communication between the front end using the rest protocol. As most of the data, that is going to be exchanged is in the form of Json data, using the rest protocol seemed as an efficient way of designing the back end controller. Flask also provides an efficient means of exchanging files using the request.files library. As our project involves a lot of file transfer, we needed flask to provide better experience. Another advantage of using flask is that it supports the use of sqlalchemy which is a robust interface for interacting with the SQL database. SQL was used as the main database tool. SQL was chosen because it is fast and it provides a simplified client for database requests. The back-end was designed using the Model-View-Controller (MVC) architecture.

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 the main editor window where they can start creating their pages. This page consists a "save" option that allows users to save their works midway. It also features a "view pages" option that allows users to regenerate old saved pages. Users can click the corresponding links and they will retrieve all the assets that enable them to visit an old page.

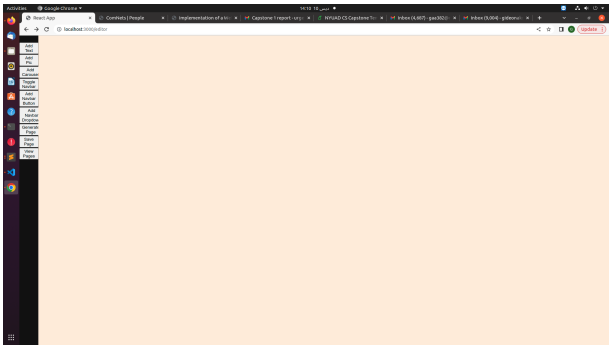

Figure 1: An image of the editor window

4.1 RESUME EDITING WINDOW

This is the window that allows user to regenerate older pages. That takes a user here are in the editing window. By clicking on corresponding links, users can regenerate the pages saved pages. Hence this window provides dynamic contents based on the page link that was clicked.

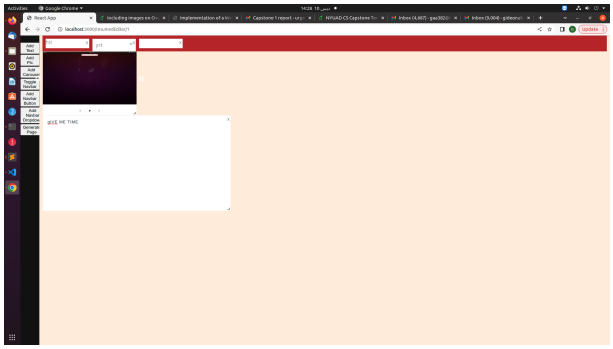

Figure 2: An image of the resume editor window

4.2 DATABASE MODELS

I started the back end by designing the models of the database. It was established by preliminary analysis that we needed two models: User Model and Pages Model. The user model, which had user id, user name, email and password fields, is to be used to store user details of all registered users of the platform. The page model, which had the page id, and user id fields, is for storing all the pages generated by each particular user. The user Id field in the page table establishes a foreign key relation between the pages and the users. This is very useful when regenerating pages for users to continue their editing. I included another model: Post model. This model allows users to save unfinished work. So it has one column where maml objects are saved. This is called "content" column. The design is such that pages are regenerated using their corresponding maml objects through reverse engineering. So this column is needed for that purpose. I added another column called the textMap where I saved all the texts corresponding to a particular page. I use these texts for regeneration purposes. Lastly the "post" table has userId and postId columns that are essentially needed for customization.

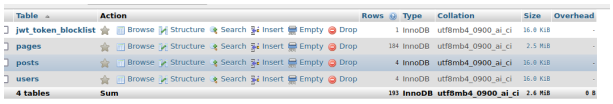

Table	Action	Rows	Type	Collation	Size	Overhead
jwt_token_blocklist	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
pages	Browse Structure Search Insert Empty Drop	184	InnoDB	utf8mb4_0900_ai_ci	2.5 MiB	-
posts	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
users	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
4 tables	Sum	193	InnoDB	utf8mb4_0900_ai_ci	2.6 MiB	0 B

Figure 3: An image of all the tables in the database

4.3 REST API ENDPOINTS/CONTROLLER

The project currently has five rest api endpoints responsible for establishing communication between the front end and the back-end. The first endpoint is the user sign up endpoint.

Figure 4: An image of the "posts" table

This allows users to register for an account in the system. This endpoints does not require any authentication hence does not need any token permissions. The second endpoint is the user sign in endpoint which allows registered users to log into their accounts. It also does not require authentication. The most important endpoint is the page upload endpoint. This endpoint receives MAML objects of the page generated by the user as well as images and any other files in the web page. They are then stored on the local disk of the web server using the specific user data to describe the path on the disk. This helps to protect data integrity of each user. The MAML objects are then used to generate the corresponding .html files of each page and again stored on the local disk of the server. The other endpoints are responsible for regenerating user pages to enable users to reedit their pages as well as listing all available pages in the database. For the feature created this semester, we used three main endpoints. The first one, addPost endpoint, is called when the user clicks the save button on the editor window. This endpoint receives maml objects corresponding to the created post as well as all the images. It then stores the images on the local disk path corresponding to each user and stores the maml objects and the texts in the database. It then runs a function to transfer all the images into the apache server to enable ease of image retrieval. The other two endpoints allows the user to either retrieve all saved posts or single saved post.

4.3.1 VIEW. : I needed not to implement much for this part because the front end took a major responsibility for it. However, I implemented a view page using flask templates library to display all the available users with their corresponding pages. With this page, one can visit any page that has been generated on this platform.

4.4 PAGE REGENERATION

This is the core of the resume editing feature. I needed a way to regenerate react components based on data saved in the backend. There are multiple ways this can be achieved. However, after extensive analysis, I settled on using the maml objects for each page. Maml provides descriptive data for each component of our webpage. Since it was already being generated in our project, it seemed like an easier choice for the implementation of the feature. So the function that regenerates the pages iterates through all the maml objects and recreate the react components based on the type and other features of the object. Since all these process had to take place automatically, I used the react state management tool "componentDidMount" to realize this function.

5 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 As discussed earlier, to develop a website editor for MAML format, the editor needs to limit JavaScript code in created pages at a minimum level and output pages in MAML-compatible formats. My partner last semester focused on implementing the carousel functionality. This semester, I extended the functionalities of the editor by developing a window that allows users to regenerate old pages and continue their editing work. The plan for the future must be ways of incorporating the other 10 JavaScript functionalities that were discussed earlier in order to create a complete system.

6 CONCLUSION

Continuing MAML's effort to reduce page loading time, this project aims to expand MAML by supporting a selected subset of JavaScript. We researched the essential JavaScript-dependent web page features to balance minimum web page complexity and interactive feature compatibility. We have implemented a huge part of the back-end, exported web pages into MAML format and implemented the carousel functionalities. We hope to expand these efforts in the future to support more JavaScript functionalities and in the end provide a better browsing experience for people in regions with poor internet infrastructure.

REFERENCES

[1] 2016. Use of smart phones in developing regions. https://www.pewresearch.org/internet/2019/03/07/use-of-smartphones-and-social-media-is-common-across-most-emerging-economies/. [Online; accessed 6-May-2021].

[2] 2019. Mobile page speed new industry benchmarks-bounce v vs loadtime. https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks-load-time-vs-bounce/. [Online; accessed 6-May-2021].

[3] 2019. Page speed report. https://unbounce.com/page-speed-report/. [Online; accessed 6-May-2021].

[4] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/netravali