Islamic University of Technology (IUT)

Systems and Software Lab (SSL)

Department of Computer Science and Engineering (CSE)

Validating the Use of Git-based Content Management System as a Component of Front End Web Architecture

Authors

Sadman Saadat - 180042126

Md. Jishan Anam - 180042129

Khalid Masum - 180042133

Supervisor

Dr. Hasan Mahmud

Associate Professor

Dept. of CSE

Co-Supervisor

Dr. Md. Kamrul Hasan

Professor

Dept. of CSE

A thesis submitted to the Department of CSE

in partial fulfillment of the requirements for the degree of

B.Sc. in SWE

Academic Year: 2021-22

May - 2023

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Sadman Saadat, Md. Jishan Anam and Khalid Masum under the supervision of Professor Hasan Mahmud, Associate Professor of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Sadman Saadat

Student ID - 180042126

Md. Jishan Anam

Student ID - 180042129

Khalid Masum

Student ID - 180042133

Supervisor:

Dr. Hasan Mahmud

Associate Professor

Systems and Software Lab (SSL)

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

Co-Supervisor:

Dr. Md. Kamrul Hasan

Professor

Systems and Software Lab (SSL)

Department of Computer Science and Engineering (CSE)

Islamic University of Technology (IUT)

Acknowledgement

We would like to express our grateful appreciation for **Munir Hossain**, Senior Software Engineer, Infobox for being our adviser and mentor. His motivation, suggestions and insights for this research have been invaluable. Without his support and proper guidance this research would never have been possible. His valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to him.

We are also grateful to **Dr. Kamrul Hasan**, Professor,Department of Computer Science & Engineering, IUT and **Dr. Hasan Mahmud**, Associate Professor, Department of Computer Science & Engineering, IUT for their valuable inspection and suggestions on our proposal of Git Based CMS as Front End Web Architecture.

Abstract

This research paper introduces an innovative concept that involves utilizing a Git-based Content Management System (CMS) as a bridge between a server-side controller application and static files. The primary objective is to expedite the delivery of static files by eliminating the need for the server to re-render them whenever they are requested by the frontend. This architectural design pattern particularly targets data that requires occasional modifications but does not necessitate frequent updates.The proposed approach is subjected to rigorous testing and validation against three common challenges encountered in web development: response performance, scalability, and information validation. Throughout the evaluation process, the research successfully demonstrates that this new architectural approach surpasses the traditional server-side rendering method across all three criteria, specifically in scenarios where there is a high volume of read-heavy operations. At the same time, it is important to note that this improvement in performance comes at the cost of reduced efficiency when dealing with write-heavy websites, which is supposedly because of too many file-system operations done by the CMS in case of write-heavy use cases.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

In an http server, web server sends data as a response to the http request, where the data is always rendered, rendered as a string, as a JSON[25] or as an HTML or any other format in case of dynamic content as by definition, dynamic contents are subjected to changes. However static contents are pre-rendered and are sent on demand, directly from file system. Therefore, static websites are very well performing and easily scalable. However, this comes with a fundamental problem, that static websites are unchangeable.

Any solution to this problem is highly advantageous. As the first five seconds of page-load time have the highest impact on conversion rates [32] and static sites leads to the best of website loading time. At the same time there are many websites that are read heavy, but the content changes often. These websites can not be implemented with static website, because their changing nature, hence these types of websites can not harvest benefits of a static website. In this work, we tried to show how using a git based flat file Content Management System (CMS) can be used to regain some of the advantages of a static website.

1.2 Problem Statement

The primary goal of this thesis is to find out whether a git based CMS can be used to regain some of the benefits of static websites in case of dynamic websites. These benefits are based on three criteria. Load time improvement, Scalability and Information validation. Ideally we would like to compare with many other web development related problems but web development being a vast area, we have focused on these three criteria as many websites are built for them. And through this research we find out how effectively these problems are solved or not solved and when to use or not to use this architecture design pattern.

1.3 Motivation and Scope

Web architectures and design pattern are very important in software engineering. The primary purpose of them is to categorize different problems into problem categories that have similar solutions. Likewise we are validating a web architecture design pattern that is designed to solve problems concerning dynamic websites that want the benefits of static websites.

Let us consider a scenario. Let there be an e-commerce website has a "Offers" section. This section is viewed by every user, yet the data comes from the database, and rendered every time as the data are changed frequently. Creating a high load time and an unnecessary pressure on the database management system.

This kind of scenario can be avoided with the proposed architecture, where we decouple data rendering from the database, being able to ensure faster serving of viewables.

Git does not inherently supports CRUD operations (Create Read Update Delete) so we will have to use a layer for making crud operations. This is the CMS layer. Recently emerging git based CMS has brought many new use cases under consideration. The ability to modify the page content without using external data source is helpful in implementing dynamic-like websites without using a dynamic server. With this advantage, we may introduce a new architecture that improves several metrics of a website.

At the same time, this type of research is not mainstream yet. This research opens scope to many interesting areas, like information validation via block chain using git and write time enhancement via log structured file-systems [29].

In summary, the motivation can be shown in following points:

- Read heavy websites with frequent data changes exists.
- Scalability and information validation are often major issues for website creation.

- With the emerge of git based CMS some dynamic logic can be shifted to static logic while giving the features of being dynamic.
- Finally, this opens a huge scope of research regarding content loading time.

1.4 Research Challenges

The research faced a significant challenge stemming from the limited amount of literature available on the specific issue being investigated. This dearth of literature encompassed both scholarly research works and published articles, leaving a gap in the knowledge base surrounding the topic. However, by delving into older literature and connecting relevant pieces of information, it was possible to establish a foundation for the claims and assertions made throughout the research.

The lack of literature also extended to the absence of peer-reviewed web development systems or git-based content management systems that could serve as benchmarks for comparison. This posed a hurdle in terms of directly evaluating the proposed solutions and their effectiveness. As a result, the research had to rely on implementing solutions based on reasonable logic and assumptions, introducing additional variables that could potentially influence the outcomes and comparisons drawn.

Furthermore, gaining a comprehensive understanding of existing content management systems (CMS) necessitated a meticulous examination of their source code. This deep dive into the inner workings of various CMS platforms enabled a thorough assessment of their strengths and weaknesses. Through detailed case studies, the research aimed to identify the advantages and disadvantages of these systems and leverage the insights gained to develop a model for a git-based CMS architecture. This model aimed to address the limitations identified and explore the potential benefits of the proposed approach.

The lack of literature also extended to the absence of peer-reviewed web development systems or git-based content management systems that could serve as benchmarks for comparison. This posed a hurdle in terms of directly evaluating

the proposed solutions and their effectiveness. As a result, the research had to rely on implementing solutions based on reasonable logic and assumptions, introducing additional variables that could potentially influence the outcomes and comparisons drawn.

At the same time, existing CMS had to be thoroughly examined from within their source code in order to understand their advantages and disadvantages. These case studies are carried out in order to understand and develop a model git based CMS that can be used to determine the usefulness of the proposed architecture.

1.5 Thesis Outline

In Chapter 1, we have provided a concise and precise overview of our study, setting the stage for the subsequent chapters. In Chapter 2, we delve into an extensive literature review, exploring and analyzing existing research and developments relevant to our study. This comprehensive review aims to establish the current state of knowledge in the field and identify gaps or areas for further investigation.Chapter 3 serves as a crucial section where we present the core elements of our proposed method. We outline the proposed architecture and delve into the intricate details of implementing our model git-based content management system (CMS). By providing a thorough insight into the working procedure of our proposed method, this chapter aims to elucidate the underlying mechanisms and operational aspects.

Moving forward to Chapter 4, we showcase the results obtained from the successful implementation of our proposed method. We conduct a comparative analysis, evaluating the performance and effectiveness of our approach against relevant benchmarks or existing solutions. This analysis aims to provide empirical evidence and substantiate the claims made throughout our study. Lastly, the concluding segment of our research encompasses a comprehensive compilation of references and credits, acknowledging the sources and individuals whose work and contributions have influenced and supported our study.

2 Literature Review

2.1 Dynamic Vs Static Content

Tomiša, Milković, and Čačić in their work "Performance Evaluation of Dynamic and Static WordPress-based Websites" shown that The static version of a dynamic WordPress-based website gives better performance for client-side and server-side website operation processes [20]. They conducted this experiment is using a dynamic WordPress-based website and a static version of that website. If a website takes too much time to load, the client will not be satisfied. From our inspection into wordpress [30] We know that server needs to load the core files, custom theme files, get data from related database files for websites running on WordPress. In any dynamic website, when any changes occurs in the website, the user can't get the updated content until user's web browser gets the complete front-end code from server. For better performance a static version of that dynamic website can be generated. Any time the connected database is changed, a static website must be produced again because WordPress needs to maintain its CMS features. After the environment setup the dynamic website is deployed and also the static version of that website is generated using WP2Static plugin. The performance testing was done by Apache Bench program. 1000 HTTP requests are sent to the URL, 10 concurrent requests at a time. From the result we can see that the static version gives better performance compared to the actual dynamic website.

Metrics	Dynamic Wordpress-based Website	Static Version
Requests Per Second	6.16 [#/sec] (mean)	2218.58 [#/sec] (mean)
Time Per Request	1622.159 [ms] (mean)	4.507 [ms] (mean)
Transfer Rate	833.20 [Kbytes/sec] received	290843.98 [Kbytes/sec] received

Table 1: "Performance Evaluation of Dynamic and Static WordPress-based Websites" Performance

2.2 Web Architecture

2.2.1 Three Tier Architecture

Three-tier architecture is a software design pattern that divides a system into three distinct layers, each with its own specific responsibilities. The presentation layer is the user interface that the user interacts with, and it is responsible for displaying information to the user and accepting input from the user. This layer is typically implemented using technologies such as HTML, CSS, and JavaScript. The application logic layer is the central processing layer that handles the business logic and interacts with the data storage layer. It is responsible for tasks such as data validation, calculation, and manipulation, and it is usually implemented using a server-side programming language like PHP or Python. The data storage layer is responsible for storing and retrieving data, and it is usually implemented using a database like MySQL or MongoDB.

Three-tier architecture is a perfect candidate to be redesigned via our proposal. In the book *Distributed Applications Engineering: Building New Applications and Managing Legacy Applications with Distributed Technologies*,Wijegunaratne and Fernandez shows us that separates the different layers and allows them to be developed and maintained independently, improving scalability, security, and maintainability compared to other architectures such as two-tier. This separation of responsibilities also makes it easier to modify and update individual layers without affecting the other layers. For example, if changes need to be made to the presentation layer, the application logic layer and the data storage layer can remain unchanged, which can save time and resources. Overall, three-tier architecture is a widely-used and effective design pattern for building complex systems that require a clear separation of responsibilities and the ability to scale and adapt to changing requirements.

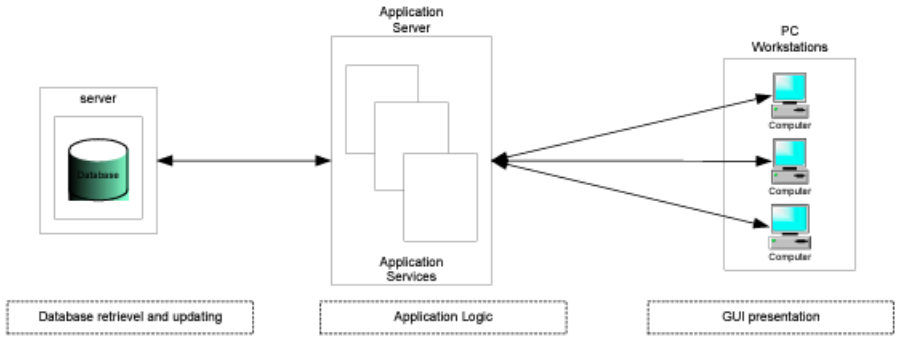

Figure 1: Three-tier Architecture [3]

Even so this architecture has drawbacks of re-fetching data over and over again, forcing the pages to re-render on demand, introducing a huge page load time. This type of rendering the viewables when a page is requested is also called server side rendering (SSR).

2.2.2 Server Side Rendering

Server Side Rendering is a software development method in web based application which handle requests from user in server, then server do some logic and algorithms as business needs. After requests were handled, server send response to user as the result from operation with almost of the rendering process called as multipage application because when user send request, it operated in server not in user device [21][2]

With SSR, when a user requests a web page, the server retrieves the necessary data, generates the HTML content, and sends it to the client. This approach allows the client to receive a fully rendered page that can be displayed immediately, as opposed to CSR where the client receives a basic HTML file and relies on JavaScript for rendering.

The advantages of server-side rendering include faster load times, improved SEO [21]. as search engines can easily index the content, enhanced user experience with

immediate display of content, and better accessibility for users with JavaScript disabled. SSR also ensures compatibility across browsers and devices [22].

However, it's important to consider that SSR may require more server resources and is better suited for static or semi-static content rather than highly interactive applications [19].

2.3 Content Management System

2.3.1 What is content management system

A content management system (CMS) offers a way to manage large amounts of web-based information that escapes the burden of coding all of the information into each page in HTML by hand [9][8]. It is a software application that allows users to manage and publish digital content. CMSs are commonly used for websites and other online platforms, but they can also be used for managing content in other formats, such as documents and images. Some common features of CMSs include the ability to create, edit, and delete content; assign roles and permissions to different users; and track and analyze user activity. Many CMSs also have templates and themes that allow users to customize the design and layout of their content. In conclusion, a CMS provides a central location for managing and organizing digital content, making it easier for users to publish and maintain their content.

2.3.2 What is Git Based Content Management System

Git [16] is a distributed revision control system [11] that is widely used in software development to track and manage changes to source code. A git based CMS is a content management system that uses git as the underlying version control system to manage and track changes to digital content. This allows users to collaborate on content and make changes without overwriting each other's work. It also provides a history of changes, so users can roll back to previous versions if necessary. Git CMSs are often used for websites and other online platforms, and they are popular

because they allow for easy collaboration and version control. Some examples of git CMSs include Netlify[27] and CrafterCMS[27]. Traditional database based CMS has some flaws regarding scalability, performance and security because of the use of traditional Database system. We can easily solve this by using git based CMS. Here we will use file system for storing data and doing other database related tasks.

2.3.3 WordPress

WordPress, known as the world's most popular content management system (CMS) [30], dominates the market with an impressive market share of around 43% [26] At its core, WordPress relies on the PHP programming language and employs a flexible plugin system to manage and display content. This powerful combination allows users to effortlessly handle various aspects of their website's content, including creating, editing, and organizing it. Behind the scenes, WordPress leverages a robust database to store and retrieve data, ensuring efficient content management.When a user requests a specific page, WordPress dynamically generates a PHP version of that page, customizing it based on the requested content and user preferences. This dynamic generation process enables WordPress to deliver a fully functional HTML file to the user's browser, enhancing performance and user experience.

By generating HTML files on the fly, WordPress reduces the server's workload and eliminates the need for processing PHP code with each request, resulting in faster page loading times.One of the key advantages of WordPress is its extensive ecosystem of plugins. These plugins expand the core functionality of WordPress, offering users a wide range of features and customization options. From enhancing security to adding e-commerce capabilities, plugins empower users to tailor their websites to their specific needs, without requiring extensive coding knowledge.Additionally, WordPress boasts a vast community of developers, designers, and enthusiasts who contribute to its growth and evolution. This vibrant community provides support, creates themes and plugins, and continuously improves the platform through updates and enhancements.

While WordPress enjoys widespread popularity as a CMS, it does have certain

limitations. One such drawback is its potential for inefficiency due to the high volume of requests made to the backend. However, it's worth noting that WordPress adheres to open API guidelines for CRUD operations, ensuring a predictable and user-friendly API experience.An interesting aspect of WordPress is its "headless" nature, which implies that it is frontend agnostic. This flexibility allows developers to build the frontend of a website using any technology they prefer, seamlessly integrating it with WordPress through the API. This decoupled architecture provides developers with the freedom to choose the most suitable frontend tools and frameworks for their specific project requirements.

By adopting a headless approach, WordPress expands its scope beyond a traditional CMS, transforming into a robust content platform that can power various applications and experiences. Developers can leverage the extensive capabilities of WordPress for content management while enjoying the flexibility to craft unique and engaging frontend experiences.Despite any limitations, WordPress remains a powerful and widely embraced CMS among developers worldwide. Its vast plugin ecosystem, user-friendly interface, and extensive community support contribute to its enduring popularity. As developers continue to innovate and extend the capabilities of WordPress, it continues to evolve as a versatile and adaptable content management solution for a wide range of projects.

2.3.4 NetlifyCMS

Netlify CMS[27] is a content management system that is built on top of Git, and it follows a similar architecture to the one that we are proposing. For read-only operations and other sections, Netlify CMS follows the guidelines that we have outlined. However, there is a critical flaw in Netlify CMS that needs to be addressed: the API is not defined and does not follow the open API guidelines.

This lack of defined API and adherence to open API guidelines can make it difficult for developers to use and integrate Netlify CMS into their projects. It also makes it more difficult to maintain and update the CMS over time, as there is no clear standard for how the API should be designed and implemented.

In summary, while Netlify CMS has some promising features and follows some of the guidelines that we have proposed, it is important for the developers of Netlify CMS to address the issue of its undefined API and lack of adherence to open API guidelines in order to make it a more viable and sustainable content management system.

2.3.5 CrafterCMS

A cutting-edge content management system (CMS), CrafterCMS [23] distinguishes itself from conventional CMS options by placing a special emphasis on Git-based architecture. Better performance, better load balancing, higher scalability, and effective information validation are just a few benefits of this method.

CrafterCMS uses Git for content management, which is one of its primary distinguishing characteristics. Git is a widely used version control system that makes it possible to track content changes effectively and to collaborate with many users at once. CrafterCMS gives content teams the ability to work continuously, effectively manage modifications, and quickly revert to earlier versions if necessary by utilizing Git. The risk of overwriting or losing important content changes is eliminated by this distributed and decentralized method of content management, making it a solid and dependable solution.

CrafterCMS maintains content on a Git repository as opposed to conventional CMS systems, which frequently rely on relational databases to do so. This essential distinction enables improved performance and scalability. Due to Git's distributed architecture, content can be provided directly from the repository, obviating the need for database calls and lowering system load. This simplified procedure makes it possible for quicker content delivery, which enhances website performance and user experience.

The Git-based methodology also makes information integrity and validation easier. It is simpler to manage changes, identify contributors, and assure data consistency

when each content update is documented as a commit in Git. The system offers a transparent audit trail of all revisions, which makes it easier to identify problems and confirm the authenticity of content changes. For compliance-driven sectors where maintaining data integrity and accountability is critical, this capability is extremely important.

So basically, CrafterCMS's Git-based architecture has a number of benefits over conventional CMS options. Organizations can gain from faster performance, better load balancing, more scalability, and simplified information validation by utilizing Git for content management. These characteristics make CrafterCMS a strong option for companies and content teams looking for a sturdy, effective, and trustworthy CMS system.

2.4 Version Control System: Git

Version control is an essential tool in software development and other fields that involve frequent changes to digital files. It allows users to track the history of changes made to a file or set of files, making it easy to recall specific versions later on. Git is a popular version control system that is widely used by developers around the world. One of the key benefits of Git is its distributed nature, which means that it stores copies of the code on multiple servers and devices, rather than relying on a single central repository. This makes it much less vulnerable to data loss or corruption, as Git is able to detect any problems and alert the user.

In Git, all changes made to the code are recorded as commits, which are essentially snapshots of the code at a particular point in time. These commits are undoable, unless they are rebased, which is a process that allows the deletion of commits. Each commit includes important information such as a checksum, which is a unique identifier that allows Git to verify the integrity of the commit, the name of the author, and the time the commit was made. Git rebase is a powerful tool that allows users to manipulate the commit history and delete commits that are no

longer needed. This can be especially useful when working on large projects with multiple contributors, as it helps to keep the commit history clean and organized.

3 Methodology

In web development, it is common to use a database to store and retrieve data for use in web applications. However, this can be inefficient for read-heavy websites, which receive a large number of requests for the same data. To retrieve the data, the application must poll the database for the desired information, embed the data into a webpage, and then send the webpage to the presentation layer for display to the user. This process must be repeated for every request, which can lead to poor scalability and performance.

One solution to this problem is to use Git, a version control system, to directly manage static files in the presentation layer. Git has the ability to make changes to existing files on a file system, and it can track these changes and insert them into the file as needed. This allows developers to update and modify the content of a website without rebuilding the entire webpage for every request.

However, Git has a limitation in that it can only track the changes to a file, and it cannot differentiate between different contents within the file. To overcome this limitation, we can use a Git-based content management system (CMS) as an additional layer over Git. This Git-based CMS can handle the organization and management of content within the presentation layer, while still leveraging the powerful version control capabilities of Git. By using this "web architecture based on Git-based CMS," we can solve the problem of inefficient data management in read-heavy websites and improve scalability, performance, and security.

In order to effectively implement our proposed method, we will need to follow a set of specific steps. The first step will involve categorizing the various use cases for different projects based on their characteristics. Some processes may be read-heavy,

while others are write-heavy, and some websites may have more relational data than others. Once we have identified the unique attributes of each project, we will be able to filter out the use cases that our methodology is best suited for. After the filtering process is complete, we will proceed to perform our operations on the selected use cases.

All dynamic websites must contain a server side rendering, whether it's a JSON or HTML, and this data is sent to the client. So that the client side may display the information. Hence to validate our proposition of using git based CMS to store already rendered data in order to fast-serve the client, we built a Git Based Content Management System, and two identical website, using MySQL. One uses the CMS to prerender static websites while the other does not.

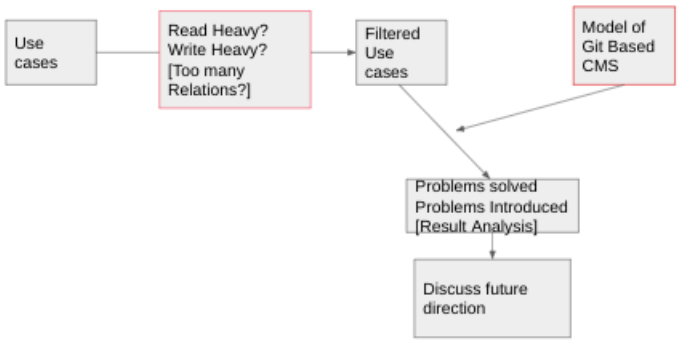

Figure 2: Methodology

In addition to this, there are websites that gets benefited from information validation. For example Passport getting website where previous version of website is very useful. Since this research is about architectural validation, we have dived into this aspect of information validation as well.

Using the websites we tried to figure out the performance based on these three research questions:

- Whether response time is better when git based frontend architecture is used.
- Scalability comparison of both traditional and git-cms based website
- Effectiveness of information validation when git-based cms is used.

3.1 Proposed Architecture

Every website receives files from back-end to function. The file types can be any, but these are the most common:

- HTML
- CSS
- JavaScript
- WebAssembly
- WebGL

Since server side rendering is a must for dynamic websites, instead of rendering again for each request and again we can put a git based CMS for rendered object for keeping track of changes and add new changes. This may decrease unnecessary rendering time.

Therefore we are not changing any file that is being sent/received by the client or the server, we are just embedding a git based CMS into it, hence original functionalities of the website is unchanged while providing a very fast response.

3.1.1 Architecture Comparison

Architecture without CMS introduced:

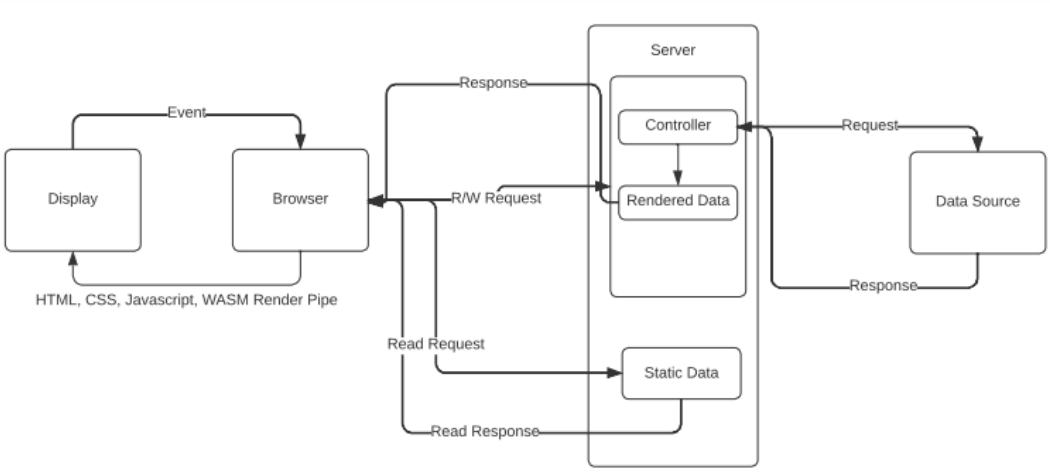

Figure 3: Web Architecture [3], [18], [25]

In this diagram, we have shown the current web architecture based on the works: "Mobile Web Browsers in Android Deriving Reference Architecture," [25] and *Distributed Applications Engineering: Building New Applications and Managing Legacy Applications with Distributed Technologies*. We showed the relevant parts here, on how browser interacts with the user device to render components, where browser gets the instructions for rendering or the server and how the server renders and sends the data to the browser. As you can see, server side has a rendering phase, and the rendered data is sent to the user, making each successful request harder to complete.

In figure 4 we show the current browser architecture in terms of a sequence diagram. It starts with the "Browser" representing a user's web browser, which sends a read or write request to the "Server." The "Server" acts as an intermediary between the browser and the data source.

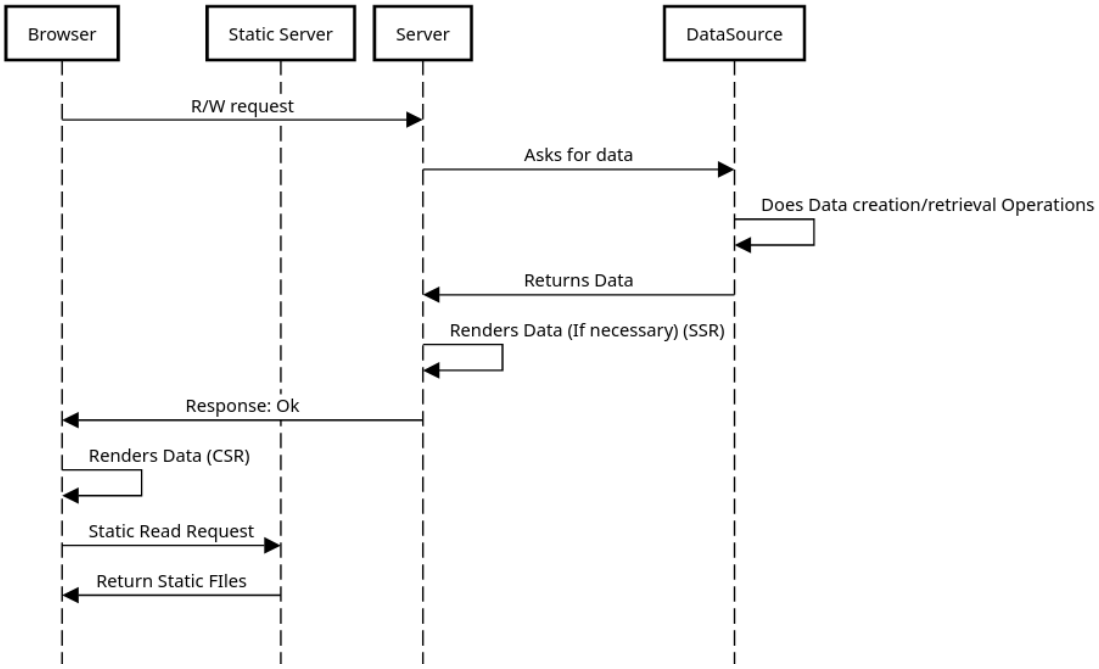

Figure 4: Web-Architecture: Sequence Diagram - Current Architecture

Upon receiving the request, the "Server" communicates with the "DataSource" to ask for the necessary data. The "DataSource" is responsible for creating or retrieving the requested data. It performs the required operations to manipulate the data as needed.

Once the "DataSource" has the data ready, it sends it back to the "Server." The "Server" may then perform additional operations like rendering the data if necessary. This step is known as Server-Side Rendering (SSR), where the server prepares the data in a format suitable for the browser to display.

After the data is processed, the "Server" sends the response back to the "Browser," indicating that the request was successful. At this point, the "Browser" takes over and renders the received data on the user's screen. This process is known as Client-Side Rendering (CSR), where the browser handles the rendering of the data without relying on the server.

Additionally, in some cases, the "Browser" may need to request static files, such as images or stylesheets, directly from the "Static Server." It sends a static read

request to the "Static Server," which then returns the requested static files back to the "Browser."

Architecture with CMS introduced:

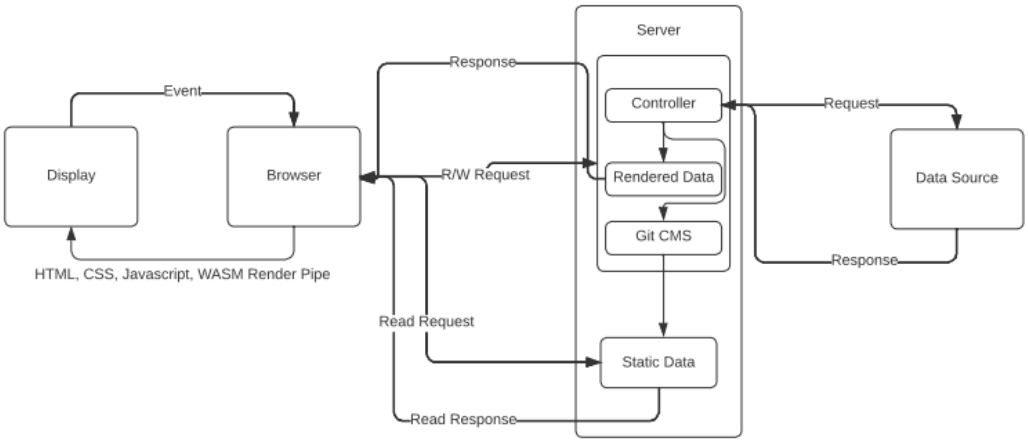

Figure 5: Proposed Web-Architecture

This diagram shows the changes made by our proposed architecture. Nothing is removed from it and an extra component was added, the git based CMS. As it can be seen, the **controller** can change contents of the **static** files using git based CMS in this new design. Allowing removal of several unnecessary render calls by http requests.

In figure 6 we show the sequence diagram [7] in case of our proposed architecture. We start with the "Browser," representing the user's web browser, which sends a read or write request to the "Server." The "Server" acts as the middleman and communicates with the "DataSource" to ask for the necessary data.

The "DataSource" is responsible for creating or retrieving the requested data. It performs the required data operations, such as fetching information from a database or generating dynamic content.

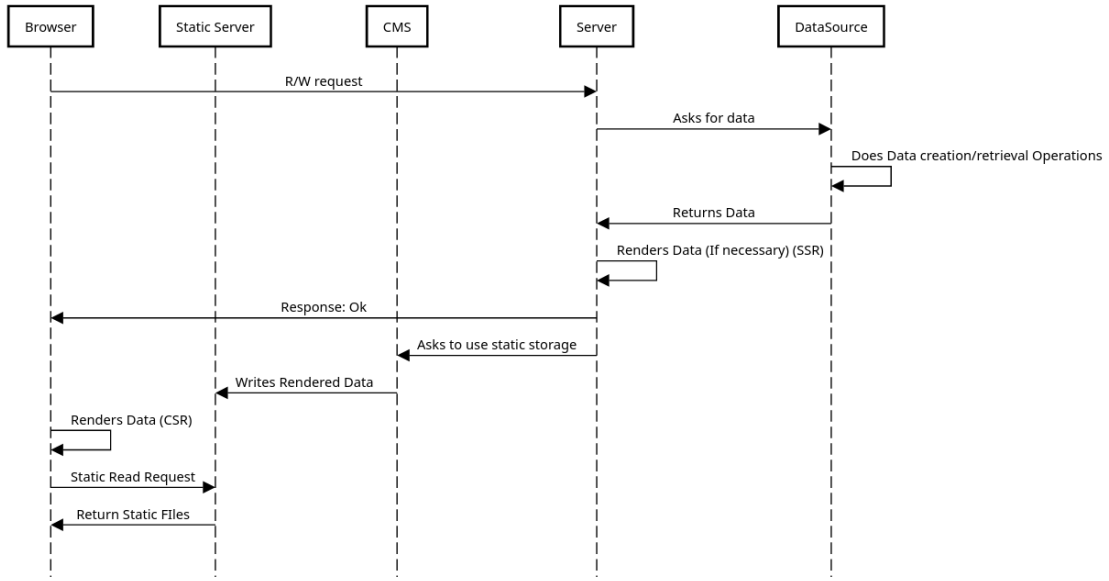

Figure 6: Web-Architecture: Sequence Diagram - Proposed Architecture

Once the "DataSource" has processed the data, it sends it back to the "Server." The "Server" may also perform additional operations, such as rendering the data if needed. This process is known as Server-Side Rendering (SSR), where the server prepares the data in a format suitable for the browser to display, same as the previous diagram.

After processing and rendering, the "Server" sends the response back to the "Browser," indicating that the request was successful. The "Browser" then takes over and renders the received data on the user's screen. This is called Client-Side Rendering (CSR), where the browser handles the rendering of the data without relying on the server.

Now, in this particular scenario, there is also a "CMS" (Content Management System) involved. The "Server" communicates with the CMS to request the use of static storage. This typically happens when the rendered data needs to be stored as a static file for later use or to be served directly by a static server.

The "Server" sends a request to the CMS, asking it to store the rendered data. The CMS, in turn, interacts with the "Static Server," which is responsible for serving

static files. The CMS writes the rendered data to the "Static Server," where it will be stored and made available for future requests.

Hence, when the "Browser" sends requests to the "Static Server" directly for static files necessary static file created by the CMS is also served, making SSR and data source call unnecessary and unused.

Architecture Regression:

In comparing the two sequence diagrams provided, it becomes evident that the presence or absence of a Content Management System (CMS) has a significant impact on the web architecture operations. Let's explore what has changed based on the differences observed.

In the first sequence diagram at figure 4, which represents web architecture operations without a CMS, the focus is primarily on the communication between the Browser, Server, and DataSource. The Server acts as an intermediary between the Browser and the DataSource, handling requests and coordinating data retrieval or creation. The DataSource is responsible for data operations, such as fetching information or generating dynamic content. The Server may also perform rendering operations (SSR) before sending the response back to the Browser, which then handles the rendering of data on the client-side (CSR). Additionally, the Browser can directly request static files from the Static Server when needed.

However, in the second sequence diagram at figure 4, which includes a CMS in the web architecture, notable changes occur. The Server's interaction with the CMS introduces an additional layer of functionality. When the Server requires static storage, it communicates with the CMS to request the usage of such storage. This is particularly useful when the rendered data needs to be stored as a static file for future use or when it should be served directly by a Static Server. The CMS acts as a bridge between the Server and the Static Server, facilitating the writing of rendered data to the Static Server for storage.

Therefore, the presence of a CMS brings advantages such as enhanced management of static files, improved content storage and retrieval, and the ability to serve rendered data directly from a Static Server. It provides a centralized system to handle content-related tasks, streamlining the web architecture operations and offering more flexibility in terms of content delivery.

Overall, the integration of a CMS into the web architecture introduces an additional layer of functionality and enhances the management of content and static files, leading to more efficient and versatile web operations.

3.2 Experimentation Criteria

3.2.1 Performance Testing

The usage of pre-rendered static websites should decrease the response time for reading. However re-rendering a website in case of writing may decrease the response time. We used Apache Bench [28] to calculate this metrics for both of these website. The comparison is done for both read heavy and write heavy use case.

3.2.2 Scalability

Scalability testing measures the user limit is the maximum number of concurrent users that the system can support while remaining stable and providing reasonable response time to users [5].

This is measured with Apache Bench as well. We set a time limit and try to find out how many concurrent request can the system handle within the time limit. This way we find out the maximum number of concurrent system user.

Furthermore we can test the amount of data necessary for better content delivery. Git, being using patch delivery system, the traffic of data should be much lower.

3.2.3 Information Validation

Git implements a patch and log based system committed overtime. This allows moving between different state of the server. Git was built to do it. However getting the same result with database based system requires a very complex system of data store and retrieval. We performed a complexity analysis to prove the usefulness of git based CMS for information validation.

3.3 Implementations

The entire experiment for read-heavy write-heavy showcase was written in JavaScript for Node.js[12] runtime. Node.js is a powerful, open-source JavaScript runtime built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript code on the server-side, enabling the development of highly scalable and efficient web applications. Node.js utilizes an event-driven, non-blocking I/O model, making it lightweight and capable of handling concurrent requests without blocking the execution. It has a rich ecosystem of modules and libraries available through its package manager, npm, which facilitates rapid development and easy integration of functionalities into Node.js applications. Node.js is widely recognized for its excellent performance and high scalability, making it ideal for building real-time applications, APIs, microservices, and even full-stack web applications. With Node.js, developers can leverage their existing JavaScript skills to build fast, scalable, and efficient server-side applications, promoting code reuse and improving overall productivity. Node.js has gained immense popularity and community support, making it one of the go-to choices for server-side development in the JavaScript ecosystem.

3.3.1 EJS-CMS

EJS-CMS is the model git-based CMS that we implemented to conduct all the necessary experiments.

This CMS uses the templating engine to store the provided data in HTML. Then we run git operations to commit the changes.

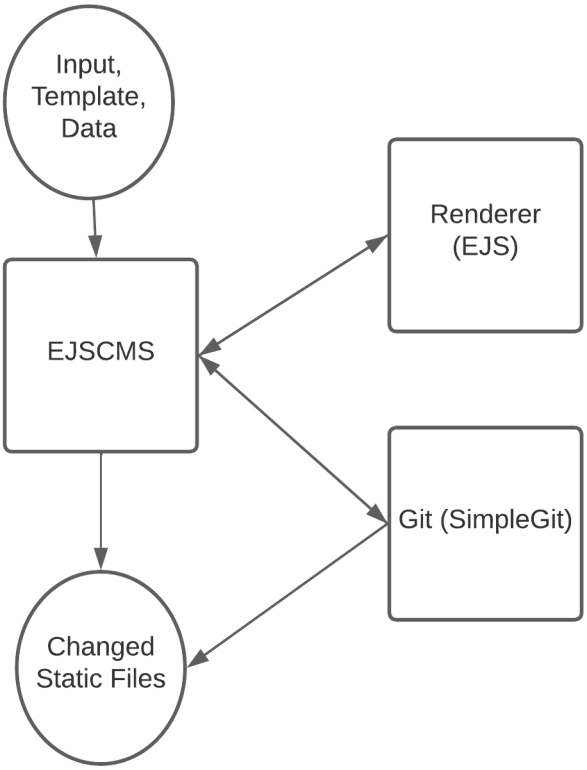

Figure 7: EJSCMS Implementation

EJS-CMS is a static Javascript library that includes two methods:

- **configure:** This takes configuration inputs. For example where the static website is going to be built.
- **commit:** It takes three parameters, the template, the data and the route. Then it performs the necessary cms operations to modify the static files in the file system.

EJSCMS is available here: https://github.com/ptpt-community/ejscms.git
EJSCMS is built with the following npmjs[10] libraries:

- **EJS** : EJS [24], or Embedded JavaScript, is a popular templating engine for Node.js and web applications. It allows developers to generate dynamic HTML content by embedding JavaScript code within HTML templates. EJS provides a simple and intuitive syntax that closely resembles regular HTML

markup, making it easy to learn and use. With EJS, developers can seamlessly integrate data from their server-side applications into the HTML templates, enabling dynamic content rendering. This templating engine supports a wide range of features, including conditional statements, loops, partials, and custom functions, which facilitate code reuse and maintainability. EJS also provides support for passing variables, rendering views, and handling error scenarios. It promotes a clear separation of concerns by keeping the logic separate from the presentation layer. EJS is highly flexible and can be integrated with various web frameworks, such as Express.js, to create robust and scalable web applications. Its versatility and simplicity make it a popular choice for developers seeking a powerful templating solution for their Node.js projects.

- **SimpleGit:** SimpleGit[14] is an npm package that simplifies the process of working with Git repositories within Node.js applications. It provides a straightforward and intuitive interface, allowing developers to seamlessly interact with Git through their Node.js code. SimpleGit abstracts away the complexities of executing Git commands and provides a convenient API for common version control operations. It enables developers to initialize repositories, perform commits, manage branches, clone repositories, handle remote operations like push and pull, and much more. With SimpleGit, developers can automate Git workflows, integrate version control functionality into their applications, and efficiently collaborate on projects. This npm package empowers Node.js developers to harness the power of Git without needing to rely on command-line tools or external processes, making version control an integral part of their development workflow.

Properties of Model Git Based CMS:

The model GitBased CMS, namely EJSCMS is an npm static library. It is a headless CMS, IE it does not have any frontend. The EJSCMS differs from NetlifyCMS, CrafterCMS at the aspect that it changes the viewables directly. At the same time, other CMSes provide a decoupled frontend, which is not implemented in case of

EJSCMS because it is not necessary for the architecture. EJSCMS also does not need a database backend, unlike Wordpress.

4 Experimentation

4.1 Theoritical Usecase Experiment

The experiments were conducted in a Linux environment with Kernel Version 6.2 and CPU architecture is x86_64. We have created a DBMS-Based, Server Side Rendering (SSR) web server and a Volatile array-based SSR Web Server experiment.

4.1.1 DBMS-Based, Server Side Rendering (SSR) Web Server

A web server that fetches data from the database and renders it using EJS. Finally, the data is sent to the client side.

We have used Express.js[13] for handling server side http gateway. Express.js is a popular web application framework for Node.js that simplifies the process of building robust and scalable web applications. It provides a minimalist yet powerful set of features that enable developers to create efficient server-side applications and APIs. With Express.js, developers can easily handle routing, middleware, and request/response handling, allowing them to focus on the core functionality of their applications. Express.js offers a flexible and modular architecture, allowing developers to choose from a vast ecosystem of middleware and extensions to enhance their applications. It also provides support for template engines, enabling dynamic rendering of HTML pages. Express.js promotes a clean and concise coding style, making it easy to understand and maintain codebases. It has gained widespread adoption due to its simplicity, performance, and extensive community support. Whether you're building a small API or a large-scale web application, Express.js empowers developers to create robust and scalable server-side solutions with ease.

For making database handling simpler, we used Sequelize[15]. Sequelize is a

powerful Object-Relational Mapping (ORM) [1] library for Node.js that simplifies the interaction between JavaScript code and relational databases. It provides a straightforward way to map database tables to JavaScript objects and vice versa, allowing developers to work with databases using familiar object-oriented paradigms. With Sequelize, developers can define models that represent database tables and define relationships between them. The library supports various database systems such as MySQL, PostgreSQL, SQLite, and MSSQL, offering compatibility and flexibility across different environments. Sequelize provides a rich set of features, including support for data validation, query generation, transaction management, and migration handling. It also offers robust support for advanced concepts like eager loading, associations, and hooks, allowing developers to efficiently work with complex data models. Sequelize abstracts away the complexities of writing raw SQL queries, providing a higher level of abstraction and productivity. It has gained popularity for its ease of use, extensive documentation, and active community support, making it a preferred choice for database interactions in Node.js applications.

The DBMS-Based, Server Side Rendering (SSR) Web Server is available hre: https://github.com/ptpt-community/dbms-based-webshowcase.git

4.1.2 Volatile array-based SSR Web Server

A web server that has preloaded data into its memory. All other functionality is like the SSR Database website. The main purpose of this implementation is to become "data source" agnostic. As volatile memory usage is the fastest way to use data[17]. For this implementation, we have used Express.js as well.

The Volatile array-based SSR Web Server is available here: https://github.com/ptpt-community/volatile-memory-showcase.git

4.2 Experiments

4.2.1 Performance Testing

To assess the response time of our system, we conducted experiments focusing on two distinct use cases: read-heavy and write-heavy scenarios. We ensured that these experiments were performed across all implementations, whenever applicable, to obtain comprehensive results. To carry out the experiments, we utilized Apache Bench, a reliable tool for benchmarking and load testing.In the read-heavy use case, our objective was to measure the system's performance when dealing with a significant volume of read operations. By simulating multiple concurrent read requests, we evaluated how efficiently the system could handle and respond to such demands. This allowed us to gauge its responsiveness and identify any potential bottlenecks or areas for optimization.Conversely, in the write-heavy use case, our focus shifted to examining the system's performance under a substantial load of write operations. We aimed to measure the system's ability to handle frequent data updates or additions and assess its stability and scalability in such scenarios.

By subjecting the system to a high volume of write requests, we could evaluate its efficiency and responsiveness in processing and persisting the changes.By conducting these experiments across all implementations, we obtained valuable insights into the performance characteristics and capabilities of our system. These findings helped us identify areas for improvement, fine-tune our implementation strategies, and optimize the system's overall performance and responsiveness.It is worth noting that the experiments were performed using Apache Bench [28], a widely recognized and reliable tool for benchmarking. This ensured the accuracy and validity of our measurements, enabling us to make informed decisions based on the obtained results.

Read Heaviness Testing:

The following workflow was used for this test. First we propagated the database with the data. We collected the read heavy route for testing and used Apache Bench to perform the operation. Finally we collected the data for analysis.

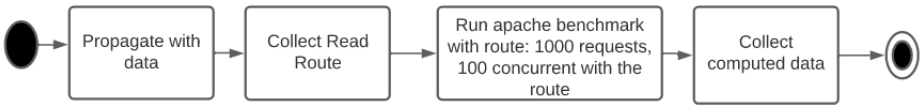

Figure 8: Workflow of Read Heaviness Testing

In this experiment, 1000 requests were done in total where 100 requests were concurrent.

	SSR with Database	SSR with Volatile Memory	Proposed Architecture
Time taken for test	57.136s	7.34s	.865s
Request per second	17.5	132	1734.04
Failed Requests	0	0	0

Table 2: Performance Testing (Read Heavy)

The data shown at 5 reveals that our proposed CMS demonstrates superior performance across all metrics. It significantly reduces the time taken for the test, achieving a mere 0.865 seconds compared to 57.136 seconds for SSR with Database and 7.34 seconds for SSR with Volatile Memory. Furthermore, the proposed CMS handles a remarkably high number of requests per second, reaching 1734.04, while SSR with Database and SSR with Volatile Memory achieve 17.5 and 132 requests per second, respectively. Notably, all three approaches exhibit zero failed requests, implying a reliable performance overall. We can see that response time has improved a lot for our proposed model for read-heavy test cases. Even if traditional SSR does not fetch data from external data sources, the result is significantly different from our proposed architecture.

Write Heaviness Testing:

The following workflow was used for this test. First we prepared the required data to be sent. Then we collected the write route for testing and used Apache Bench to perform the operation. Finally we collected the data for analysis.

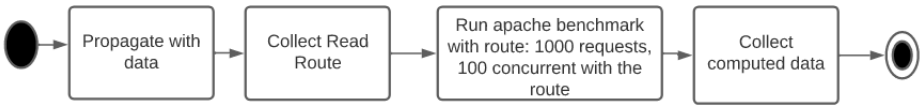

Figure 9: Workflow of Write Heaviness Testing

The post data was a JSON with the following schema:

```
{
  "type": "object",
  "properties": {
    "content": {
      "type": "string"
    }
  },
  "required": [
    "content"
  ]
}
```

	SSR with Database	SSR with Volatile Memory	Proposed Architecture
Time taken for test	2.56	Not Applicable	40.446s
Request per second	24		390
Failed Requests	0		0

Table 3: Performance Testing (Write Heavy)

From the data shown at 3, the write heavy case, however we can see that response time has decreased a lot for our proposed model. Proposed architecture took a very long 40 s to finish the test while the database call took only 2.5 seconds. This strongly suggests that, this architecture is not for write heavy use cases. Data from SSR with volatile memory is not applicable here as there is no write operation that can be stored in volatile memory.

4.2.2 Scalability

We have restricted the time to 5 seconds and 2 seconds to conduct two tests to determine the scalability, based on Oracle's load testing documentation [5].

Requests completed in	SSR with Database	SSR with Volatile Memory	Proposed Architecture
5 seconds	89	623	1000*
2 Seconds	33	234	1000*

Table 4: Sent Request in Fixed Time

[1]

In the given table, the performance of three systems, namely SSR with Database, SSR with Volatile Memory, and the Proposed Architecture, is compared based on the number of requests completed within fixed time intervals. The proposed architecture demonstrates superior scalability, with over 1000 requests completed within 5 seconds and over 1000 requests completed within 2 seconds, surpassing the performance of the other systems. It is to be noted that Apache Bench limits parallel request to 1000.

4.3 Information Validation

Speculation on 'git log' and 'git patches' gives in the cms:

- When a data was changed.
- Who changed the data.
- What was changed.
- The tree [4] data structure of how everything was changed.

To demonstrate, here is a patch created from the git used within ejscms:

```
From 33301c116d250332d49c20d3b78043c6310d278c Mon Sep 17 00:00:00 2001
From: Username <username@mail.edu>
Date: Fri, 19 May 2023 21:34:11 +0600
Subject: [PATCH] Sucessfully committed

---
 home/index.html | 2 ++
 1 file changed, 2 insertions(+)

diff --git a/home/index.html b/home/index.html
index c428861..de4f559 100644
--- a/home/index.html
+++ b/home/index.html
@@ -7915,6 +7915,8 @@

   <li>Test Patch 2</li>

+ <li>Test Patch 3</li>
+
   </ul>

   </body>
--
2.34.1
```

In this experiment we simply observed whether a git patch has all the necessary information to determine where the changes are, when it was done and who has performed the change.

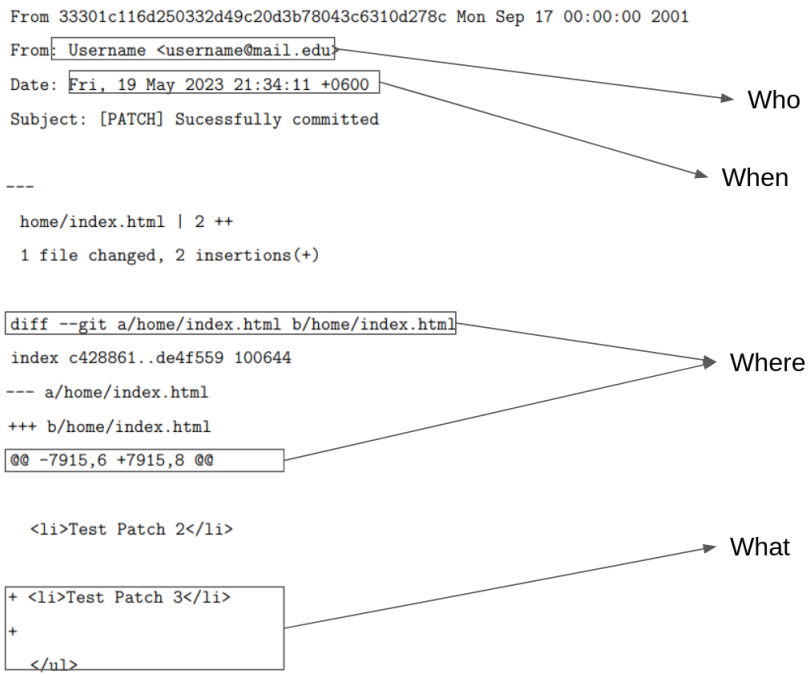

Figure 10: Git Patch Deomnstration

Hence it is established that information validation is a benefit introduced from this architecture.

4.4 Practical Usecase Experiment

For this experiment, we have used MediaWiki[6] to demonstrate the benefits of this experiment. MediaWiki is a widely-used open-source wiki software platform that facilitates collaborative content creation and editing in a structured manner. Originally developed for Wikipedia, it serves as the backbone for numerous other wiki-based websites and knowledge management systems. With its robust feature set, MediaWiki empowers users to organize, manage, and present information effectively. Its intuitive syntax allows users to format and structure content easily using headings, lists, tables, links, and multimedia elements. MediaWiki supports collaborative editing, revision tracking, and user authentication, enabling seamless contributions from multiple users. It also offers powerful search capabilities for efficient information retrieval. With its modular architecture and customization options, MediaWiki can be tailored to specific requirements and integrated with

various extensions and plugins. MediaWiki has made significant contributions to online communities and information sharing by providing a versatile and powerful platform for creating and managing collaborative knowledge repositories.

We have used a docker image of Mediawiki, available at *MediaWiki - Docker Hub* to create a Wiki for the work. The docker image was hosted in Linux fedora 5.17.5-300.fc36.x86_64 #1 SMP PREEMPT x86_64 x86_64 x86_64 GNU/Linux. Then we created a EJSCMS hosting server, that uses rendered data from MediaWiki to perform its read operations.

To implement, we created a Node based proxy server that sits in between the data source (In this case WikiMedia) and the browser. The server is also home to the ejscms that we are using. The diagram 11 shows how the architecture is implemented here.

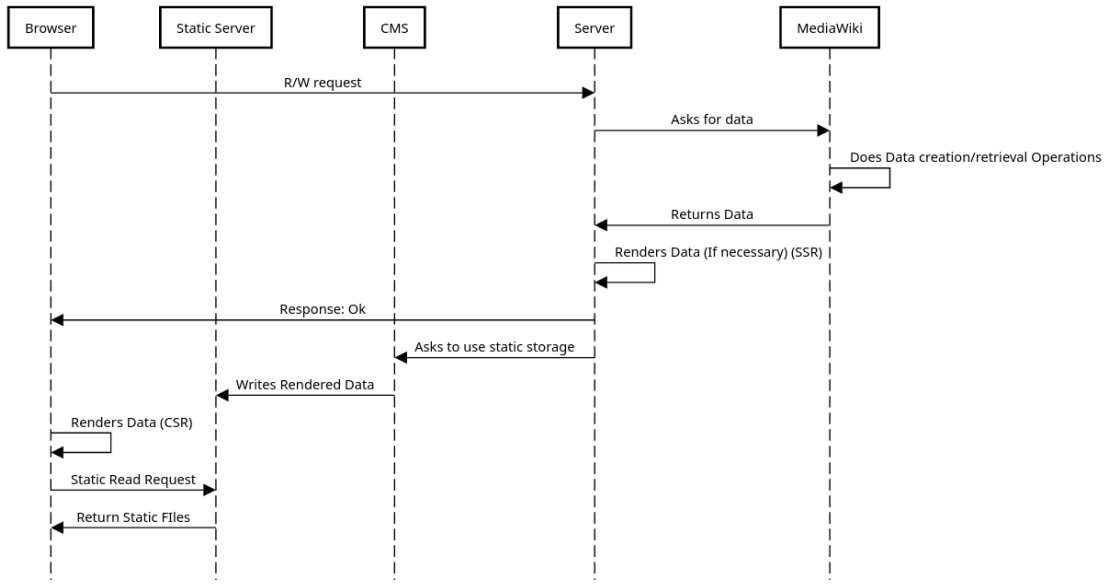

Figure 11: MediaWiki: Sequence Diagram - Proposed Architecture

4.4.1 Performance Testing

The MediaWiki's updated architecture's CMS Proxy server is available at: https://github.com/ptpt-community/mediawiki-cms-testing.git

In this experiment, 1000 requests were done in total where 100 requests were concurrent, for MediaWiki, and our changed version of MediaWiki for a certain article that needs sclability. This is a read heavy use case of the web application.

	MediaWiki	MediaWiki with proposed Architecture
Time taken for test	8.753s	0.422s
Request per second	114.25	2369.20 (mean)
Failed Requests	798	0
Successful Requests	1000	1000

Table 5: Performance Testing: MediaWiki (Read Heavy)

4.4.2 Scalability

We have restricted the time to 5 seconds and 2 seconds to conduct two tests to determine the scalability, based on Oracle's load testing documentation for both version of MediaWiki.

Requests completed in	MediaWiki	MediaWiki with Proposed Architecture
5 seconds	524	1000*
2 Seconds	217	1000*

Table 6: Sent Request in Fixed Time

[2]

The table 6 provides a comparison between the performance of MediaWiki and MediaWiki with the proposed architecture in terms of request completion within fixed time intervals. It highlights the potential improvement achieved with the proposed architecture, as it allowed the system to reach the maximum limit of parallel requests specified by Apache Bench, denoting the proposed architecture provides superior scalability.

4.5 Research Insights from the Experiments

With all those experiments perforemd we have established the follwing:

- The proposed front end architecture is better for read heavy websites.
- The architecture provides better scalability and information validation.
- The new architecture decreases performance for write heavy section. Hence it is better implemented with traditional architecture.

5 Future Directions

The work described above lays the foundation for numerous potential research areas and opens up exciting avenues for further improvements. One possible direction for future research involves propagating the problem to the kernel level, leveraging log-based file systems like nilfs2[29] to enhance performance. By integrating such file systems, the efficiency and responsiveness of the overall system can be significantly improved.

Another promising area for exploration is enhancing the security of information validation in Git. Research efforts can focus on developing changes and additions to Git that support more secure information validation mechanisms. For instance, embedding an encryptor directly into Git can provide an added layer of protection for sensitive data, ensuring its integrity and confidentiality throughout the version control process.

Given that Git is a distributed revision control system, exploring the potential of blockchain technology derived from it holds great promise. By incorporating blockchain concepts and principles into Git, information validation can be further enhanced. The decentralized and tamper-resistant nature of blockchain can provide robust validation mechanisms, reinforcing the integrity and reliability of the version control system.

Additionally, there is room for improvement in the performance of a Git-based Content Management System (CMS), particularly in scenarios with heavy write loads. Future research can focus on optimizing the write-heavy performance of

such systems by investigating techniques such as data caching, parallel processing, or distributed storage solutions. Enhancing the CMS's ability to handle high volumes of write operations will contribute to improved overall efficiency and user experience.

Moreover, this research can be conducted for multiple other web application solutions, while taking account of storage usage as well.

In conclusion, the work discussed here sets the stage for a multitude of exciting research directions. Exploring kernel-level integration, enhancing security measures, leveraging block chain technology, and improving write-heavy performance in Git-based systems are all promising avenues for future investigations. Continued research and innovation in these areas will undoubtedly lead to further advancements and improvements in version control systems and content management.

6 Conclusion

In this work, we have demonstrated the advantages of integrating a git-based CMS to link server controller logic with static files, resulting in various benefits for web applications. Our analysis began by examining the strengths and weaknesses of the current front-end architecture, followed by an investigation into existing CMS implementations to address the response time loss associated with Server Side Rendering. Through this research, we developed a novel architecture that incorporates a git-based CMS.

To evaluate the effectiveness of our proposed solution, we conducted experiments targeting three common web development issues. For this purpose, we created a git-based CMS and corresponding models, with one utilizing the proposed architecture and the other two not employing it. Among the latter, one model relied on a MySQL database for data storage, while the other utilized volatile memory to eliminate biases from API dependencies. Our experimental results consistently demonstrated that the proposed solution outperformed the other two approaches,

highlighting that frequent Server Side Rendering and data retrieval are the main contributors to reduced response times, which can be mitigated by adopting a git-based CMS architecture while providing additional services. Nonetheless, it should be noted that this improved performance comes at the cost of sacrificing write-heavy performance.

Furthermore, we validated our claims by implementing a practical showcase using MediaWiki, a widely used Wiki site creator employed by organizations such as Wikipedia. Our showcase demonstrated a significant enhancement in performance and scalability achievable with the proposed architecture.

This research paves the way for further exploration in various research areas, including git and blockchain. Moreover, there is room for improvement by discovering methods to enhance write-heavy performance.

References

[1] D. Barry and T. Stanienda, "Solving the java object storage problem," *Computer*, vol. 31, no. 11, pp. 33–40, Nov. 1998, Excerpt at `https://www.service-architecture.com/articles/object-relational-mapping/transparent-persistence-vs-jdbc-call-level-interface.html`. [Online]. Available: `https://www.computer.org/csdl/magazine/co/1998/11/ry033/13rRUxC0SRY`, Lines of code using O/R are only a fraction of those needed for a call-level interface (1:4). For this exercise, 496 lines of code were needed using the ODMG Java Binding compared to 1,923 lines of code using JDBC.

[2] E. Meijer, D. Leijen, and J. Hook, "Client-side web scripting with haskellscript," in *Practical Aspects of Declarative Languages*, G. Gupta, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 196–210, ISBN: 978-3-540-49201-6.

[3] I. Wijegunaratne and G. Fernandez, *Distributed Applications Engineering: Building New Applications and Managing Legacy Applications with Distributed Technologies* (Practitioner Series). Springer London, 1998, ISBN: 9783540762102. [Online]. Available: `https://books.google.com.bd/books?id=ZtJQAAAAMAAJ`.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction To Algorithms* (Mit Electrical Engineering and Computer Science). MIT Press, 2001, ISBN: 9780262032933. [Online]. Available: `https://books.google.com.bd/books?id=NLngYyWFl%5C_YC`.

[5] Oracle Corporation. "Oracle Load Testing User's Guide." (2001), [Online]. Available: `https://docs.oracle.com/cd/E25292_01/doc.901/e15484/oltchap2.htm` (visited on 05/18/2023).

[6] MediaWiki Contributors. "MediaWiki." (2002), [Online]. Available: `https://www.mediawiki.org/`.

[7] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (Addison-Wesley object technology series). Addison-Wesley, 2004, ISBN: 9780134865133. [Online]. Available: https://books.google.com.bd/books?id=19yjzQEACAAJ.

[8] D. Goans, G. Leach, and T. M. Vogel, "Beyond html: Developing and re-imagining library web guides in a content management system," *Library Hi Tech*, 2006.

[9] M. Seadle, "Content management systems," *Library Hi Tech*, vol. 24, no. 1, B. L. Eden, Ed., pp. 5–7, Jan. 2006. DOI: 10.1108/07378830610652068. [Online]. Available: https://doi.org/10.1108/07378830610652068.

[10] npm, Inc. "npm." (2009), [Online]. Available: https://www.npmjs.com/.

[11] B. O'Sullivan, "Making sense of revision-control systems," *Communications of the ACM*, vol. 52, no. 9, pp. 56–62, 2009.

[12] The Node.js Contributors. "Node.js." (2009), [Online]. Available: https://nodejs.org/.

[13] The Express.js Contributors. "Express.js." (2010), [Online]. Available: https://expressjs.com/.

[14] David Elbe. "simple-git." (2011), [Online]. Available: https://www.npmjs.com/package/simple-git.

[15] Sequelize Contributors. "Sequelize." (2011), [Online]. Available: https://sequelize.org/.

[16] S. Chacon and B. Straub, *Pro Git*, 2nd. USA: Apress, 2014, ISBN: 1484200772.

[17] S. Mishra, N. K. Singh, and V. Rousseau, "Chapter 3 - generic soc architecture components," in *System on Chip Interfaces for Low Power Design*, S. Mishra, N. K. Singh, and V. Rousseau, Eds., Morgan Kaufmann, 2016, pp. 29–51, ISBN: 978-0-12-801630-5. DOI: https://doi.org/10.1016/B978-0-12-801630-5.00003-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128016305000037.

[18] A. A. Althanoon, "Mobile web browsers in android deriving reference architecture," *International Journal of Computer Applications*, vol. 180, pp. 975–8887, Jan. 2018.

[19] A. Mukhamadiev, "Transitioning from server-side to client-side rendering of the web-based user interface: A performance perspective," 2018.

[20] M. Tomiša, M. Milković, and M. Čačić, "Performance evaluation of dynamic and static wordpress-based websites," in *2019 23rd International Computer Science and Engineering Conference (ICSEC)*, 2019, pp. 321–324. DOI: 10.1109/ICSEC47112.2019.8974709.

[21] T. F. Iskandar, M. Lubis, T. F. Kusumasari, and A. R. Lubis, "Comparison between client-side and server-side rendering in the web development," *IOP Conference Series: Materials Science and Engineering*, vol. 801, no. 1, p. 012 136, May 2020. DOI: 10.1088/1757-899X/801/1/012136. [Online]. Available: https://dx.doi.org/10.1088/1757-899X/801/1/012136.

[22] S. K. Shivakumar and S. K. Shivakumar, "Modern web performance patterns," *Modern Web Performance Optimization: Methods, Tools, and Patterns to Speed Up Digital Platforms*, pp. 273–300, 2020.

[23] Crafter CMS. "Crafter cms documentation," Crafter CMS. (2023), [Online]. Available: https://docs.craftercms.org/en/4.0/index.html (visited on 05/22/2023).

[24] EJS. "EJS: Embedded JavaScript Templates." (2023), [Online]. Available: https://ejs.co/ (visited on 05/18/2023).

[25] IBM. "Ibm developer for z/os documentation - json4j," IBM. (2023), [Online]. Available: https://www.ibm.com/docs/en/developer-for-zos/14.1?topic=json-javascript-object-notation-json4j (visited on 05/19/2023).

[26] Kinsta. "WordPress Market Share." (2023), [Online]. Available: https://kinsta.com/wordpress-market-share/ (visited on 05/22/2023).

[27] Netlify CMS. "Netlify cms documentation," Netlify. (2023), [Online]. Available: https://preview-auth-doc--netlify-cms-www.netlify.app/docs/ (visited on 05/22/2023).

[28] The Apache Software Foundation. "Ab - apache http server benchmarking tool," Apache Software Foundation. (2023), [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html (visited on 05/19/2023).

[29] The Linux Kernel Archives. "NILFS2: The New Implementation of a Log-structured File System," The Linux Kernel Archives. (May 2023), [Online]. Available: https://www.kernel.org/doc/html/next/filesystems/nilfs2.html.

[30] "WordPress Documentation," WordPress. (), [Online]. Available: https://wordpress.org/documentation/ (visited on 07/22/2022).

[31] Docker. "MediaWiki - Docker Hub." (N/A), [Online]. Available: https://hub.docker.com/_/mediawiki.

[32] Portent. "Research: Site speed is hurting everyone's revenue." Accessed on May 23, 2023, Portent. (N/A), [Online]. Available: https://www.portent.com/blog/analytics/research-site-speed-hurting-everyones-revenue.htm (visited on 05/23/2023).