Vol.12-1 (2025) 103–119
International Journal of Computers and Information (IJCI)
Available online at https://ijci.journals.ekb.eg/

Optimizing Web Application Development: A Proposed Architecture Integrating Headless CMS and Serverless Computing

Ahmed M. Eassa

Management Information Systems Department, Faculty of Management, MTI University, Cairo, Egypt

Abstract

Headless Content Management Systems (CMS) provide great flexibility by decoupling the content repository or back-end from the presentation layer or front-end. On the other hand, serverless computing architectures bring key advantages, including enhanced performance, automatic scalability, high availability, cost efficiency, and optimized management of execution time and resources. This paper proposes architecture that integrates the serverless computing infrastructure and headless CMS functions to optimize the web application development process by allowing developers to decouple the content management backend from the presentation layer while leveraging the event-driven nature of serverless functions. The proposed architecture aims to enhance scalability, reduce operational complexity, increase availability, and improve performance during the web application development process. The proposed architecture has been evaluated through extensive testing, demonstrating significant improvements in response time, error rates, and resource utilization under heavy loads. The results demonstrate a reduction in response time to an average of 88–782 milliseconds, an error rate under 2%, and efficient CPU and RAM utilization that did not exceed 56% and 39%, respectively. According to testing results, the proposed architecture is suited for modern digital experiences, providing a robust and efficient framework for web application development. Furthermore, proposed architecture flexibility ensures adaptability to evolving technologies, making it future-proof for the rapidly changing landscape of digital platforms. This paper contributes to the web applications field by presenting a comprehensive approach to leveraging headless CMS and serverless computing for more efficient and scalable web development.

Keywords: Headless CMS; Serverless Computing; Web Application; Optimization Development.

1. Introduction

A headless CMS represents a significant change in how content is managed and published online. Unlike traditional CMS, which tightly couple the content management back-end with the front-end [1]. In a headless CMS architecture, content is created and stored in a back-end platform as a main repository for different types of front-end [2]. However, the content is delivered from the main repository via APIs (Application Programming Interfaces). The decoupling architecture of headless CMS allows greater flexibility in how content is presented across various channels and devices, such as web applications, smart devices, IoT devices, or integrating with emerging technologies like Siri, Alexa, and Google Assistant [3]. The term "headless" refers to the absence of a traditional front-end or end-user interface. In addition, developers can gain the advantage of complete freedom to build different user interfaces that are compatible with different display devices using their preferred frameworks, such as React, Angular, and Vue [4]. This approach empowers developers to create highly tailored user experiences without being constrained by the limitations of a monolithic CMS front-end.

Headless CMS architecture offer several advantages [2, 5, 6], including:

- Scalability: it can easily scale to fit with growing traffic and evolving additional needs by handling content delivery using APIs.
- Flexibility: web application Developers have full freedom power to generate different front-end designs on different channels and devices.
- Future-proofing: By separating back-end from front-end, In the future, it will become easier to modify each separate part easily and adapt to new technologies without modifying and rebuilding other parts.
- Omnichannel Content Delivery: Where the ability to present content in different forms and on different channels, and thus the user's ability to use the appropriate and easiest channel for him. Therefore, it will be a competitive advantage for any business.

On the other hand, serverless computing represents one of the important transformations for cloud-based application development. On the contrary, in traditional systems, developers are responsible for managing the servers themselves, in addition to developing the applications that they work on [7,8]. It revolves around the concept of "functions as a service" (FaaS) [9], A cloud service provider (CSP) automatically handles tasks such as scaling, maintenance, and server management. Therefore, the developers focus on delivering value to end-users and writing code without having to worry about the underlying infrastructure. Developers can write many software functions code and then execute them in ephemeral containers managed by CSP scaling automatically based on demand [10]. Key characteristics of serverless computing include [11-14]:

- Auto-scaling: CSP manages all resources, such as processing and storage are added automatically when developer applications need it, and upon completion of their implementation, resources are reduced automatically as well. Thus, ensuring the optimal exploitation of resources.
- Event-driven: The operation of serverless computing functions depends on events such as HTTP requests, file uploads, database triggers, message queues, or scheduled tasks. This event-driven model enables highly responsive and scalable applications.
- Pay Per Use: Payment is made only for the resources that are used and according to the actual implementation time, thus reducing costs for developers.
- No server management: Where developers do not focus on adjusting the infrastructure of running environment, such as setting server OS or setting any resource required to provide the appropriate work environment.
- Ephemeral execution: Serverless functions run in something similar to ephemeral containers that are allocated for quick execution of functions, and they are destroyed after execution. This architecture promotes scalability and isolation.

Serverless computing is well-suited for a wide range of use cases, such as CMS web applications. Overall, serverless computing offers web developers a powerful and efficient way to build and deploy CMS web applications, enabling greater performance, auto-scalability, high availability, cost-effectiveness, and good management of execution time and resources [15]. Also, the flexibility and scalability offered by headless CMS solutions make them an attractive option for business seeking to deliver dynamic, personalized content experiences.

The integration of Headless Content Management Systems (CMS) with serverless computing architectures presents a unique and valuable contribution to web application development, addressing several persistent challenges in flexibility, scalability, and infrastructure management. Unlike traditional web development frameworks, this hybrid architecture empowers developers to decouple content management from the presentation layer while leveraging serverless functions to optimize performance. This dual approach fills a notable gap in current literature, where solutions often focus on either flexible content management or scalable computing without fully integrating both. Therefore, this study introduces a comprehensive solution that allows for seamless adaptation to fluctuating user demands while reducing operational complexity. By advancing this approach, this paper aims to support developers and organizations in creating efficient, scalable digital experiences with minimal backend maintenance.

In this paper, an architecture is proposed for optimizing headless CMS based on serverless computing. The proposed architecture aims to leverage the benefits of serverless computing while preserving the flexibility and extensibility of headless CMS platforms. The organization of this paper is as follows: related work is discussed in Section 2. The proposed architecture and its implementation details are introduced in Sections 3 and 4, respectively. The results of the evaluation are presented in Section 5. Finally, the paper concludes in Section 6.

2. Related Work

Many studies have addressed the integration of serverless computing with many domains, such as web applications, Internet of Things (IoT), image recognition, predictive analytics, and microservices [16-18]. For the purpose of this paper, the focus will be on three levels of depth.

In the first level, some of them have contributed to the area of integrating serverless with web applications. Olli Paakkunainen [19] demonstrates the advantages of serverless computing for web applications by using the FaaS platform to implement a web application backend. Moreover, it identified the FaaS as a cost-effective solution for low traffic web applications, and it also gave better performance than the PaaS-based backend solution in smaller projects or utilized in sub-components of a large project. Cosmina Ivan et al. [20] examine the performance for building and deploying Web APIs by using two deployment environments which are virtual machines (VMs) and function-as-a-service (FaaS). The search results found no significant response time differences between deployments when VMs are configured for the expected load, and test scenarios are within the FaaS hardware limitations.

Fatima Samea et al. [21] presents a novel approach to developing data-driven architecture called UMLPDA (Unified Modeling Language Profile for Data-driven Applications) based UML to model the front-end and the back-end requirements for data-driven applications by abstracting away low-level infrastructure concerns and automating common tasks. Diego Zanon [22] explores the principles, practices, and tools necessary for building scalable, cost-effective, and resilient web applications without managing server infrastructure. It covers various aspects of serverless development, including designing the building blocks of Amazon Web Services (AWS), deploying web applications using AWS S3 with HTTPS using CloudFront, building front-end web applications using React as an SPA, and developing the Node. js backend to handle requests and connect to a SimpleDB database. Markus Ast and Martin Gaedke [23] proposes an innovative approach to developing self-contained web components using serverless computing techniques. It explores how Serverless Computing and Web Components could be combined to create truly self-contained Serverless to reduce the time required to add functionality to a website and simultaneously reduce the costs for global scale. Garrett McGrath [24] discusses different serverless platforms and frameworks, including AWS Lambda, Azure Functions, Google Cloud Functions, Apache OpenWhisk, IronFunctions, and OpenLambda, highlighting their features, capabilities, and explores real world applications utilizing these platforms, and explore how existing applications can be adapted to run in serverless environments. Additionally, it presented a novel performance-oriented serverless computing platform implemented in .NET, deployed in Microsoft Azure, and utilizing Windows containers as function execution environments.

Nabil El Ioini et al. [25] investigate different platforms that enable serverless architecture at the edge of computing. It explores the emerging trend of leveraging the main edge computing platform infrastructure to host serverless functions closer to end-users. Also, it compares their characteristics, identifies issues, and provides research directions. Vijay Prakash et al. [26] provides thorough guidance for developing a serverless web-based blogging application using CloudyPages that utilizes AWS services to achieve security, scalability, and availability. Manoj Kumar [27] compares various architectural patterns and deployment models within the serverless paradigm such as AWS, Azure, Google Cloud Platform, and some open source. Comparison criteria focused on compute, storage, database, messaging, API management, and tooling. Also, it emphasizes how it maximizes the benefits of serverless computing in real-world web applications. Khoi Huynh [28] discusses modern popular architectures of web development based Serverless applications and how to create a serverless application from scratch. Also, it evaluates their implications for web application development. Jodhan Saji and Ashish Kumar [29] discuss the implications of serverless computing on web application development, and key characteristics and use cases of serverless computing for web applications. Also, it presents a general architecture for web application development based on serverless computing. Pubali Datta et al. [30] investigates Valve framework which is a serverless platform that enables developers to enhance the security of function workflows on serverless computing platforms through the ability to defend against known serverless attack behaviors such as container reuse-based persistence and data exfiltration. Also, it discusses how Valve provides web application developers with the function workflow lifecycle, including function invocation, data access, and inter-function communication. Alfonso Pérez et al. [31] addresses the challenge of achieving high throughput in serverless computing environments and describes a middleware implementation that supports the execution of customized execution environments based on Docker images on AWS Lambda. The proposed approach allows efficient utilization of compute resources and improve throughput.

Aleksi Pekkala [32] explores and surveys applicable design patterns of serverless computing models and the five common ones were applied in migration process. The migration outcome is evaluated in terms, which typically include scalability, cost-effectiveness, and operational simplicity. Josef Spillner et al. [33] explores several demanding computing tasks based on FaaS models and compares them with conventional monolithic algorithm execution. Also, it investigates how serverless architectures can offer advantages and provides a roadmap for future work. M.Gospodinov and Evgeniya Gospodinova [34] explore the application of serverless architecture in the healthcare domain, specifically focusing on HRV analysis and presents a web application based on serverless architecture for analysis and evaluation of HRV by applying linear and nonlinear mathematical methods. Bangar Cherukuri [35] addresses the challenge of maintaining web development standards across multiple devices using serverless computing and hybrid optimization techniques through artificial intelligence evolving hybrid optimization techniques such as genetic algorithm and particle swarm optimization techniques. Matt Crane and Jimmy Lin [36] presents a novel application of serverless architectures in the domain of information retrieval systems using AWS, and postings lists stored in the DynamoDB NoSQL and the postings traversal algorithm executed in the Lambda service.

In the second level, some of them have contributed to the area of integrating serverless with CMS. Khondokar Solaiman and Muhammad Adnan [37] introduces a novel architecture which is called WLEC, that designed to minimize the cold start time in serverless computing environments for hosting CMS web applications. The WLEC was implemented in OpenLambda and evaluated in both the AWS and local VM environments, where the average start time decreased by 31%. Pauli Huhtiniemi [38] delves into the concepts of microservices, CMS, and serverless computing within a video on demand services, various examples of serverless microservices are presented, then proposed a suitable use case for serverless approach for video on demand services. Harsh Joshi [39] discusses how WebAssembly (Wasm) can enable portability, security, and scalability in cloud infrastructure and serverless computing environments by allowing developers to run code written in multiple languages across different platforms such as docker and cloud, microservices and CMS. Also, it evaluates the impact of Wasm, and the results showed that Wasm provides better performance, small bite size, safety, safe sandbox, cross platform and cross architecture. Syed Afraz Ali [40] explores the multifaceted design considerations critical to deploying a robust e-commerce CMS platform, including microservices architecture, containerization, and serverless computing. Also, it examines the importance of implementing security measures such as encryption, access control, and monitoring to ensure robust, secure, and efficient online retail operations. Niko Nummi [41] investigates the feasibility of utilizing a FaaS or serverless computing as a front-end server for hosting web applications and CMSs. It tests the proposed architecture for the front-end layer based on AWS Lambda, and some testing aspects are included such as performance, scalability, and cost-effectiveness. Thomas Smart [42] describes how Serverless and cloud-native systems work, their benefits and roles in automating and optimizing for web applications and web CMSs. Also, it explores various use cases and scenarios for implementing event-driven applications to backend services and data processing tasks.

In the third level, some of them have contributed to the area of integrating serverless with headless CMS. Prayudi Utomo [43] presents a comprehensive guide to show how to implement the JAMstack approach based on serverless functions for designing and building web applications and headless CMS, then hosts it on GitHub. Also, it discusses the testing results and shows that the presented approach can run with good performance, although it is connected to other services through an API. Raymond Camden and Brian Rinaldi [44] illustrate in their book how to use Jamstack frameworks to create fast, secure, and scalable web applications and headless CMSs by utilizing the following technologies, which are JavaScript, APIs, and Markup, to create fast, dynamic pages without the overhead of heavyweight frameworks. Juho Vepsäläinen et al. [45] explores the intersection of edge computing, headless CMS, and static site generation (SSG) to create performant web applications that are easy to host by reducing latency and improving content delivery. Danny Sebastian et al. [46] compares traditional CMS, headless CMS and SSG in building web applications. There are 10 criteria used to compare CMS and SSG, namely components, website type, service speed, flexibility, security, source-control, development speed vs website scale, dynamic content, admin page, and hosting. Jarkko Uro [47] explores the concept of building headless WordPress sites in a serverless environment based on LAMP architecture using "Less Server WP" plugin which offers a serverless solution to the web application that pre-processed before publication. Pre-processed pages are typically published on a CDN service to improve performance, scalability, and reliability. Jeno Laszlo [48] discusses how to create a special UI element for the Contentstack, a headless Content Management System (CMS), and how it can benefit from temporal workflows to automate content publishing at predetermined times.

As the third level aligns most closely with the proposed architecture, all included studies were evaluated against this architecture, as shown in Table 1. The comparison considers factors such as purposes, employed technologies, targeted layers or components, conclusions, strengths, and limitations.

Table 1. *The Related Work Comparison in The Third Level*

Reference	Purpose	Used Technologies	Layers/Components	Conclusion	Strengths and Limitations
[43]	Building a serverless website on GitHub pages	GitHub Pages, JavaScript, HTML, CSS	Static Site Generation, Deployment	Demonstrates the feasibility of using GitHub Pages for hosting serverless websites.	**Strengths:** Enhances speed and security for static websites. **Limitations:** Lacks scalability for dynamic applications.
[44]	Exploring the Jamstack architecture and its applications	JavaScript, APIs, Markup, JAMstack	Static Site Generation, API Integration, Markup	Provides a comprehensive guide on building modern web applications using the Jamstack model.	**Strengths:** Suitable for fast, secure small-scale web applications. **Limitations:** Limited scalability; lacks auto-scaling features.
[45]	Examining the implications of edge computing for static site generation	Edge Computing, Static Site Generators	Static Site Generation, Edge Computing Integration	Highlights the benefits of edge computing for enhancing static site performance and scalability.	**Strengths:** Reduces latency for static sites. **Limitations:** Does not address dynamic content management or automated scaling.
[46]	Reviewing technology trends in web development for small-scale websites	Various Web Technologies	Web Development Trends, Small-Scale Web Solutions	Identifies current trends and technologies effective for small-scale web development.	**Strengths:** Useful criteria for CMS selection for small sites. **Limitations:** Lacks support for dynamic content or large-scale scalability.
[47]	Migrating WordPress records to a CDN environment for serverless operation	WordPress, CDN	WordPress Integration, Content Delivery Network	Discusses methods and benefits of deploying WordPress content in a serverless environment using "Less Server WP" plugin.	**Strengths:** Improves performance for static content with CDN. **Limitations:** Limited to static content; lacks broader scalability and flexibility.
[48]	Adding scheduled publishing functionality to Contentstack using Temporal	Contentstack, Temporal	Content Management, Scheduled Publishing	Demonstrates an approach to enhance Contentstack with scheduled publishing capabilities.	**Strengths:** Enhances content scheduling capabilities. **Limitations:** Does not utilize serverless computing for high-demand scenarios.
The Proposed Architecture	Optimizing web applications development by integrating headless CMS and serverless computing	a set of nineteen integrated tools such as Gatsby, Webpack, OWASP ZAP, Contentful, AWS Lambda.	There are three main layers which are presentation layer, headless CMS layer, serverless layer	Reduces complexity and improves scalability and reliability for web application development process.	**Strengths:** Offers scalability, flexibility, and reduced operational complexity by integrating headless CMS and serverless computing. **Limitations:** May require advanced technical knowledge for initial setup.

Compared to previous and related studies, the proposed architecture provides a more integrated and effective framework for web application development process. It combines the flexibility of headless CMS with the scalability and cost-efficiency of serverless computing.

3. The Proposed Architecture

The proposed architecture will be presented to create web applications without the need for coding and setting up hosting environments on web servers. This is achieved by integrating the functionalities of headless CMS with serverless computing. headless CMS enables the creation of websites without coding, while serverless computing eliminates the need for developers to handle hosting environment setup. The proposed architecture comprises three primary layers as shown in Figure 1, which are presentation layer, headless CMS layer, serverless layer. Below, the details of each layer and its components are presented.

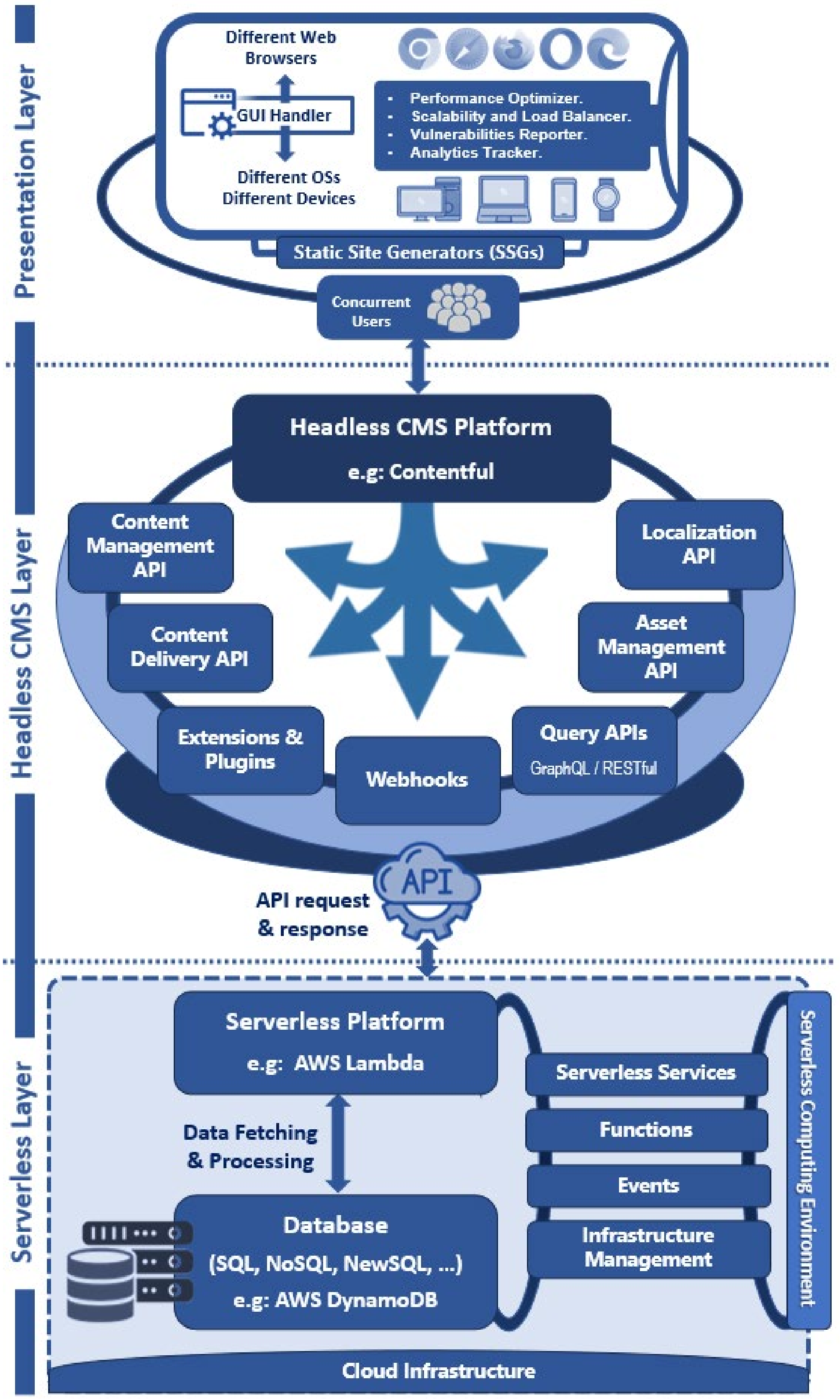

Fig. 1. The Proposed Architecture

3.1 Presentation Layer

The presentation layer plays a critical role in providing a smooth, easy, secure environment while using the web application as shown in Figure 2.

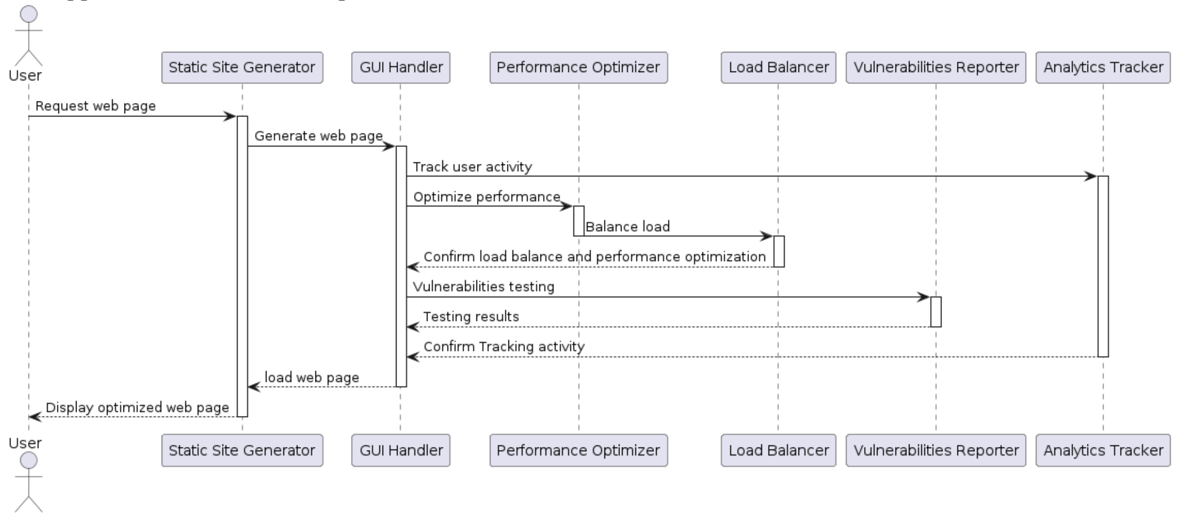

Fig. 2. The Sequence Diagram for Presentation Layer

This layer enables concurrent users to interact with the web application from heterogeneous environments using a set of components in this layer that integrate with each other to achieve the requirements of this layer. These components are:

- Static Site Generators (SSGs): combine content and templates into a fully static website and the content can be generated from different APIs sources. Thus, the website is generated and ready to serve users ahead of time, and because they are pre-built, they can load very quickly in users' browsers and doesn’t have any server-side features, which makes them less susceptible to attacks and data breaches.
- GUI Handler: providing the easiest and most powerful functions by using visual elements such as icons, buttons, and most menus that make interaction more intuitive. This component provides the following:
    - providing responsive design to ensure that the User Interface (UI) adapts gracefully to different screen sizes and devices.
    - supporting multiple languages and adapting date formats, currency symbols, and other cultural conventions based on the user's locale, as well as providing tools for translators to localize content effectively.
    - supporting heterogeneous platforms and browsers and users can access using different web browsers (e.g., Chrome, Safari, Firefox, Edge) and different operating systems (e.g., Windows, Linux, macOS).
    - managing and handling errors that are defined by HTTP status codes, including the method and scenario for dealing with each error.
- Performance Optimizer: minimizing page load times and optimizing content delivery (such as images, CSS files, and scripts) by adopting caching techniques to reduce latency and improve overall performance.
- Scalability and Load Balancer: handling concurrent user interactions and fluctuations by using different techniques such as load balancing, resource optimization, and horizontal scaling to ensure that the application can scale efficiently to meet growing demand.
- Vulnerabilities Reporter: preparing an interactive sitemap for the web application by carrying out a recursive crawl and dictionary-based probes. The resulting map is then annotated with the output from an active vulnerability such as injection attacks, cross-site scripting (XSS), and broken access control.
- Analytics Tracker: tracking and monitoring user interactions, behavior, and performance metrics. By collecting and analyzing data on user engagement, navigation patterns, and conversion rates,

this enables the proposed architecture to data-driven decision-making and optimization of the user experience.

3.2 Headless CMS Layer

This Layer serves as the backbone of proposed architecture and gains the flexibility to create and manage content, also enabling content to be delivered across various platforms and devices seamlessly as shown in Figure 3.

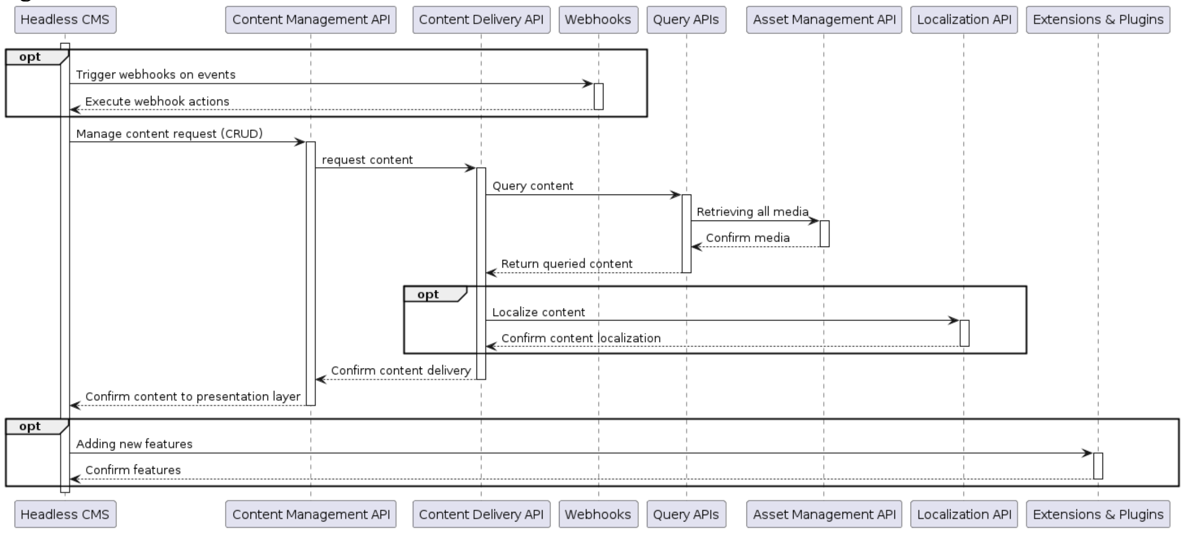

Fig. 3. The Sequence Diagram for Headless Layer

This layer comprises a suite of components which includes robust APIs and tools that facilitate comprehensive content management, seamless integration, and dynamic delivery. These components are:

- Content Management API: this API is responsible for providing the ability for creating, updating, deleting, and managing content. Also, it has the ability to the following:
  - Tracking change versions of content entries and retrieves or reverts to previous versions of content.
  - Managing metadata like tags, categories, and SEO fields.
  - Managing content workflows including draft, review, and publish states.
  - Providing content editors and stakeholders with the ability to preview changes to content before they are published.
- Content Delivery API: this API is responsible for delivering content to the presentation layer and ensuring that the right content is displayed in the right context and maintaining consistency across different channels and generating the content in a format that's consumable by the request from presentation layer.
- Webhooks: providing automated messages that are sent to other applications with real-time information each time an event happens. Therefore, it allows web applications to communicate with each other for the purpose of integration and performing automated tasks.
- Query APIs: this API communicates directly with the Content Management API to fetch the content, where it uses a set of sub-APIs such as GraphQL and RESTful APIs to connect to content repository sources, even if they are in different multiple sources. Also, it can explore the schema and understand the available data and operations.
- Asset Management API: is responsible for retrieving and storing all media assets of all types, such as video, audio, images, and text files and provide the ability for the following:
  - Transforming assets by resize, crop, or modify media assets.
  - Optimizing and minimizing load times and bandwidth usage by using different compression methods.
  - Monitoring and ensuring copyright information to media assets.
  - Integrating with content and ensuring cohesive and visually appealing presentations across digital channels.

- Localization API: is the process of making the content local, where different content is needed for different people in different places at different times. This API is responsible for generating content appropriate for web application users, considering different user preferences, languages, regions, and cultural contexts and provide the ability for regionalization, where content can be tailored to specific geographic regions or cultural preferences.
- Extensions and Plugins: are responsible for enhancing the core functionalities of the web application by allowing users to add new features, integrations, and customizations to meet their specific needs, such as sharing content in social media platforms directly after added into web application.

3.3 Serverless Layer

This layer serves as the backend cloud layer that is responsible for handling the backend functionalities as shown in Figure 4.

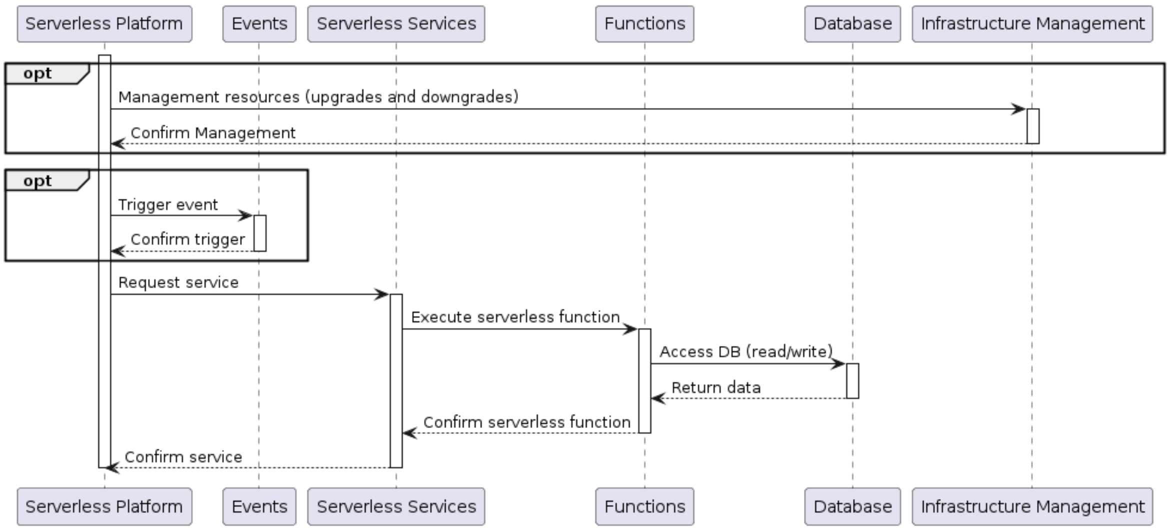

Fig. 4. The Sequence Diagram for Serverless Layer

This layer comprises a group of components that focus on managing the infrastructure of the proposed architecture, such as data storage, networking, resource monitoring, and triggers. These components are:

- Serverless Platform: is responsible for interfacing with headless layers and managing communication between other components in this layer. There are many platforms that can be used as serverless platform in this layer such as, Amazon Web Services (AWS) Lambda, Microsoft Azure Functions, Google Cloud Functions, IBM Cloud Functions, and Alibaba Cloud Function Compute.
- Events: are the triggers that initiate the execution of functions according to scheduled time or a set of rule condition are met, and they can originate from various sources within the proposed architecture.
- Serverless Services: are responsible for providing many functionalities that support and facilitate the execution of all components in this layer. These services handle various aspects such as authentication, notifications, data processing, monitoring, and logging.
- Functions: are small stateless independent units of code that perform specific tasks, and it designed to handle discrete units of work. The functions can be written in various programming languages such as PHP, Python, JavaScript, Java, or Go.
- Database: is responsible for providing scalable, reliable, and flexible data storage infrastructure to store, retrieve, and manage content efficiently and other data related to proposed architecture management. There are many types of content to be stored, including text, images, videos, and metadata. Also, it can handle both structured data and unstructured data using different types of databases at the same time such as, SQL, NoSQL, and NewSQL database management systems.
- Infrastructure Management: is responsible for automate handling of servers, storage, networking, and other resources required to run the proposed architecture.

4. The Proposed Architecture Implementation

The proposed architecture was implemented through a set of nineteen integrated tools to achieve the main functions for each layer as shown in Table 2, and the process of implementing this architecture will be addressed through discussing the used tools in each layer as in the following:

Table 2. *The Implementation Details for The Proposed Architecture*

#	Tool	To Achieve The Function of	Layer
1	Gatsby	Static Site Generators	Presentation Layer
2	React	GUI Handler	Presentation Layer
3	Webpack	Performance Optimizer	Presentation Layer
4	Application Load Balancer (ALB)	Scalability and Load Balancer	Presentation Layer
5	OWASP ZAP	Vulnerabilities Reporter	Presentation Layer
6	Google Analytics	Analytics Tracker	Presentation Layer
7	Contentful Management API (CMA)	Content Management API	Headless CMS Layer
8	Contentful Delivery API (CDA)	Content Delivery API	Headless CMS Layer
9	Contentful webhooks	Webhook	Headless CMS Layer
10	GraphQL	Query APIs	Headless CMS Layer
11	Contentful Asset API	Asset Management API	Headless CMS Layer
12	Localization API	Localization API	Headless CMS Layer
13	UI extensions	Extensions and Plugins	Headless CMS Layer
14	AWS Lambda	Serverless Platform	Serverless Layer
15	AWS EventBridge Scheduler	Events	Serverless Layer
16	AWS App Runner	Serverless Services	Serverless Layer
17	AWS Amplify	Functions	Serverless Layer
18	AWS DynamoDB	Database	Serverless Layer
19	AWS CloudFormation	Infrastructure Management	Serverless Layer

4.1 Presentation Layer

In this layer, many tools are used as open-source tools for ease of adaptation and integration with other tools, which are as follows:

- Gatsby: It is an open-source tool working as a static site generator in the proposed architecture and it's built-on top of Node.js using React and GraphQL.
- React: It is a JavaScript library that was developed by Meta and it's an open-source tool working as GUI Handler in the proposed architecture, and it's customized to integrate it with other components in the presentation layer to be capable for building front-end user interfaces based on components, then combine them to build single-page applications into entire screens, pages, and apps.
- Webpack: It is a free and open-source tool working as a performance optimizer in the proposed architecture. It can transform front-end assets such as HTML, CSS, and images into bundles and present them in a way that is efficient for the browser to load. It also provides a range of plugins and loaders to handle tasks like minification, code splitting, and tree shaking, contributing to a faster loading time for web applications.
- Application Load Balancer (ALB): is a type of Elastic Load Balancer (ELB) works at the Application layer of the OSI model and supported by Amazon, working as a scalability and load balancer tool in the proposed architecture that automatically distributes incoming application traffic across all the instances that are running.
- OWASP ZAP: is an open-source scanner tool working as vulnerabilities reporter in the proposed architecture that can work as a proxy server to detect and prevent many vulnerabilities such as, SQL injection, cross-site scripting (XSS), broken access control, security miss-configuration and broken authentication.
- Google Analytics: is a web analytics service offered by Google and it's working as an analytics tracker in the proposed architecture that tracking and analyzing web traffic.

4.2 Headless CMS Layer

In this layer, many tools are used based on the Contentful headless CMS for ease of adaptation and integration, the tools are as follows:

- Contentful Management API (CMA): is working as a Content Management API in the proposed architecture that manipulates creating, updating, deleting, managing content and metadata.

- Contentful Delivery API (CDA): is working as a content delivery API in the proposed architecture that ensures the right content is delivered to the presentation layer, maintaining consistency across different channels that is ready to be displayed to end-users.
- Contentful webhooks: is working as a webhook tool in the proposed architecture that allows other systems to react automatically to changes when content has been changed.
- GraphQL: is a fully functional equivalent to the REST implementation and working as a Query APIs in the proposed architecture that responsible for fetching the content repository.
- Contentful Asset API: is working as a Asset Management API in the proposed architecture that retrieving and storing all media assets of all types, such as video, audio, images, and text files.
- Localization API: is working as a Localization API in the proposed architecture that manages multilingual content effectively considering user preferences, languages, regions, and cultural contexts.
- UI extensions: is working as extensions and plugins in the proposed architecture that extend the functionality to meet specific needs of the web application.

4.3 Serverless Layer

In this layer, many tools are used based on the AWS for ease of adaptation and integration, the tools are as follows:

- AWS Lambda: is a serverless computing service provided by Amazon that works as a serverless platform in the proposed architecture and is responsible for handling incoming requests from headless CMS layer.
- AWS EventBridge Scheduler: is a scheduler provided by Amazon that works as an Events tool in the proposed architecture and is responsible for managing scheduling for running and creating tasks.
- AWS App Runner: is a fully managed service provided by Amazon that works as a serverless services tool in the proposed architecture and is responsible for simplifying the process of building, deploying, and scaling containerized services for web applications and web REST APIs.
- AWS Amplify: is a full-suite platform provided by Amazon that works as a function tool in the proposed architecture that simplifies the process of building scalable and secure web and mobile applications.
- AWS DynamoDB: is a fully managed NoSQL database provided by Amazon that works as a database tool in the proposed architecture and is a data repository that is responsible for storing and managing all data needed by web applications, such as web content, metadata, web templates, and end-user activities.
- CloudFormation: is a service provided by Amazon that works as an infrastructure management tool in the proposed architecture and is responsible for handling, configuring and provisioning the resources by using templates that are repeatable, testable and auditable manners to describe the AWS services.

To facilitate the implementation of the proposed architecture, developers can start by configuring the headless CMS to manage content independently from the frontend. Serverless functions, such as those deployed on AWS Lambda or similar platforms, can then be integrated to handle backend processes on demand. This setup allows for flexibility and scalability, as serverless functions automatically adjust resources based on real-time user requests. Additionally, APIs such as RESTful and GraphQL are utilized for seamless content integration across devices, making the architecture adaptable to different web application requirements. The simplicity of using serverless platforms, which require minimal server management, further enhances ease of use. Developers can focus on creating dynamic user experiences while relying on the system's automatic scalability and low-maintenance requirements.

5. Experimental Results

To investigate the effectiveness of the proposed architecture, the tested web application was hosted in AWS EC2 Instances. The tested web application is loaded by simulated traffic using Apache JMeter. Then, the testing results were analyzed and measured the performance of the tested web application. Also, AWS CloudWatch was used to analyze and measure system resource utilization. The testing environment configured by some parameters as shown in the following Table 2.

Table 3. *The Testing Environment for The Proposed Architecture*

Tool	Testing Environment Variables	Value
AWS EC2 Instance	Number of Instances	3
	Instance Type	m5.large
	Processor	2 vCPUs
	RAM	8 GB
	System Type	64-bit
Apache JMeter / AWS CloudWatch	Initial number of virtual users	20
	Maximum number of users	10000
	Step time duration	5 seconds
	Number of concurrent users added at each step	50
	Test time duration	20 minutes
	Server connect timeout	60 seconds
	Send request timeout	60 seconds
	Receive response timeout	300 seconds
	Request correlation timeout	300 seconds

Apache JMeter is used to generate a simulate ramp-up traffic by concurrent users at the same time and it is used to analyze and measure the performance of web applications, especially response time, throughput and error rates. On the other hand, the system resource utilization for CPU and RAM measured using AWS CloudWatch by monitoring the RAM and CPU usage for each AWS EC2 instance during the testing time.

The testing environment was configured to replicate real-world usage scenarios, employing AWS EC2 instances with 2 vCPUs and 8 GB RAM for each instance. Simulated user traffic was generated using Apache JMeter, starting with 20 users and scaling up to 10,000 concurrent users over a 20-minute period, with increments of 50 users every 5 seconds. The chosen benchmark metric (response time, error rate, and CPU/RAM utilization) were selected based on their relevance to web applications reliant on serverless architecture, as these metrics reflect the architecture's ability to handle high demand, maintain efficiency, and scale resources dynamically. This setup ensures the reproducibility of the experiment and aligns with the objectives of testing serverless and headless CMS integration for optimal performance.

The average response time is the total time taken to respond from all AWS EC2 instances during the selected time divided by the number of responses in the selected time and calculated by equation 1.

$$\mathrm{ART} = \frac{\sum_{\mathrm{i}=1}^{i=3}\left(\sum_{\mathrm{k}=1}^{\mathrm{k}=60}\mathrm{RT_k}\right)}{\sum_{\mathrm{i}=1}^{i=3}\mathrm{n}} \qquad (1)$$

Where:

- i is the number of AWS EC2 instance.
- k is the testing time in seconds.
- RT is the response tome in one second.
- n is the total number of generated requests.

The throughput is the number of handled requests by all AWS EC2 instances in a specific time and calculated by equation 2.

$$\mathrm{T} = \frac{\sum_{\mathrm{i}=1}^{i=3}\mathrm{n}}{\mathrm{m}} \qquad (2)$$

Where:

- i is the number of AWS EC2 instance.
- n is the total number of requests handled.
- m is the total time.

The error rate is the percentage of requests resulting in errors as compared to the total number of requests and calculated by equation 3.

$$\mathrm{ER} = \left(\frac{\sum_{\mathrm{i}=1}^{\mathrm{i}=3}FR}{\sum_{\mathrm{i}=1}^{\mathrm{i}=3}n}\right) \times 100 \qquad (3)$$

Where:

- i is the number of AWS EC2 instance.
- FR is the number of failed requests.
- n is the total number of requests.

The CPU and RAM utilization is the percentage that indicates at what point the system resources are fully utilized and no longer able to efficiently handle additional load and calculated by equation 4,5.

$$\mathrm{CPU_u} = \left(\frac{\sum_{\mathrm{i}=1}^{\mathrm{i}=3}\mathrm{TCPU}}{\sum_{\mathrm{i}=1}^{\mathrm{i}=3}\mathrm{h}}\right) \times 100 \qquad (4)$$

Where:

- i is the number of AWS EC2 instance.
- $TCPU$ is the actual time used.
- h is the available CPU time.

$$\mathrm{RAM_u} = \left(\frac{\sum_{\mathrm{i=1}}^{\mathrm{i=3}} TRAM}{\sum_{\mathrm{i=1}}^{\mathrm{i=3}} Q}\right) \times 100 \tag{5}$$

Where:

- i is the number of AWS EC2 instance.
- $TRAM$ is the actual memory size used.
- h is the available RAM memory size.

By applying equations 1, 2, and 3 during 20 minutes of testing and using a type of test called a ramp-up user load or stepwise load, which means increasing the number of concurrent users every period of time during the test, the testing results for the average response time, throughput, and error rate were figured out as shown in Figure 5.

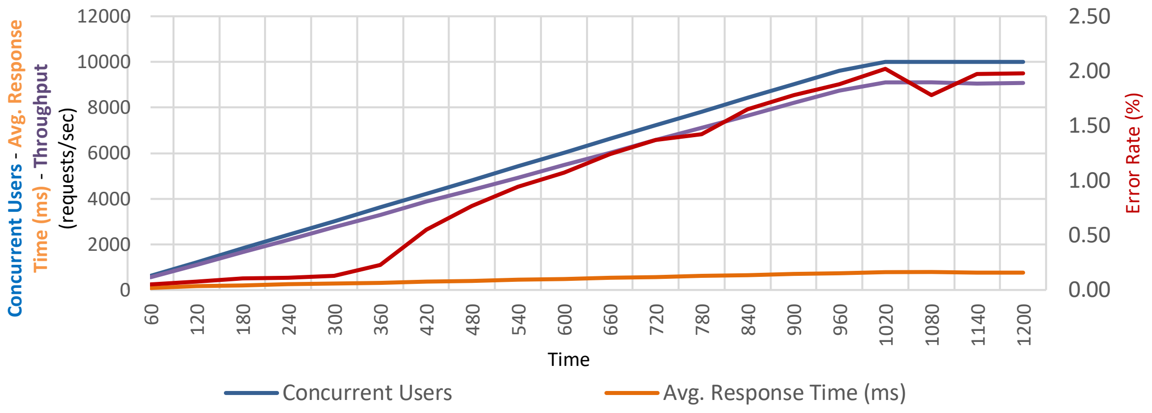

Fig. 5. The Testing Results for The Average of Response Time, Throughput, and Error Rate

Also, by applying equations 4 and 5 during the testing time, the testing results for CPU and RAM utilization were figured out as shown in Figure 6.

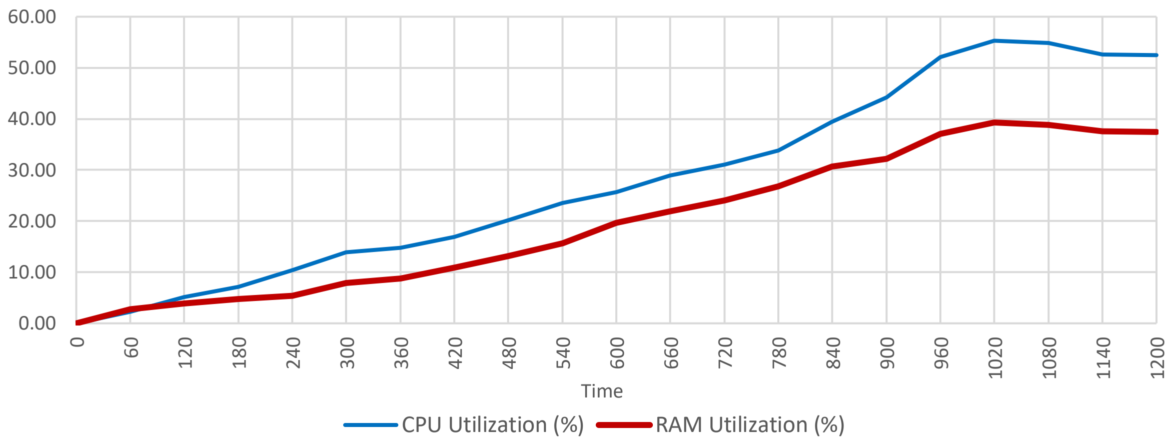

Fig. 6. The Testing Results for CPU and RAM Utilization

The testing of the proposed architecture was mainly based on two types of key performance metrics:

- Response Time, Throughput, and Error Rate: Figure 5 shows that the average response time ranges from 88 to 782 milliseconds, which indicates that the proposed architecture can perform effectively under increasing load conditions, up to 10,000 concurrent users. Response times show that the proposed architecture can perform under load. Throughput indicates good scalability during an increasing traffic load. The error rate increases slightly as the system gets more loaded, with a maximum of 2%, which is expected, and in the acceptance rate.

- CPU and RAM Utilization: Figure 6 shows that resource consumption efficiently utilizes CPU and RAM resources and is able to handle high loads up to 10,000 users, as the processor consumption rate does not exceed 56%, and the memory consumption rate does not exceed 39%.

The testing results demonstrate the efficiency and robustness of the proposed architecture across key metrics. Response time remained within an average range of 88–782 milliseconds, showing the system's capacity to manage increasing user demand with minimal delay, which is essential for responsive web applications. The error rate was consistently low, never exceeding 2%, indicating high reliability even under peak loads. This low error rate reflects the architecture's stability and its ability to handle concurrent requests without significant failures, thereby reducing downtime and enhancing user experience. Resource utilization was also efficient, with CPU usage not surpassing 56% and RAM usage staying below 39%, which shows the architecture's capability to scale resources dynamically according to demand. This efficient utilization contributes to cost-effectiveness, as resources are allocated only as needed, and prevents system overloads.

In comparison to similar studies which are discussed in the related work section, the proposed architecture offers significant improvements in scalability and operational flexibility. For instance, Utomo's [43] Jamstack-based serverless solution enhances speed and security for static sites but lacks the adaptability required for dynamic, high-demand applications. Camden and Rinaldi [44] focus on secure, fast applications using Jamstack but do not address the auto-scaling capabilities essential for large-scale deployments, which are achieved in the present work through serverless computing. Similarly, Vepsäläinen et al. [45] leverage edge computing to reduce latency in static content delivery; however, their model does not accommodate the dynamic content management provided by the current architecture. Sebastian et al. [46] review various CMS models, particularly suited to small-scale websites, but do not tackle the need for flexible, scalable content management, which is provided by integrating headless CMS with serverless functions in the proposed solution. Uro [47] explores the use of WordPress with CDN for improved static content performance, but this approach lacks the dynamic scaling and automation capabilities necessary for high-demand applications, an area where the proposed architecture excels. Lastly, Laszlo [48] presents an enhancement for scheduled content publishing in Contentstack using Temporal, which aids in scheduling but does not integrate serverless features to handle scalability and responsiveness. By integrating headless CMS with serverless functions, the proposed solution addresses these limitations, making it more suitable for high-demand applications where dynamic content and real-time responsiveness are critical. This comparison highlights the unique contribution of the proposed architecture in bridging the gaps identified in previous studies and providing a comprehensive solution.

Accordingly, the proposed architecture can handle large traffic loads while maintaining acceptable and good rates of the average of response time, throughput, and error rate and resource utilization for CPU and RAM. Therefore, the proposed architecture is compatible with real working environments and it's a reliable solution for web application development.

6. Conclusion

This paper presents an architecture that integrates headless CMS and serverless computing to create an efficient, scalable, and reliable framework for web application development. The proposed architecture utilizes headless CMS to enable content management without requiring extensive coding, and serverless computing to handle backend processes without the need for server management. The proposed architecture comprises three primary layers which are presentation layer, headless CMS layer, and serverless layer. Each layer performs its own functions by integrating a set of components. All integrated components are already available and most of them are open source. The proposed architecture tested through ramp-up traffic by10.,000 concurrent users at the same time and it can response with minimal impact on key performance metrics which are, response time, throughput, and resource utilization (CPU and RAM). The key performance metrics indicate that the system can effectively work as a suitable solution for real-world applications development and offer several advantages, including automatic scalability, cost efficiency, and a reduction in server management responsibilities.

Academically, this paper contributes to the field of serverless computing by offering a novel approach that bridges content management with scalable, event-driven infrastructure. Practically, it provides developers and organizations with a robust framework to build efficient, scalable applications with reduced operational complexity.

However, limitations exist; architecture relies on compatibility with serverless platforms and may require advanced technical knowledge for initial implementation. Future work could expand upon this architecture by exploring its application in various real-world scenarios, testing across different serverless providers, or enhancing support for complex data processing tasks. Additionally, further studies could investigate architecture's adaptability to alternative CMS frameworks and emerging cloud technologies.

References

[1] A. Singh, A. Chaudhary, and K. Chaudhary, "Content Management System," Global Journal of Enterprise Information System, vol. 15, no. 1, pp. 87-92, 2023.

[2] X.-A. Cao, "Headless CMS and Qwik Framework and Their Practicalities in the Future of Application Development," Ph.D. dissertation, School of Technology, Vaasa University of Applied Sciences, Finland, 2024.

[3] M. C. Enache, "Personalizing Shopping Experiences with Headless Commerce," Annals of the University Dunarea de Jos of Galati: Fascicle I, Economics & Applied Informatics, vol. 30, no. 1, pp. 71-76, 2024.

[4] L. Mezzalira, "Building Micro-Frontends," O'Reilly Media, Inc., 2021.

[5] R. Bailie and C. Evia, "Operationalizing Content Creation, From Start to Scale," Virginia Tech Publishing, 2024.

[6] A. Belila, "Reporting Website for a Localtapiola's Digital Service," MSc. dissertation, School of Technology, Vaasa University of Applied Sciences, Finland, 2024.

[7] S. S. Gill et al., "Modern computing: Vision and Challenges," Telematics and Informatics Reports, no. 100116, 2024.

[8] A. Miran, "The distributed systems landscape in cloud computing," Journal of Information Technology and Informatics, vol. 3, no. 1, 2024.

[9] G. Merlino et al., "FaaS for IoT: Evolving Serverless towards Deviceless in I/Oclouds," Future Generation Computer Systems, vol. 154, pp. 189-205, 2024.

[10] J. Scheuner and P. Leitner, "Function-as-a-service performance evaluation: A multivocal literature review," Journal of Systems and Software, vol. 170, no. 110708, 2020.

[11] A. Arjona, P. López, J. Sampé , A. Slominski and L. Villard, "Triggerflow: Trigger-based orchestration of serverless workflows," Future Generation Computer Systems, vol. 124, pp. 215-229, 2021.

[12] A. S. George, A. H. George, and T. Baskar, "Unshackled by Servers: Embracing the Serverless Revolution in Modern Computing," Partners Universal International Research Journal, vol. 2, no. 2, pp. 229-240, 2023.

[13] D. Mo, R. Cordingly, D. Chinn and W. Lloyd, "Addressing Serverless Computing Vendor Lock-In through Cloud Service Abstraction," in Proc. 2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Chicago, USA, Dec. 2023, pp. 233-241.

[14] I. E. Akkus et al., "SAND: Towards High-Performance serverless computing," in Proc. 2018 Usenix Annual Technical Conference (USENIX ATC 18), Boston, USA, Jul. 2018, pp. 173-185.

[15] V. Goar and N. S. Yadav, "Exploring the World of Serverless Computing: Concepts, Benefits, and Challenges," in Serverless Computing Concepts, Technology and Architecture, IGI Global, pp. 51-73, 2024.

[16] F. Tusa, S. Clayman, A. Buzachis and M. Fazio, "Microservices and serverless functions–lifecycle, performance, and resource utilization of edge-based real-time IoT analytics," Future Generation Computer Systems, vol. 155, pp. 204-218, 2024.

[17] G. A. S. Cassel et al., "Serverless computing for Internet of Things: A systematic literature review," Future Generation Computer Systems, vol. 128, pp. 299-316, 2022.

[18] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: a survey of opportunities, challenges, and applications," ACM Computing Surveys, vol. 54, no. 11s, pp. 1-32, 2022.

[19] O. Paakkunainen, "Serverless Computing and FaaS Platform as a Web Application Backend," MSc. dissertation, Metropolia University of Applied Sciences, Finland, 2019.

[20] C. Ivan, R. Vasile, and V. Dadarlat, "Serverless computing: An investigation of deployment environments for web APIs," Computers, vol. 8, no. 2, pp. 14-22, 2019.

[21] F. Samea et al., "A model-driven framework for data-driven applications in serverless cloud computing," PLOS ONE, vol. 15, no. 8, pp. 1-32, 2020.

[22] D. Zanon, "Building Serverless Web Applications," Packt Publishing Ltd., 2017.

[23] M. Ast and M. Gaedke, "Self-contained web components through serverless computing," in Proc. 2nd International Workshop on Serverless Computing, Nevada, USA, Dec. 2017, pp. 28-33.

[24] G. McGrath, "Serverless Computing: Applications, Implementation, and Performance," Ph.D. dissertation, University of Notre Dame, Indiana, USA, 2017.

[25] N. El Ioini, D. Hästbacka, C. Pahl and D. Taibi, "Platforms for serverless at the edge: a review," in Advances in Service-Oriented and Cloud Computing: International Workshops of ESOCC 2020, Springer International Publishing, Crete, Greece, Sep. 2020, pp. 91-102.

[26] V. Prakash, S. Jaiswal, S. Purohit, P. Singh and L. Garg, "CloudyPages: A Secure and Scalable Serverless Web-Based Blogging Application," in Proc. International Conference on Advances in Data-driven Computing and Intelligent Systems, Springer Nature Singapore, Goa, India, Sep. 2023, pp. 109-121.

[27] M. Kumar, "Serverless architectures review, future trend and the solutions to open problems," American Journal of Software Engineering, vol. 6, no. 1, pp. 1-10, 2019.

[28] K. Huynh, "The development of a web application: The new trend-Serverless application," BSc. dissertation, Turku University of Applied Sciences, Turku, Finland, 2020.

[29] J. Saji and A. Kumar, "A Review Paper on Serverless Architecture of Web Applications," in Proceedings of the KILBY 100 7th International Conference on Computing Sciences 2023 (ICCS 2023), Jalandhar, India, May 2023, pp. 1-6.

[30] P. Datta et al., "Valve: Securing function workflows on serverless computing platforms," in Proc. The Web Conference 2020, Taipei, Taiwan, Apr. 2020, pp. 939-950.

[31] A. Pérez, G. Moltó, M. Caballer and A. Calatrava, "A programming model and middleware for high throughput serverless computing applications," in Proc. 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, Apr. 2019, pp. 106-113.

[32] A. Pekkala, "Migrating a Web Application to Serverless Architecture," MSc. dissertation, Metropolia University of Applied Sciences, Finland, 2019.

[33] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and HPC," in High Performance Computing: 4th Latin American Conference, CARLA 2017, Springer International Publishing, Buenos Aires, Argentina, and Colonia del Sacramento, Uruguay, Sep. 2017, pp. 71-85.

[34] M. Gospodinov and E. Gospodinova, "Implementing a web-based application for analysis and evaluation of heart rate variability using serverless architecture," International Journal on Advanced Science - Engineering and Information Technology, vol. 9, no. 6, pp. 1927-1935, 2019.

[35] B. R. Cherukuri, "Maintenance of Web Development Standard for Multiple Devices with Serverless Computing through Cross Browser Affinity Using Hybrid Optimization," in Proc. 2024 IEEE International Conference on Computing Power and Communication Technologies (IC2PCT), Uttar Pradesh, India, Feb. 2024, pp. 1855-1859.

[36] M. Crane and J. Lin, "An exploration of serverless architectures for information retrieval," in Proc. ACM SIGIR International Conference on Theory of Information Retrieval, Amsterdam, Netherlands, Oct. 2017, pp. 241-244.

[37] K. Solaiman and M. A. Adnan, "WLEC: A not so cold architecture to mitigate cold start problem in serverless computing," in Proc. 2020 IEEE International Conference on Cloud Engineering (IC2E), Sydney, Australia, Apr. 2020, pp. 144-153.

[38] P. Huhtiniemi, "Experiences of Using Serverless Microservices on Video Publishing Platform," MSc. dissertation, Metropolia University of Applied Sciences, Finland, 2019.

[39] H. Joshi, "Analysis of web assembly technology in cloud and backend," International Research Journal of Modernization in Engineering Technology and Science, vol. 4, no. 9, pp. 121-128, 2022.

[40] S. A. Ali, "Designing Secure and Robust E-Commerce Platform for Public Cloud," The Asian Bulletin of Big Data Management, vol. 3, no. 1, pp. 164-189, 2023.

[41] N. R. J. Nummi, "Feasibility of a Function as a Service System as a Frontend Server for a Website," MSc. dissertation, Metropolia University of Applied Sciences, Finland, 2018.

[42] T. Smart, "Serverless Beyond the Buzzword: What Can Serverless Architecture Do for You?," Partridge Publishing Singapore, 2020.

[43] P. Utomo, "Building serverless website on GitHub pages," in IOP Conference Series: Materials Science and Engineering, vol. 879, no. 1, IOP Publishing, Bandung, Indonesia, Jun. 2020, pp. 123-130.

[44] R. Camden and B. Rinaldi, "The Jamstack Book: Beyond Static Sites with JavaScript, APIs, and Markup," Manning Publications, 2022.

[45] J. Vepsäläinen, A. Hellas, and P. Vuorimaa, "Implications of edge computing for static site generation," In Proceedings of the 19th International Conference on Web Information Systems and Technologies (WEBIST 2023), Rome, Italy, Nov. 2023, pages 223-231.

[46] D. Sebastian, I. Sembiring, E. Sediyono and K. Hartomo, " Literature Review and Survey of Website Development Technology Trends for Small Scale Websites," Information System for Educators and Professionals: Journal of Information System, vol. 7, no. 2, pp. 185-194, 2023.

[47] J. Uro, "Serverless WordPress: Moving WordPress records to a CDN environment," BSc. dissertation, Jamk University of Applied Sciences, Finland, 2021.

[48] J. Laszlo, " Adding Scheduled Publishing Functionality to Contentstack Using Temporal," BSc. dissertation, Tallinn University of Technology, Estonia, 2023.

المجلد (١٢) – العدد (١) – السنة (٢٠٢٥)
المجلة الدولية للحاسبات والمعلومات
متاح على الانترنت على الرابط : https://ijci.journals.ekb.eg/

تحسين عملية تطوير تطبيقات الويب: بنية مقترحة للتكامل بين أنظمة إدارة المحتوى بدون رأس والحوسبة بدون خادم

أحمد محمد منير عيسى

قسم نظم المعلومات الإدارية، كلية الإدارة، الجامعة الحديثة للتكنولوجيا والمعلومات، القاهرة، مصر

ملخص البحث

توفر أنظمة إدارة المحتوى بدون رأس مرونة كبيرة من خلال فصل مستودع المحتوى أو الجزء الخلفي عن طبقة العرض أو الجزء الأمامي. من جهة أخرى، تقدم بنى الحوسبة بدون خادم العديد من المزايا، بما في ذلك الأداء المعزز، التوسع التلقائي، التوافر العالي، الكفاءة في التكلفة، وتحسين إدارة وقت التنفيذ والموارد. يقترح هذا البحث بنية تجمع بين بنية الحوسبة بدون خادم ووظائف نظام إدارة المحتوى بدون رأس لتحسين عملية تطوير تطبيقات الويب، مما يسمح للمطورين بفصل إدارة المحتوى عن طبقة العرض مع الاستفادة من الطبيعة المعتمدة على الأحداث لوظائف الحوسبة بدون خادم. تهدف البنية المقترحة إلى تعزيز قابلية التوسع، تقليل تعقيد العمليات، زيادة التوافر، وتحسين الأداء خلال عملية تطوير تطبيقات الويب. تم تقييم البنية المقترحة من خلال اختباراها ومحاكاة الواقع، حيث أظهرت نتائج جيدة في وقت الاستجابة، معدلات الأخطاء منخفضه، والقدرة على استخدام الموارد تحت الأحمال الثقيلة. أظهرت النتائج وقت الاستجابة جيد يصل إلى متوسط يتراوح بين ٨٨ و٧٨٢ ميلي ثانية، ومعدل خطأ أقل من ٢٪، وكفاءة في استهلاك وحدة المعالجة المركزية والذاكرة العشوائية لم تتجاوز ٥٦٪ و٣٩٪ على التوالي. تشير نتائج الاختبار إلى أن البنية المقترحة مناسبة لبناء تطبيقات الويب الحديثة، حيث توفر إطار عمل قوي وفعال لتطويرها. بالإضافة إلى ذلك، يضمن مرونة البنية المقترحة القدرة على التكيف مع التقنيات المتطورة، مما يجعلها متوافقة مع التطورات المستقبلية في العديد من المنصات الرقمية. يساهم هذا البحث في مجال تطبيقات الويب من خلال تقديم نهج شامل يستفيد من إدارة المحتوى بدون رأس والحوسبة بدون خادم لتحقيق تطوير أكثر كفاءة وقابلية للتوسع.