Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 270 (2025) 2057–2065

Procedia Computer Science

www.elsevier.com/locate/procedia

29th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2025)

Optimizing WordPress Architecture: A next-generation approach with Micro-Frontends

Ștefan Secrieru[a,*], Sebastian Ștefănigă[a]

[a] *West University of Timișoara, Faculty of Mathematics and Computer Science, Bulevardul Vasile Pârvan 4, Timișoara 300223, Romania*

Abstract

The micro-frontend architecture represents a truly important paradigm shift in the realm of frontend web development. However, integrating it with the monolithic WordPress approach seems to be impossible due to the lack of valuable interoperability work in this area. The goal of this article is to increase the previously mentioned interoperability level between WordPress and the micro-frontend architecture. It presents improvements and optimizations to a WordPress solution that facilitates easy integration with micro-frontends: automatic performance measurements and rollback support for and elevated developer experience; multi-layered security architecture; real-world scenarios with data gathered from real users; static assets delivery unified experience with the WordPress ecosystem, for improved performance. Industry-scale testing shows an increase of more than 400% in the delivery stability of micro-frontend-enabled webpages with a minuscule decrease in the First Content Paint (FCP) key performance indicator of the application. This paper made unique and novel advances to the field of interoperability while taking into account many technical and real-world-related aspects.

Keywords: micro-frontend; WordPress; interoperability

1. Introduction

The micro-frontend architecture has gained a lot of traction in recent years as it promotes many software engineering ideals in the frontend realm. It represents the natural counterpart of the backend-specific microservices and aims to deliver, among other ideals, looser coupling, which represents a problem that is specific to the monolithical architecture often used in frontend development.

WordPress [13] is one of the largest Content Management Systems (CMS) out there, and Elementor [2] is, to this day, one of the largest WordPress front-end builders. However, there are situations where specific applications are pre-built in different frameworks and libraries, and this makes them very difficult to integrate with existing WordPress

[*] Corresponding author.
E-mail address: stefan.secrieru@e-uvt.ro

sites due to the lack of any valuable interoperability work in this area. The DevOps process required to host web applications developed in different programming languages, frameworks, and libraries is a rather tedious one even if virtual hosting is used. It requires specialist knowledge and does not integrate well with existing content management systems. There is also a notable difference between how these two architectures work: WordPress uses Server Side Rendering (SSR) to deliver complete webpages and micro-frontends use Client Side Rendering (CSR) to deliver a HTML skeleton from where JavaScript takes over to dynamically populate the page. This can potentially slow down the delivery of micro-frontends as any solution presented in the form of a WordPress plugin could potentially wait for the core functionalities of the content management system to be loaded before being able to take over.

However, a plugin [9] that allows seamless integration between WordPress and React JS., Angular, Vue JS. and Vanilla JS. frontends exists and represents one of the very few solutions to bridging the gap between WordPress and micro-frontends. This solution takes care of all the technical details involved, such as: file parsing and serving, security, and common state management. Seamless integration with the WordPress ecosystem is facilitated through automatic rewrite rules that take over when there are requests for the micro-frontend-based components and through supporting various build tools as they exhibit technical particularities when it comes to interpreting local routes. The plugin focuses on overcoming the interoperability challenges that arise between incompatible technology stacks.

Building upon this solution [9], this paper aims to analyze and test the performance of the WordPress plugin with real-world, industry scale scenarios while also improving certain areas such as: security, static assets delivery, rollback support, micro-frontend split capabilities and automatic performance measurements for an improved developer experience.

Furthermore, the actual study is not limited to the WordPress ecosystem, as this architecture could be replicated to other systems and have an impact on broader web development practices and CMS interoperability. Additionally, WordPress is widely used as stated by [5]. It powers more than 17 million websites and 48% of the top 100 blogs. Moreover, it is used by more than 58% people according to a poll in the paper [5].

The paper is divided into 7 sections. The first one briefly introduces the micro-frontends architecture while emphasizing the lack of interoperability between it and the WordPress ecosystem. The second section focuses on analyzing related work in order to gain insight into the state of the literature with regards to frontend web development. Starting with Section 3, the main improvements and optimizations to the interoperability of WordPress micro-frontends are presented. Section 3 focuses on shedding light on the real-world testing performance, while Section 4 is all about improving the developer experience with automatic performance and monitoring tools. Section 5 sheds light on the security architecture proposed and employed. Section 6 expands the micro-frontends split from vertical to horizontal, unifies the delivery of static assets with the WordPress ecosystem, and enables versioning and rollback support for each and every micro-frontend used. Finally, the last section focuses on underlining conclusions and future research and development directions.

The solution and improvements to the WordPress ecosystem of this paper can be consulted through the official repository [10].

2. Related Work

Paper [1] transformed a monolithic website into its modular micro-frontend-based counterpart using a horizontal split strategy. The final result is composed on the client side using module federation while communication between different components is facilitated by injecting an event emitter inside each micro-frontend-based component. According to Antunes et al. [1], when studying the benefits of micro-frontends, all developers in their case study unanimously agreed that the micro-frontend architecture allows integrating diverse technology stacks into the same highly customizable project. With this in mind, there is a clear need for improvements and further study in the WordPress - micro-frontends interoperability area. Future work in filling this gap would bring many advantages of the microservices counterpart in the WordPress ecosystem.

Moreover, micro-frontends have many benefits as identified in [3]. They represent the counterpart of backend-specific microservices and can significantly improve the consistency of the user experience and user interface. This can be done through the use of a component library. Clear isolation between micro-frontends and independence can also be achieved by using iframes [3].

Performance can be improved by using micro-frontends, as seen in the paper [6]. The authors have implemented a web application using micro-frontends, following the atomic design principles, and recorded an improved total blocking time, the same first contentful paint and a worse largest contentful paint. Implementing a good caching strategy could potentially speed up the micro-frontend-constructed web application [6]. The importance and advantages of the micro-frontend architecture is underlined in [8]. Various ways of developing and integrating micro-frontends into webpages have been identified and described by the authors, i.e. types of composition and splits. The issues, motivations, and benefits are brought into the spotlight, with autonomous teams, independent development, and highly scalable operations being the most impactful. Moreover, paper [14] describes a system architecture based on the MOOA micro-frontends-enabled framework. The way this framework works is then explained and described by the authors.

Furthermore, paper [4] presents a way of improving maintainability of microservices by creating a micro-frontend-based client-side application. This approach features a backend that is split into multiple components based on their functionalities and then assigned to dedicated teams. The micro-frontends are bundled up behind a routing layer. The authors of [4] further define a plethora of architectural metrics including: number of operations, direct coupling, coupling factor and more, all designed to quantify the performance of the micro-frontend-based architecture. The paper states that using both micro-frontends and microservices greatly reduces the complexity of the system. This shows the relevance and importance of integrating the micro-frontend architecture with modern content management systems. However, the paper [4] does not take into account the WordPress ecosystem and its particularities.

Moreover, paper [11] shows multiple ways of implementing micro-frontends in a vanilla environment, not accounting for WordPress-specific particularities. The authors mention using IFrames, vertically splitting the frontends, shared event busses, varnish cache and even web components. While all of these provide insightful details on how to implement the micro-frontend architecture, they do not account for any particularities of the WordPress ecosystem.

To the best of the authors' knowledge, there have been no other studies focused on optimizing the WordPress architecture by using the next-generation micro-frontends architecture. The following sections are all about emphasizing the optimizations and improvements made into the WordPress-micro-frontends interoperability area.

3. Measuring real-life - production impact

Real life, production data have been gathered in order to validate the effectiveness of the interoperability solution in two phases. This endeavor is described in the following lines. The micro-frontend used in this study is a Single-Page Application (SPA) deployed on the Faculty of Mathematics and Informatics of West University of Timisoara website. The first phase consists of gathering data related to the DOM Content Loaded (DCL), Largest Contentful Paint (LCP), and First Contentful Paint (FCP) registered for the WordPress native-like approach: Elementor. This is made possible by utilizing a PHP plugin that determines a specific page of the website to measure the before-mentioned performance indicators and report them back to the server, where the plugin stores them in the database alongside their creation timestamp. This eliminates the bias of a locally hosted web application and reports real-world data, as it is experienced by the end-user.

The second phase is all about silently changing the specified page, without prior announcement, with one created exclusively in Vanilla JS deployed using the interoperability plugin [9] inside the already running WordPress website. It is vital to note that the performance-measuring plugin was left in place as is and not modified between the two phases, and the data gathered was exclusively from end-users, with a large variety in terms of times and days of accessing the webpage.

A total of more than 3800 records were collected and analyzed for each phase. The Elementor page had 4360 records gathered, while the Vanilla JS micro-frontend had 3876. After removing the outliers, the Elementor build exhibited only 3887 records, and the Vanilla JS-constructed webpage only dropped to 3840. This shows a large improvement in terms of stability for the micro-frontend deployment. The results of the first stage can be consulted in Table 1.

Analyzing the results obtained in Table 2, a massive increase in outlier stability can be seen when the target website is deployed as a micro-frontend component. This means that the micro-frontend is more consistent in terms of delivery performance than the Elementor approach with an increase of more than 400%. As stated by Kunštnár et al. [6], micro-frontends are not without flaws: the identified solution shows close to a 35% increase in the time needed to load the

Table 1. Performance indicators recorded for both scenarios.

Technology / Key Performance Indicator	Elementor - WordPress	Vanilla JS - micro-frontend
DCL	1774.77	2397.76
FCP	1221.07	1320.52
LCP	1325.64	1807.86
Total Outliers	1238	244

Table 2. Performance changes, percentage-wise.

Key Performance Indicator	Vanilla JS - micro-frontend
DCL	+35.1%
FCP	+8.14%
LCP	+36.38%
Outlier Stability	+407%

Document Object Model (DOM) and a 36% increase in the largest contentful paint. However, the identified plugin bridges the gap between one of the largest content management systems out there and the next-generation approach to front-end development. The increase in the DCL metric suggests that the page takes a longer time to actually load and parse the HTML and the JavaScript code. This can be a result of using more synchronous JavaScript and making a lot more network requests as the micro-frontend might require a plethora of static assets. The increased LCP suggests the main content of the page takes a long time to appear on screen. CSS and JS bloat could potentially be one of the causes as the micro-frontends are not optimized to share pieces of code.

4. Performance and scalability measuring support

In order to enhance the developer experience exhibited by the interoperability plugin, various measuring tools have been added. These include: (a) static asset serving failures; (b) micro-frontend serving performance indicators.

Whenever an error in (a) is encountered in the provisioning of any static assets, it is logged and presented to the developer to offer insight into such problems. The (b) performance indicators are calculated with the help of a custom script that gets injected into the ”index.html” file whenever a new micro-frontend is added. If the user opts in to measure the performance of the micro-frontends, the plugin will take care of all previously added ones with no manual intervention required.

From a technical point of view, the custom script utilizes the Google Web Vitals package [12] that is loaded into each micro-frontend via a CDN. The returned page to the user will begin computing the performance metrics as soon as possible and send them back to the WordPress site, where a custom REST route has already been created by the plugin when the user opted to measure the performance. The plugin proceeds to store the received data alongside the slug of the micro-frontend for later identification. Storing the performance metrics is done in a cyclic way: whenever a user-defined limit is reached, the oldest ones get replaced. This way developers always see fresh data aggregated in their dashboard. The authors of this paper acknowledge the latency introduced by loading the Google Web Vitals package to facilitate accurate measurement of key performance indicators arguing that it is rather small. Since it is included in every micro-frontend that opts in for performance measuring, it does not create discrepancies between the modular components of the web application.

5. A state of the art approach to multi-layered security

This section details how the solution moves away from the flawed strategy of security through obscurity. The complete security flow can be consulted in Figure 1 and consists of a multi-layered approach. The following layers are part of the proposed security architecture: (a) Origin layer, used to prevent CSRF attacks; (b) Nonce layer, used to prevent replay attacks; (c) HMAC layer, used to prevent Man-In-The-Middle (MITM) attacks; (d) key signing layer, used to secure the transportation of the nonce.

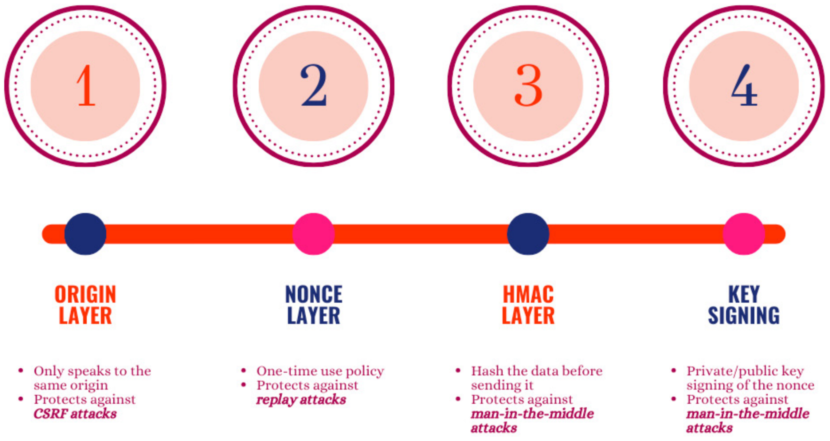
1
2
3
4
ORIGIN LAYER
• Only speaks to the same origin
• Protects against *CSRF attacks*
NONCE LAYER
• One-time use policy
• Protects against *replay attacks*
HMAC LAYER
• Hash the data before sending it
• Protects against *man-in-the-middle attacks*
KEY SIGNING
• Private/public key signing of the nonce
• Protects against *man-in-the-middle attacks*

Fig. 1. Overview of security layers and their intended purpose

The very first layer of defense, i.e. the (a) origin one, sets the bedrock for the next, more specialized protection layers by firstly verifying if the request comes from the same origin as the WordPress instance. This helps in mitigating CSRF attacks. The origin of any request can be forged, which makes this protective measure only a basic one. Next, the second layer of defense, referred to as the (b) nonce layer, is used to prevent replay attacks and works by first acknowledging that a malicious user could replay requests in order to overload the server or to insert corrupted or fake data. This would not be possible when making use of this security layer because the client has to request a randomly generated nonce from the server before talking to it. The nonce follows a one-time use policy: it gets discarded right after being used the first time. This way, no nonce could be replayed multiple times. Furthermore, the third security measure i.e. (c) the HMAC layer effectively protects against man-in-the-middle attacks by hashing and sending the data to the server that will proceed to re-hash the received data. If there are changes to the data received, the hashes will differ and the plugin will drop the request. Finally, the (d) key signing layer encrypts the nonce with the public key of the server before being sent back to it. This further protects against MITM attacks as it removes the ability of an attacker to recompute the hash of the intercepted and modified data by taking advantage of the plain-text nonce that is part of the HMAC hashing secret.

In summary, it is very important to understand that, when talking about a non-authenticated, open route, there can be no complete security measurements employed. Attackers could find ways to trick the system, but the plugin hardens this endeavor by using a multi-layered security paradigm. The performance measurement option that is presented to the user poses a security risk towards unauthorized, spam requests. It is thus secured using the above-mentioned security architecture, but still remains vulnerable due to being an open, non-authenticated route.

6. Streamlining micro-frontends. Horizontal deployment, media management, and rollback support

Further enhancements were made to the cross-platform plugin: expanding the deployment capabilities to also accommodate horizontally split micro-frontends; unifying the native WordPress media management with the micro-frontends static assets delivery solution; enabling versioning and rollback support. The identified interoperability solution only accommodates micro-frontends as vertically split ones. In order to accommodate the popular horizontal split of micro-frontends, the following work flow was employed by this paper: the user / developer must upload an archive consisting of the following three mandatory files: (a) an ”index.html” file; (b) a JavaScript file; (c) a CSS file. The complete flow of the horizontal split for the micro-frontends can be consulted in Figure 2.

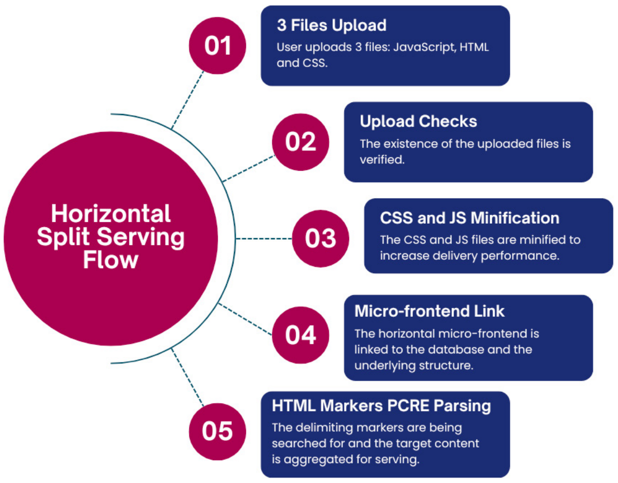
Horizontal Split Serving Flow
01
3 Files Upload
User uploads 3 files: JavaScript, HTML and CSS.
02
Upload Checks
The existence of the uploaded files is verified.
03
CSS and JS Minification
The CSS and JS files are minified to increase delivery performance.
04
Micro-frontend Link
The horizontal micro-frontend is linked to the database and the underlying structure.
05
HTML Markers PCRE Parsing
The delimiting markers are being searched for and the target content is aggregated for serving.

Fig. 2. Horizontal split flow of the proposed solution

Inside the (a) index.html file, there has to be at least one segment of code delimited by special markers that will be picked up by the plugin and outputted in the form of a WordPress shortcode whenever necessary. If there are multiple such segments, they will all be aggregated in the final shortcode result. The (b) JavaScript file and (c) CSS file will be served in a minified version of what was uploaded by the developer whenever there is a request for the specified shortcode. Minification is done using the Mathiasmullie minify PHP package [7]. This improves the performance of the solution by only loading the necessary CSS and JS code when needed, with no wasted bytes on spaces and newline characters.

The content of the three uploaded files is up to the developer, but the authors of this study recommend they only contain code related to the specific part of the micro-frontend that is going to be served as a shortcode. This is done to improve performance, reduce redundancy, and minimize clashes and conflicts with other pieces of code.

If a certain micro-frontend, be it a vertically or horizontally split one, utilizes images, it must be uploaded to the build archive of the micro-frontend by the developer. Although this is done automatically by modern build tools such as Vite and CRA, using the images from the regular WordPress uploads folder represents a valuable extension point. This is because: (a) WordPress uses caching, optimizers, resizers, and even CDN delivery for static assets either through a third-party plugin or through core functionalities; (b) this way images can be maintained in a single, centralized place; (c) static media files are not being redundant and duplicated. The whole process can be consulted by referring to Figure 3.

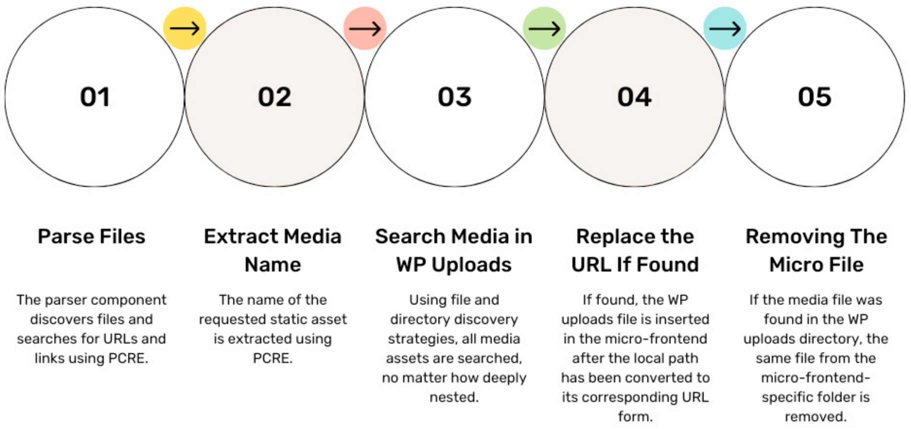
01
02
03
04
05
Parse Files
The parser component discovers files and searches for URLs and links using PCRE.
Extract Media Name
The name of the requested static asset is extracted using PCRE.
Search Media in WP Uploads
Using file and directory discovery strategies, all media assets are searched, no matter how deeply nested.
Replace the URL If Found
If found, the WP uploads file is inserted in the micro-frontend after the local path has been converted to its corresponding URL form.
Removing The Micro File
If the media file was found in the WP uploads directory, the same file from the micro-frontend-specific folder is removed.

Fig. 3. Flow of media assets optimization

Optimizing media management is done in multiple stages: (a) parsing the files; (b) extracting media names; (c) searching for the media name in the WP uploads directory; (d) replacing the URL; (e) removing the file. First, the (a) parser goes over every file inside the uploaded micro-frontend no matter how deeply nested it might be. Using the PHP PCRE engine, occurrences of URLs are found in the parsed files. Then, the (b) media file name is extracted from the identified URLs and links. The following is the lookup process where the target file is being (c) searched for in the WordPress ”uploads” directory. If a file with a matching name and extension is found, the original URL is replaced (d) with a new one, pointing to the already present resource.

However, if no file is found, the requested one will be moved to the WordPress uploads directory by the plugin, with no manual intervention required from the developer of the web application. This is done by actually relocating the file and registering it in the database. All operations performed at this stage on the database are atomic; they must all be completed in order to have a successful migration. Finally, the target media file is (e) removed from the upload directory of the micro-frontend.

This process allows developers to leverage the WordPress optimizations of static assets and manage all of them in a centralized flow. In addition, to further enhance the developer experience, a rollback system has been developed. It can be consulted in Figure 4.

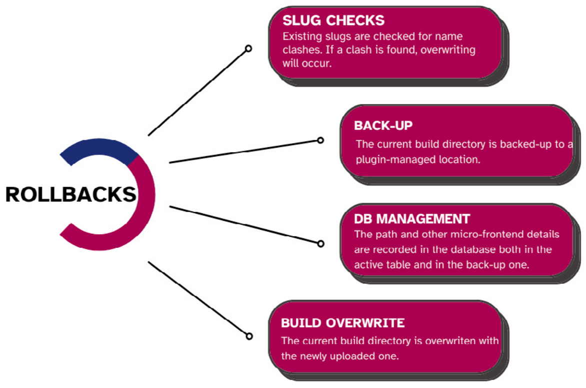
ROLLBACKS
SLUG CHECKS
Existing slugs are checked for name clashes. If a clash is found, overwriting will occur.
BACK-UP
The current build directory is backed-up to a plugin-managed location.
DB MANAGEMENT
The path and other micro-frontend details are recorded in the database both in the active table and in the back-up one.
BUILD OVERWRITE
The current build directory is overwriten with the newly uploaded one.

Fig. 4. Rollback and overwriting flow

The flow mentioned above can be described in multiple steps: (a) slug existence checks; (b) backup the current build directory; (c) overwriting the current build; (d) managing the database records; (e) moving the backup directory back in place when rolling back. First, whenever a new micro-frontend is deployed (a) all existing slugs are checked to account for name clashes. If such a slug is found, the plugin will continue with the versioning strategy. The following is the (b) backup process. The current build directory is copied over to another location inside the plugin, and the new build folder (c) overwrites the old one at the correct serving location. Moreover, the new backup directory is recorded inside the database (d) to allow for rollbacks on user request. The user is then presented, in a separate plugin subpage, with the ability to either delete back-ups or roll them back. Whenever a rollback is initiated, the micro-frontend it is moved back to the correct serving path, and the corresponding details related to its type, path, name, and creation timestamp are all updated accordingly.

7. Conclusions & Further Work

Evolving WordPress into a micro-frontend-compatible architecture represents a pivotal step in modernizing and optimizing content management systems, and this study has shown that the extensibility and maintainability of WordPress can be improved by shifting its approach to a more modular-based strategy.

This paper has strengthened the interoperability between two paradigms that are seemingly incompatible, i.e. content management systems and micro-frontends. The study focused on bringing into the spotlight real-life industry-scale scenarios, performance optimizations regarding static assets delivery, developer experience enhancements via automatic performance measuring capabilities, and rollback support. While server-side rendering and client-side rendering seem to be at odds, they can be merged together seamlessly with the right tools, approaches, and strategies. Furthermore, a multi-layered security approach has been proposed to properly secure the interaction between the server and the client micro-frontends. Thus, advances have been made in the field of interoperability within the world of web development.

Expanding the solution to other content management systems such as Joomla and Drupal represents one of the most interesting expansion points left by this study. Porting the solution to other CMSs would prove to be a difficult endeavor but could be possible with careful planning and execution. There are several technical translations that can be done to address this, such as changing the shortcode logic to the content preparation flow of Joomla and using the XML manifests instead of the settings API. When translating the plugin to the Drupal environment, shortcodes must be transformed into custom blocks or filters, HTML forms into the form API classes, and custom post types into entities. Migrating a WordPress-specific plugin form into the Drupal CMS implies transforming the regular HTML input fields into the Drupal form API class and building it inside an associative array where each item has its type, title, and metadata defined as properties of the associative array entry.

Moreover, integrations with no-code or low-code solutions like Elementor or the Guthenberg editor to allow non-technical WordPress users to develop webpages would be a very valuable future research direction. This would reduce the need for technical and programming expertise when migrating an existing monolithical WordPress website to a micro-frontend-based counterpart. With this in mind, even automatic migration tools from one architecture to another would represent a direction worth investigating.

Allowing micro-frontends to be installed as regular plugins inside the WordPress ecosystem and not like uploaded build directories in the master plugin itself would allow each and every micro-frontend component to be managed in a WordPress native way. This would also enable updates and sharing of micro-frontend splits between projects without the need to manually re-upload the newer version of the build directory every time. In this architecture, updates would be delivered in the native way, where developers can opt for automatic updates or manually run them. Configuration files and settings could also be exposed by the micro-frontends when installed as native plugins. In this way, they could even be customized by non-technical users.

Improvements in delivery performance are still needed, as seen in both the paper [6] and in this study. The first contentful paint is hurt by the paradigm shift alongside the DOM content loaded metric, while the outlier stability has seen a massive increase.

In conclusion, bridging the interoperability gap between content management systems and micro-frontends is not just about overcoming technical difficulties, but rather about embracing and using the next-generation approach to web development in all seemingly incompatible technology stacks without any added tech debt.

References

[1] Antunes, F., Lima, M.J.D.D., Araújo, M.A.P., Taibi, D., Kalinowski, M.: Investigating Benefits and Limitations of Migrating to a Micro-Frontends Architecture. In: Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software (SBES 2024). pp. 103–113. Sociedade Brasileira de Computação, Brasil (Sep 2024). doi:10.5753/sbes.2024.3303, https://sol.sbc.org.br/index.php/sbes/article/view/30353

[2] Elementor Developer Documentation (2025), https://developers.elementor.com/docs/

[3] Gashi, E., Hyseni, D., Shabani, I., Çiço, B.: The advantages of Micro-Frontend architecture for developing web application. In: 2024 13th Mediterranean Conference on Embedded Computing (MECO). pp. 1–5. IEEE, Budva, Montenegro (Jun 2024). doi:10.1109/MECO62516.2024.10577836, https://ieeexplore.ieee.org/document/10577836/

[4] Kaushik, N., Kumar, H., Raj, V.: Maintainability improvement of microservices based applications using micro frontend. International Journal of Computers and Applications **47**(2), 188 – 196 (2025). doi:10.1080/1206212X.2025.2450250, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85216192495&doi=10.1080%2f1206212X.2025.2450250&partnerID=40&md5=515505a1ebfad1397ac69998fd7256ff, cited by: 0

[5] Kumar, A., Kumar, A., Hashmi, H., Khan, S.A.: Wordpress: A multi-functional content management system. In: 2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART). pp. 158–161. Moradabad, India (2021). doi:10.1109/SMART52563.2021.9675311

[6] Kunštnár, V., Podhorský, P.: Micro Frontend Architecture. In: 2024 Zooming Innovation in Consumer Technologies Conference (ZINC). pp. 124–129. IEEE, Novi Sad, Serbia (May 2024). doi:10.1109/ZINC61849.2024.10579400, https://ieeexplore.ieee.org/document/10579400/

[7] Official GitHub repository of the Mathiasmullie Minify Package (2025), https://github.com/matthiasmullie/minify
[8] Peltonen, S., Mezzalira, L., Taibi, D.: Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review. Information and Software Technology **136**, 106571 (Aug 2021). doi:10.1016/j.infsof.2021.106571, https://linkinghub.elsevier.com/retrieve/pii/S0950584921000549
[9] Official GitHub repository of the identified interoperability solution (2025), https://github.com/CS-Research-Group-UVT/micro_web
[10] Official GitHub repository of the proposed solution (2025), https://github.com/CS-Research-Group-UVT/micro_web_improved
[11] Savani, N.: The future of web development: An in-depth analysis of micro-frontend approaches. International Journal of Computer Trends and Technology pp. 65–69 (11 2023). doi:10.14445/22312803/IJCTT-V71I11P109
[12] Web-vitals Developer Documentation (2025), https://web.dev/articles/vitals
[13] Wordpress Developer Documentation (2025), https://wordpress.org/documentation/
[14] Yang, C., Liu, C., Su, Z.: Research and Application of Micro Frontends. IOP Conference Series: Materials Science and Engineering **490**, 062082 (Apr 2019). doi:10.1088/1757-899X/490/6/062082, https://iopscience.iop.org/article/10.1088/1757-899X/490/6/062082