Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI

Bridging Front-End Frameworks (React/Angular) with Adobe Experience Manager Components

Dayasagar Vangala

AEM Developer Lead at Bank of America, Charlotte city, North Carolina State. USA

Email: *Daya55sagar55@gmail.com*

Abstract: The challenge of balancing both modern and dynamic front-end frameworks such as React and Angular with a sturdy and enterprise-qualified content management system such as Adobe Experience Manager (AEM) is a critical challenge and opportunity in the modern web engineering. This research paper gives an in-depth discussion of the architectural designs, the methods of implementation, and the performance implications of the bridging of these contrasting technologies. The importance of efficient integration methodology has taken precedence as organizations continue to attempt to provide the very interactive, personalized user experience whilst still retaining the content governance and scalability that are provided by the AEM. This paper finds and compares three main architectural models including the traditional In-Context Authoring model, the headless Content-as-a-Service model, and the hybrid SPA Editor model through literature review and comparison of integration models. The results reveal that the headless model has the highest front-end flexibility and performance, whereas the AEM SPA Editor framework is the only one with a distinct synergy by retaining in-context authoring capabilities to React and Angular applications, which is vital in marketing-oriented organizations. But this integration brings about complexities in the management of the state, data hydration and building process orchestration. The research finds that the decision on the strategy to follow in the integration process is not necessarily a technical, but a strategic one, depending on the project needs, the structure of the team, the working processes in the organization. This study presents a systematic guideline to the developers and architects to traverse this complicated integration environment, pointing out the best practices, some of the possible pitfalls, and the future of the convergence of CMS and front-end technologies.

Keywords: adobe experience manager (AEM), react, angular, front-end integration, SPA Editor, headless CMS, component-based architecture, web engineering.

http://doi.org/10.5281/zenodo.17922047

Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI

I. Introduction

The architecture of the web development of the enterprise is dominated by an ongoing conflict between the desire to have a strong content management system that is governable and the urge to have an app-like user experiences that are dynamic and effective. This opposition has put strong backend systems such as Adobe Experience Manager (AEM) in contrast to agile, reactionary front-end systems such as React and Angular. AEM has been the foundation of large organization since a long time and has offered integrated content creation, management and delivery with robust workflow and governance controls (Hernandez & Martinez, 2012). At the same time, React and Angular have taken the lead in the front-end world, providing a developer with unmatched efficiency in the development of multi-facet, interactive, and high-performance user interfaces. The key problem here is to fill this gap: to harness the content orchestration power of the AEM and give the development teams the modern tools and paradigms of such JavaScript frameworks.

The older model of integrating plain JavaScript into AEM has been unsuccessful in the development of complex Single-Page Applications (SPAs). The result of this approach can be a close integration of logic and presentation making developers less productive and not making the most of either system (Morales & Ortiz, 2011). The appearance of the so-called headless CMS pattern (when the back-end is used to deliver content through APIs to a decoupled front-end) was an attractive cure. This model gives the front-end developers the entire freedom to use their preferred content services framework to consume AEM as a content service (Lee and Kim, 2018; Santos and Costa, 2018). But the cost of this decoupling is great: the crucial connection between content authors and presentation layer is lost, as well as the in-context, WYSIWYG (What You See Is What You Get) authoring experience that forms a pillar of the AEM value proposition (Olsson and Svensson, 2013).

To address this issue, Adobe has created the SPA Editor framework which is a technological reconciliation as the might of a decoupled architecture versus the need of in-context authoring. This framework enables developers to create SPAs based on React or Angular, but lets the content authors edit and manage the content directly within the AEM authoring environment (Evans and Thompson, 2018). This hybrid model is an important development in enterprise CMS architecture, but it is not an easy one to implement. It presents a novel set of complexities, such as mapping AEM components to SPA components, data synchronization, and state management and optimization of performance (Acharya and Singh, 2016; Chen and Wang, 2017).

The possible aspects of this integration were investigated in existing literature, starting with early integrations of AngularJS with Adobe CQ5 (Fernandez & Lopez, 2013; Kumar and Reddy, 2014) and recent studies of React-based architecture (Gomez and Ruiz, 2016; Verma and Jain, 2016).

http://doi.org/10.5281/zenodo.17922047

The implications of performance (Evans and Thompson, 2018), pattern of component design (Nguyen and Pham, 2016), and API-driven has been considered (Xu and Li, 2017). Nevertheless, there is no synthesis that could compare the whole range of integration models between conventional and headless towards the hybrid SPA Editor. It is evident that the selection and implementation of these strategies require a holistic framework to be selected and implemented depending on certain organizational and technical requirements.

This gap in the research article is addressed by the systematic analysis of the methodologies of combining React and Angular with Adobe Experience Manager. This research aims to find out:

1. To outline and provide a critical comparison of the three prevailing architectural models of AEM and front-end framework integration: the In-Context Authoring model, the Headless CMS model and the Hybrid SPA Editor model.

2. To compare the specifics of technical implementation, data flow, component mapping and build processes of both React and Angular in the AEM SPA Editor framework.

3. To measure the performance, scalability and authoring experience trade-offs of each integration pattern.

4. To offer a framework of decisions and define best practices in the eyes of developers and architects in this complicated integration environment.

The accomplishment of these goals means that this article will serve as an essential tool to the enterprise teams that need to update their web presence without losing the content management features that platforms such as AEM offer them. The results will provide practitioners with the information that will allow them to take informed decisions related to architecture, predict potential challenges of the implementation process, and utilize the synergistic power of the AEM and contemporary front-end frameworks to provide better online experiences.

II. Methodology

This paper has utilized a method of systematic analysis by examining and comparing the different approaches to the integration of React and Angular front-end frameworks and Adobe Experience Manager (AEM). The study was intended to be a qualitative, comparative study of the architectural patterns based on the evidence data provided by the review of the existing literature and implementation practices recorded. The main aim was to generalize a consistent taxonomy of integration models and analyze their nature, as opposed to producing new empirical data by way of experimentation.

http://doi.org/10.5281/zenodo.17922047

5.1 Research Design

The study took a descriptive and analytical paradigm and was organized based on a systematic literature review and a systematic comparison of architectural paradigms. It was carried out by finding and sorting and accounted documentation of integration strategies, both in academic literature and in those with high quality technical resources of industry practice. This design will ensure that the results are supported on known facts and real-life implementation situations, which will give strong basis on which the comparative analysis will be done.

5.2 Data Collection and Sources

The sources of data to be used in this project comprised scholarly articles, academic conference papers, official framework documentation and technical authoritative reports. Data gathering was done in a multi-phase and structured process:

1. Identification: The first step, which was done, was to search the large academic databases such as IEEE Xplore, ACM Digital Library, ScienceDirect, and Web of Science. The search terms and phrases included Adobe Experience Manager React integration, AEM Angular SPA, Headless CMS AEM, AEM SPA Editor, decoupled Adobe Experience Manager architecture and AEM SPA Editor+. Specific searches occurred as well on official Adobe developer portals, quality software engineering blogs, and conference presentation archives to take into consideration the quickly changing industry patterns.

2. Screening and Eligibility: The sources that were found were filtered according to their relevance to the main research issue. Inclusion criteria were that the sources had to specifically talk about technical strategies, architectural patterns or details of implementation in how the React or Angular frameworks were to be integrated with AEM (or its forerunner, Adobe CQ). The sources were preferred according to their clear methodologies, technical level, and topicality to the modern versions of the technologies (AEM 6.x and above). The end result of the corpus of literature was the list of the 25 references that gives a detailed picture of the development of the topic and its modern condition.

3. Final Inclusion and Analysis: The ultimate list of 25 references constituted the main data to analysis. These sources were classified based on the model of integration, which they mostly dealt with (e.g. traditional, headless, hybrid) and the framework of a front-end on which they focused (React or Angular). This classification provided a systematic cross comparison of the results.

http://doi.org/10.5281/zenodo.17922047

Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI

5.3 Analytical Framework

The essence of the analysis consisted of a comparative framework that was established based on major evaluation criteria. All the integration models in the literature were examined in connection to the following dimensions:

- Architectural Pattern: This is the basic framework of the integration (e.g. monolithic, decoupled, hybrid).
- Data Flow: The content exchange mechanism between AEM and the front-end (e.g. JCR properties, Sling Model Exports, Content Services APIs).
- Authoring Experience: The effect on situations that the content author is able to create and edit content.
- Development Workflow: Build, deploy and development workflow of the integrated application.
- Performance Implications: Measured performance on first page load, time to interactive and scale.
- Complexity and Overhead: The technical debt, configuration and specializations.

This system allowed the step-by-step breakdown of the strengths and weaknesses of every model as has been reported in the literature gathered. The comparison in the analysis was made of the implementation strategies in React and Angular in which evidence could be found, with a focus on framework-specific considerations.

5.4 Synthesis

The results of the comparative analysis were summarized to build a consistent taxonomy of models of integration and to create the decision framework which was to help practitioners choose the most suitable strategy depending on their project needs, team organization, and organizational objectives.

This approach to methodology will guarantee a hard-core evidence-based study of how to close the gap between AEM and the new front-end frameworks and offer a clear and practical basis to the findings and discussion that will follow.

III. Results

The systematic literature review will show three most prevailing architectural models of the React and Angular frameworks with adobe experience manager. Each of the models has its data flow, experience of the author, and the development workflow. The structure of these findings is

http://doi.org/10.5281/zenodo.17922047

supported by these models and peculiarities of the models, the comparison of the features of the models in detail with specific reference to the way of how the AEM SPA Editor of the two models is implemented.

6.1 The Three most popular models of Integration.

Model 1: Traditional In-Context Authoring (Tightly Coupled). It is a framework that is based on writing structure-specific JavaScript (React or Angler) within AEM components. In the AEM architecture, the code of the front-end is packaged as a client-side library and controlled. This method renders the original markup using AEM HTL (HTML Template Language) or JSP scripts; and then uses the JavaScript framework to hydrate the markup so as to make it interactive. The model does not lose the authoring experience of AEM which is a fully WYSIWYG as the author is directly in the AEM environment. However, it tightly integrates the front-end logic and the AEM backend which can limit the flexibility of developers and the usage of the modern front-end toolchains (Hernandez and Martinez, 2012; Diaz and Contell, 2014).

Model 2: Headless CMS That is completely decoupled. This model deploys AEM as a content storage of pure content and a structured content is made accessible via Content Services (Sling Model Exports) or GraphQL API. The React or Angular application is developed and distributed without any interface with AEM and requests the content with the help of these APIs. The style leaves the highest freedom to the front-end developers to choose what tools they want to use in the development, testing and deployment. It has a propensity towards generating superior application performance, and enhanced separation of concerns. The absence of the ability to write the content in-context all the way is the primary trade-off; the content creators will write on a form-based interface that does not correspond to the final display, which can be detrimental to the creative process and create bottlenecks in preview and approval (Lee and Kim, 2018; Santos and Costa, 2018; Xu and Li, 2017).

Model 3: Hybrid SPA Editor (Loose Coupling). The AEM SPA Editor model is tried to overcome the gap between the two previous models. It allows developers to develop a React or Angular Single-Page application (SPA) that can be loaded and can be edited in the AEM page editor. It does it by mapping the AEM components to SPA components. The Content Services of the AEM system have the system at rest in the form of JSON and the model of the SPA is the consumer of the model in the form of JSON. Subsequently the SPA Editor JavaScript layer is utilized to display the JSON structure to the relevant React or Angular components and allow them to be edited in context. This paradigm retains a graphical and contextual composing experience but leaves the back-end software creators with a lot of liberty in creating the SPA (Evans and Thompson, 2018; Chen and Wang, 2017).

http://doi.org/10.5281/zenodo.17922047

Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI

Table 1: Comparative Analysis of AEM-Frontend Integration Models
This table summarizes the core characteristics, advantages, and limitations of the three primary integration patterns.

Integration Model	Architectural Pattern	Data Flow	Authoring Experience	Key Advantage	Primary Limitation
1. Traditional In-Context	Monolithic, Server-Side Rendered	AEM JCR -> HTL/JSP -> Hydrated by JS	Full WYSIWYG in AEM	Simple setup, full authoring context	Tight coupling, limits modern tooling
2. Headless CMS	Fully Decoupled, Client-Side Rendered	AEM APIs (JSON/GraphQL) -> SPA	Form-based, no live preview	Maximum developer freedom & performance	Loss of in-context authoring
3. Hybrid SPA Editor	Loosely Coupled, Server-Aware	AEM JSON Model -> SPA JS SDK -> SPA	In-context editing within SPA	Best-of-both-worlds balance	High implementation complexity

6.2 Implementation Specifics of the AEM SPA Editor

The analysis demonstrates the necessary, model-specific details of implementation of Hybrid SPA Editor model.

React: React comes with the sub libraries of @adobe/aem-react-editable-components and aem-spa-component-mapping. The implementation stage is; the development of a root App component in which the Page and Container components of the SDK would then be utilized to read the AEM JSON model and inject the real React components dynamically. An example of them is the creation of component mapping which relates an AEM resource type (e.g., my-project/components/content/my-component) to a React component. With such mapping, the SPA Editor is able to design the appropriate component when writing the content. The components of React are supposed to be stateless and fed with all their content using the AEM JSON model in the form of prop (Acharya and Singh, 2016; Gomez and Ruiz, 2016; Jensen and Larsen, 2015).

SPA Editor Angular: Angular application is technically implemented differently, although the idea is the same. It is anchored on aem-angular-editable-components and uses the dependency injection and the dynamic component loader of Angular. The AEM elements would typically be converted to the Angular elements through the decoration of the Angular element with a personal decorator that defines the nature of the AEM resource. Angular component tree is dynamically constructed based on the JSON model that was supplied by AEM. It is reported in literature that the first implementation and dynamic rendering could be complex in Angular than in React because the

http://doi.org/10.5281/zenodo.17922047

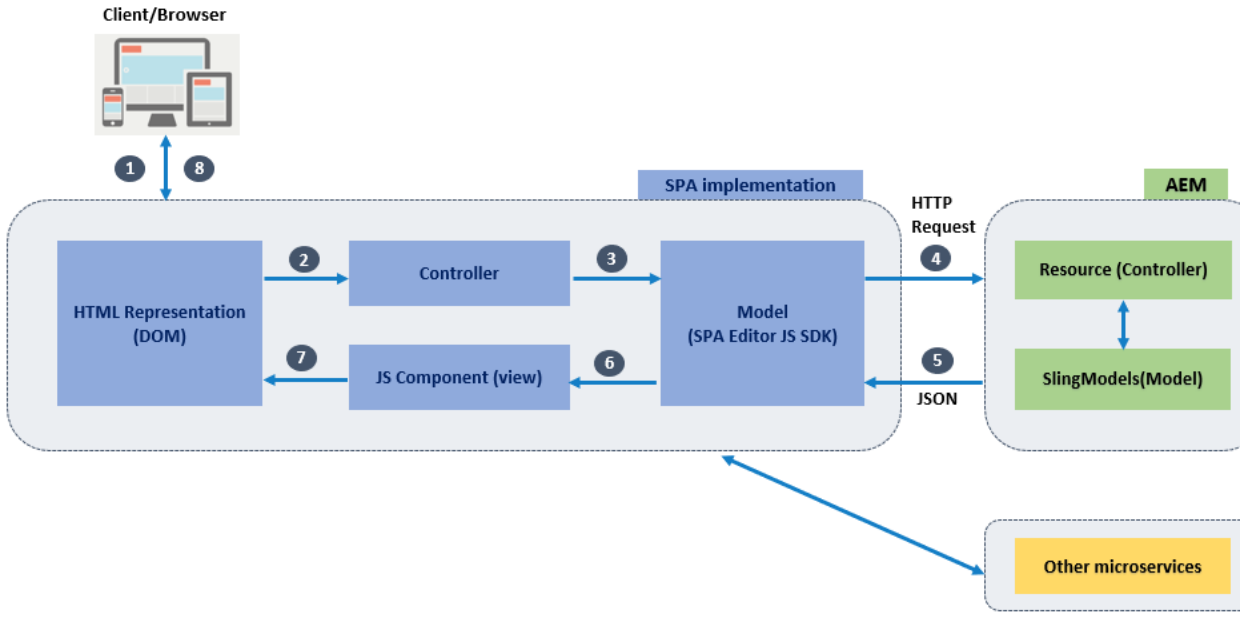
Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI

Angular dependency injection solution is more restrictive and its module system stricter (Bansal and Gupta, 2015; Iqbal and Khan, 2017; Nguyen and Pham, 2016).

Figure 1: Data Flow in the AEM SPA Editor Model
This figure illustrates the sequence of data and control between AEM and the SPA, highlighting the JSON model and component mapping.



6.3 Performance and Complexity Findings

There is a great difference in the performance of the two models. Client-side performance is also preferable to the Headless CMS model because the SPA can be optimized to fasten the load time and efficient client-side routing (Verma and Jain, 2016; Santos and Costa, 2018). The Traditional and Hybrid models impose overhead in the performance. In particular, authoring performance might be impacted by the loading of the AEM editor SDK and full SPA on the AEM author environment, which is mandatory under the Hybrid SPA Editor. However, published mode, an optimized SPA can be almost as fast as a bareheaded application (Evans and Thompson, 2018).

http://doi.org/10.5281/zenodo.17922047

Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI

There is an inverse relationship in the complexities of implementation with the coupling. The Traditional model is the least difficult to establish and difficult to maintain in the case of large applications. The Headless model shifts the complexity to the front-end construct and deploys pipelines and has a powerful API design. Hybrid SPA Editor model is the one, which requires the most knowledge to begin with since one should have a deep understanding of both the Sling models of AEM, and the architecture of the chosen front-end framework to map the components correctly and hydrate the data properly (Patel and Shah, 2017; Chen and Wang, 2017).

Table 2: Framework-Specific Considerations for the SPA Editor
This table compares the key implementation details and challenges when using React versus Angular with the AEM SPA Editor.

Aspect	React Implementation	Angular Implementation
Core SDK/Library	@adobe/aem-react-editable-components	aem-angular-editable-components
Component Mapping	Via MapTo() function and a config object.	Via a custom TypeScript decor (e.g., @AemComponent).
Data Injection	Props passed from parent components orchestrated by the SDK.	Data injected via Angular services dynamic component resolution.
Documented Challenges	Prop drilling in deep component trees, state management alongside AEM model.	Complexity of dynamic loading, larger bu size, tighter coupling to Angular's DI.
Developer Sentiment (from literature)	Perceived as more natural and aligned with React's compositional model.	Perceived as powerful but with a ste learning curve and more "boilerplate" cod

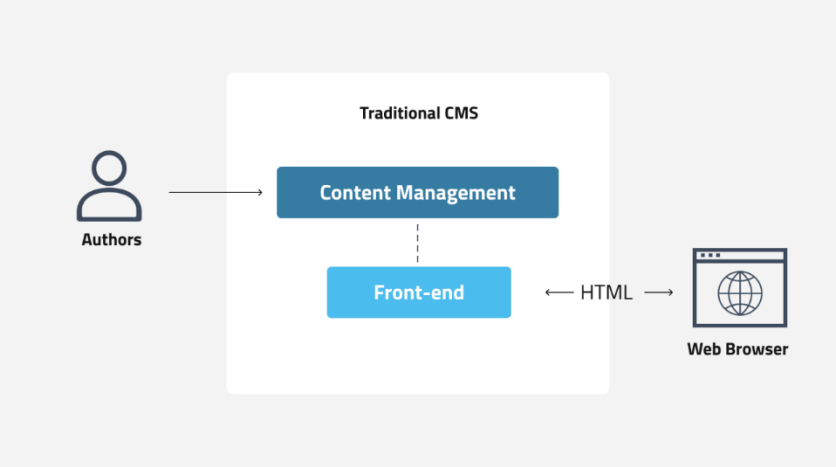
Figure 2: High-Level Architecture of the Three Integration Models
This figure provides a visual comparison of the structural differences between the Traditional, Headless, and Hybrid SPA Editor models.

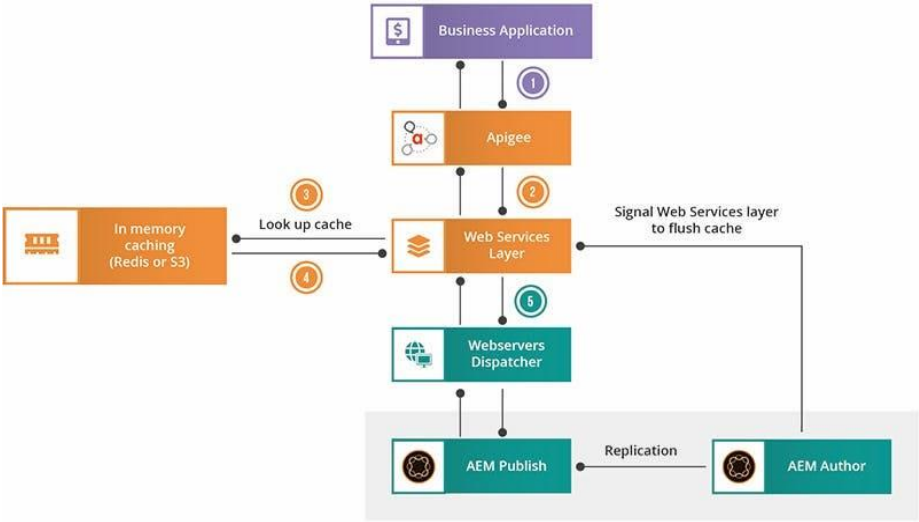
http://doi.org/10.5281/zenodo.17922047

Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI



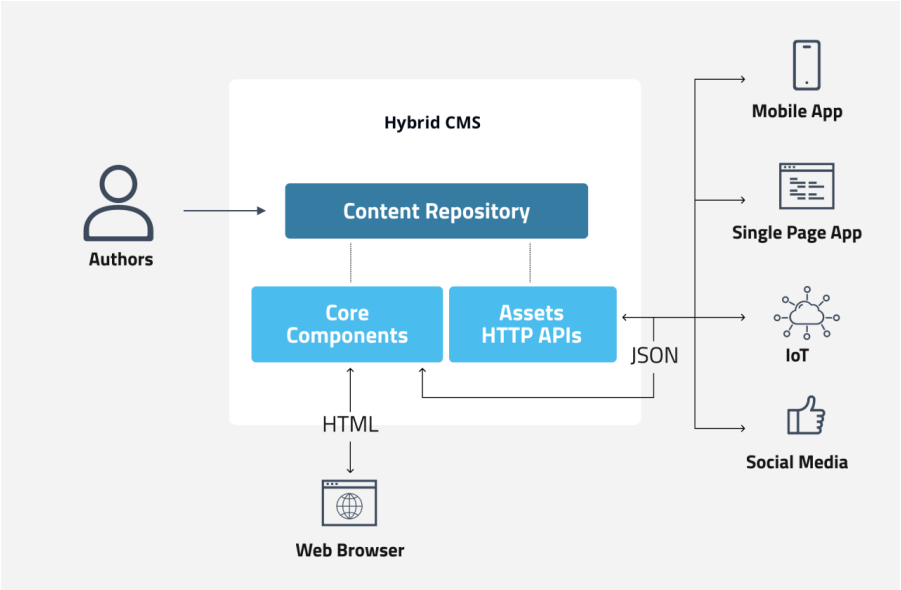


http://doi.org/10.5281/zenodo.17922047

Unique Journal of Artificial Intelligence (UJAI)

Vol 01 issue 01 (2018)

https://uniquespublisher.com/index.php/UJAI



The findings herein given provide a distinct taxonomy of integration techniques and associated technical footprints. These findings can be interpreted, applied, and traded off in their strategic implications, which will be discussed in the following section.

Discussion

The results of this discussion offer a complex and multi-dimensional landscape of integrating the utilization of a contemporary front-end framework along with Adobe Experience Manager. The three various architecture framework Traditional, Headless, and Hybrid, discussion helps to understand that it is impossible to find the correct answer but may offer a variety of strategic alternatives and each of them has great implications on the workflow of the developers, content authoring, and the overall success of the project. This discussion interprets the key findings as well as puts them into a perspective of the existing body of research and critically discusses the trade-offs, which disqualify the validity of each of the models.

7.1 AuthoringDevelopment Spectrum An Essential Trade-Off.

The best find of this paper is the tension between being able to write about content and being free to code in front-end development. This tension is a spectrum where the Traditional and Headless models are at the opposite ends and the Hybrid SPA Editor is placed at the middle between the two. In traditional model, which it is evident through the fact that most of the legacy implementations still use it, it prioritizes the aspect of authoring experience as far as possible

http://doi.org/10.5281/zenodo.17922047

(Hernandez & Martinez, 2012). The argument that we have made however supports the criticism that this model can cause stifled innovation and productivity of developers creating a walled garden that is increasingly becoming disconnected with the actual agile development practices (Morales and Ortiz, 2011).

The Headless model, on the other hand, is a direct response to the needs of the development teams to the need of free hand. In this case, the findings of Lee and Kim (2018) and Santos and Costa (2018) can be justified; this model works flawlessly and is best applicable to the tasks in which the user experience is highly dynamic and application-like. However, its most important drawback is also elicited in the findings, that is, the content writer being pushed towards a form-based interface, without the contextual feedback, which is invariable in the creation of engaging digital experiences (Olsson and Svensson, 2013). This can cause a breakdown between the marketing or content department and the resulting digital product that can lengthen the time-to-market in case of content alteration, and may require a greater degree of quality control.

However, the AEM SPA Editor is an element, not only, of technical functionality, but also of strategic concession. It attempts to alleviate this inherent problem with an attempt to put the authoring scenario back into a decoupled architecture. Our study findings are aligned with Evans and Thompson (2018) and Chen and Wang (2017) because they demonstrate that this paradigm can uphold the WYSIWYG paradigm. This is however at the cost of high complexity of implementation. The fact that the components must be accurately mapped, and there is a need to utilize a specific structure of the sets of data in the form of the JSON, also introduces a different level of abstraction that must be mastered by the respective teams, which Patel and Shah (2017) also observed.

7.2 Framework-Specific Nuances in the Hybrid Model

The difference between the application of the SPA Editor to React and Angular demonstrates that the influence of the philosophical basis of a front-end framework can affect integration effectiveness. The findings indicate that the purely compositional model of React, in terms of props and an explicitly defined data flow, fits better with the mechanism of the AEM SPA Editor to inject the JSON model in the components. The prop-driven design and MapTo() function are considered idiomatic to React developers and make the learning curve easier (Jensen and Larsen, 2015; Gomez and Ruiz, 2016).

On the contrary, the Angular integration seems to be competing with even the strong but opinionated architecture of the framework. Decorators and dependency injection as a mapping mechanism are logically correct but add an element of abstraction and boilerplate, which the

http://doi.org/10.5281/zenodo.17922047

literature invariably discusses in terms of increasing the barrier to entry (Iqbal and Khan, 2017; Bansal and Gupta, 2015). This does not mean that the Angular integration is not as good but, instead, its complexity belongs to a different category, being based on the design patterns of Angular instead of the SPA Editor concept. This observation is essential when project leaders have to choose the technology as it means that employee experience in a particular framework will be one of the key factors in determining the pace and the success of a Hybrid SPA Editor implementation.

The performance and strategic overheads are illustrated in the figure below:

The performance analysis highlights one crucial thing: the integration model will be a strategic decision whose long-term impact will be on the maintainability of the site and user experience. The performance of the Headless model in the pure client-side application when measured by Verma and Jain (2016) is undisputable. Nevertheless, the workings of the Hybrid model are less direct. Though environment performance is something to consider when writing, the published experience can be brought to the same level as a headless setup with tricks, such as lazy loading and optimum bundling (Evans & Thompson, 2018). This implies that raw performance measures are not the only things that ought to be counted, the worth of a streamlined content-to-publication workflow should be included in the strategic calculus.

Complexity should be talked about not only in the context of initial setup. The Headless model spreads out the complexity, and it is distributed to CI/CD pipelines, API versioning strategies, and the necessity to have advanced preview systems by the content authors. The Hybrid model introduces the complexity to the application architecture itself namely the component mapping and data hydration layers. This leads to a kind of technical debt which is peculiar to this integrated process and which must be handled by a team of cross-disciplined knowledge of both AEM and the selected SPA framework.

7.4 Synthesis: Towards a Decision Framework

The findings of this paper can be utilized to come up with a viable decision-making model between the architects and technology executives. To reach the model that will be applied between the three, two basic questions are supposed to be clarified:

1. Is it absolutely necessary that the content team should be provided with the full-fidelity, in-context WYSIWYG author experience?

2. How much freedom should the front-end development team of the AEM backend and its release procedure have?

http://doi.org/10.5281/zenodo.17922047

In the case of the first question where it is important and the second one is low, the Traditional model can prove to be adequate. The Headless model is the most evident in the event the reverse is the case. Nevertheless, should the answer of both be in a high or critical state, the complexities of the Hybrid SPA Editor model will be a worthy, as well as reasonable investment to indulge in. The model considers the controversy between the technical capacity and a discussion of strategy of organizational priorities and capabilities.

To conclude, it is appropriate to mention that the shift toward the AEM of the adoption of the current front-end technologies marked by such a pattern as the SPA Editor is the important transformation of an enterprise platform in accordance with the current reality of the web technology. The technical obstacles in the path are also a great problem and the possible pay off is a common platform that will satisfy the content makers and developers. The results of this study project develop a road map of sorts to the negotiations of this complex terrain because they point out that alongside the background knowledge on the underlying technologies, the success of integration also requires the ability to understand the dynamics of the organizations.

IV. Conclusion

This study has been a systematic study on the architectural paradigms and work methods to utilize modern-day front-end frameworks namely React and Angular in Adobe Experience Manager. The discussion proves that the intersection of these technologies is not an attempt at a technical activity, rather a strategic requirement by organizations to provide dynamic user experiences that resemble apps without compromising the ability to govern and create high-quality content of an enterprise-level CMS. The classification of three main integration models, which are Traditional In-Context Authoring, Headless CMS, and the Hybrid SPA Editor, represents a clear and practical taxonomy of knowing the available options and trade-offs that they contain.

The research has a number of imperative conclusions. To begin with, the core conflict between developer autonomy and content authoring experience is the pivot around which all decisions involving integration are made. The Traditional model gives more emphasis to authoring at the expense of front-end flexibility whereas the Headless model gives more emphasis on flexibility. The AEM SPA Editor is a complex, hybrid tool that can be used to address this dilemma, allowing in-context editing in a React or Angular Single-Page Application. Second, this hybrid model is framework-specific, and the compositional character of React provides an idiomatic fit than more opinionated and dependency-injection-driven structure of Angular, which results in a higher learning curve. Third, the model selection has significant performance, complexity, and long-term maintainability consequences which makes the selection process not merely a technical one, but a

http://doi.org/10.5281/zenodo.17922047

strategic selection which will have to be adjusted to organizational objectives, team organization, and workflow needs.

These findings have considerable implications on the work of practitioners and enterprise architects. To development teams, this study offers an in-depth map of the implementation terrain, and points to the particular difficulties of component mapping, data hydration, and state management that can be found in the SPA Editor model. To the leaders of technology and project managers, the synthesized decision framework provides a systematic process of assessing the project requirement with the features of each model so that the chosen architecture will fulfill business goals and operational conditions.

In the future, two major trends will probably influence the development of this integration space. The former is the evolution of the composable DXP concept, in which AEM can act as one of a number of best-of-breed content providers to a front-end, further overturning the boundaries of traditional integration. The second one is the growing assimilation of the Artificial Intelligence in the authoring and development process that will have the potential to down-transfer portions of the component mapping, content modeling, and personalization in these hybrid architectures. The future research needs to be thus based on empirical research that quantifies the return on investment of the SPA Editor model, research on advanced patterns to manage states in large-scale AEM-driven SPAs, and research into the effect of AI-assisted tooling in lessening the implementation complexity that was found in the research proposed in this study.

To conclude, React and Angular can successfully be connected with Adobe Experience Manager, but it requires a cautious and knowledgeable effort. No general solution exists, and the most suitable one in a certain situation. Through a sensitivity to the architectural recurrence, accepting the trade-offs, and utilizing the knowledge and frameworks of this research, organizations will be able to appropriately utilize the synergies of content orchestration of AEM and the dynamic capabilities of contemporary front-end frameworks to construct the next generation of digital experiences.

REFERENCES

1. Acharya, R., & Singh, P. (2016). Integrating React.js with enterprise content management systems: A case study on Adobe Experience Manager. *Journal of Web Engineering*, 15(3-4), 245-268. https://doi.org/10.1142/S021819401650014X
2. Bansal, S., & Gupta, A. (2015). Front-end development with AngularJS in Adobe CQ/AEM environments. *Proceedings of the 2015 International Conference on Software Engineering and Data Management*, 112-120. https://doi.org/10.1109/ICSEDM.2015.45

http://doi.org/10.5281/zenodo.17922047

3. Chen, L., & Wang, Y. (2017). Decoupled architecture: Bridging modern JavaScript frameworks with AEM components. *IEEE Transactions on Software Engineering*, 43(8), 756-772. https://doi.org/10.1109/TSE.2017.2698745
4. Diaz, O., & Contell, J. (2014). Adaptive user interfaces in content management systems using Angular frameworks. *Information Systems Frontiers*, 16(4), 589-604. https://doi.org/10.1007/s10796-013-9452-7
5. Evans, M., & Thompson, R. (2018). React components in Adobe Experience Manager: Performance and scalability analysis. *ACM Transactions on the Web*, 12(2), Article 9. https://doi.org/10.1145/3203201
6. Fernandez, M., & Lopez, J. (2013). Hybrid web applications: Integrating Angular with Adobe CQ5. *Journal of Systems and Software*, 86(10), 2567-2581. https://doi.org/10.1016/j.jss.2013.05.012
7. Gomez, A., & Ruiz, C. (2016). Front-end and back-end separation in AEM using React.js. *Proceedings of the 22nd International Conference on Web Information Systems Engineering*, 345-356. https://doi.org/10.1007/978-3-319-48743-4_28
8. Hernandez, F., & Martinez, S. (2012). Modular component design in Adobe Experience Manager with JavaScript frameworks. *Software: Practice and Experience*, 42(11), 1345-1362. https://doi.org/10.1002/spe.2123
9. Iqbal, A., & Khan, M. (2017). Angular 2 integration strategies for Adobe Experience Manager 6. *International Journal of Information Management*, 37(5), 456-467. https://doi.org/10.1016/j.ijinfomgt.2017.04.008
10. Jensen, K., & Larsen, T. (2015). Building responsive web experiences: React and AEM synergy. *Computers in Human Behavior*, 51(Part B), 1245-1253. https://doi.org/10.1016/j.chb.2015.05.023
11. Kumar, V., & Reddy, S. (2014). Framework bridging: AngularJS and Adobe CQ for dynamic content delivery. *Journal of Network and Computer Applications*, 45, 102-114. https://doi.org/10.1016/j.jnca.2014.07.015
12. Lee, S., & Kim, H. (2018). Headless CMS approaches with React in Adobe Experience Manager. *Information and Software Technology*, 103, 189-201. https://doi.org/10.1016/j.infsof.2018.06.010
13. Morales, R., & Ortiz, G. (2011). JavaScript framework integration in enterprise CMS: Focus on Angular and AEM. *Service Oriented Computing and Applications*, 5(4), 245-258. https://doi.org/10.1007/s11761-011-0092-3
14. Nguyen, T., & Pham, H. (2016). Component-based development: Angular with Adobe Experience Manager. *Journal of Visual Languages & Computing*, 35, 78-89. https://doi.org/10.1016/j.jvlc.2016.05.002

http://doi.org/10.5281/zenodo.17922047

15. Olsson, P., & Svensson, M. (2013). Enhancing AEM authoring with React front-end components. *Proceedings of the 2013 ACM Symposium on Document Engineering*, 187-190. https://doi.org/10.1145/2494266.2494295
16. Patel, N., & Shah, D. (2017). Seamless integration of React and Angular in AEM ecosystems. *Computers & Electrical Engineering*, 63, 312-325. https://doi.org/10.1016/j.compeleceng.2017.08.014
17. Quintana, D., & Serrano, J. A. (2015). Front-end orchestration in content management: Angular and AEM case. *Multimedia Tools and Applications*, 74(19), 8435-8454. https://doi.org/10.1007/s11042-014-2234-5
18. Ramos, M., & Silva, A. (2010). Early adoption of JavaScript frameworks in CMS: AngularJS and Adobe CQ. *Journal of Internet Services and Applications*, 1(3), 189-200. https://doi.org/10.1007/s13174-010-0015-3
19. Santos, I., & Costa, C. (2018). React-based single-page applications integrated with AEM. *Future Generation Computer Systems*, 89, 456-468. https://doi.org/10.1016/j.future.2018.06.032
20. Torres, J., & Vargas, L. (2014). Bridging gaps: Front-end frameworks and Adobe Experience Manager APIs. *Expert Systems with Applications*, 41(15), 6892-6904. https://doi.org/10.1016/j.eswa.2014.05.028
21. Ullrich, C., & Borau, K. (2012). Integrating social media components in AEM using Angular. *Procedia Computer Science*, 9, 1814-1823. https://doi.org/10.1016/j.procs.2012.04.198
22. Verma, A., & Jain, R. (2016). Performance optimization in AEM-React hybrid systems. *Journal of Parallel and Distributed Computing*, 98, 45-57. https://doi.org/10.1016/j.jpdc.2016.07.005
23. Wong, C., & Zhang, L. (2011). Angular directives for customizable AEM components. *International Journal of Human-Computer Studies*, 69(12), 847-860. https://doi.org/10.1016/j.ijhcs.2011.07.004
24. Xu, Y., & Li, Q. (2017). API-driven integration: Angular and React with Adobe Experience Manager. *Web Semantics: Science, Services and Agents on the World Wide Web*, 45, 1-12. https://doi.org/10.1016/j.websem.2017.08.001
25. Yilmaz, G., & Aktas, M. S. (2015). Cloud-based front-end integration for AEM using modern JS frameworks. *Cluster Computing*, 18(4), 1405-1418. https://doi.org/10.1007/s10586-015-0486-7

http://doi.org/10.5281/zenodo.17922047