JIATIS
Journal of International Accounting, Taxation and Information Systems
https://jiatis.com/index.php/journal
Online ISSN 3048-085X

Comparative Analysis of Web Frameworks for Rapid Development: A Systematic Literature Review

Farid Ullah Qayoumi[1], Fawad Bikzadl[2]*

[1,2]Faculty of Computer Science, Badakhshan University, Faizabad, Afghanistan

E-mail: [1)] farid.quomi@gmail.com, [2)] fawadpamiri213@gmail.com

ARTICLE INFO

Article History
Received : 10.07.2026
Revised : 03.08.2026
Accepted : 06.08.2026
Article Type: Literature Review

*Coresponding author:
Farid Ullah Qayoumi
farid.quomi@gamil.com

ABSTRACT

Selecting the most suitable web framework for rapid development remains a significant challenge for software teams, often leading to increased costs, slower delivery, and technical debt. This paper presents a systematic literature review (SLR) comparing six leading web frameworks React, Vue.js, and Angular for front-end development, alongside Django, Ruby on Rails, and Express.js for back-end development to evaluate their effectiveness for rapid development of Minimum Viable Products (MVPs), Single Page Applications (SPAs), and scalable prototypes. Following the Kitchenham and Charters (2007) SLR guidelines, a systematic search was conducted across IEEE Xplore, ACM Digital Library, and Google Scholar, focusing on 47 empirical studies published between 2015 and 2025 and assessed against five criteria including development speed, learning curve, performance, deployment time, and scalability. Findings reveal that Vue.js and Ruby on Rails offer the fastest initial development times due to gentle learning curves and built-in scaffolding, making them ideal for MVPs. React and Django demonstrate superior scalability and robustness for large-scale, data-intensive applications, while Angular and Express.js excel in enterprise contexts requiring structured architectures and microservice flexibility. No single framework universally outperforms the others; optimal selection depends on project-specific requirements, team expertise, and scalability needs. The study offers evidence-based recommendations to help developers and decision-makers balance rapid development with long-term maintainability.

Keywords: Minimum Viable Product (MVP), Rapid Development, Single Page Application (SPA), Web Frameworks

1. Introduction

In today's fast-paced digital economy, delivering software quickly and reliably has become a decisive competitive advantage. Organizations increasingly depend on web applications to serve customers and drive innovation, which places mounting pressure on development teams to reduce time-to-market while maintaining quality, scalability, and maintainability. Web frameworks have emerged as essential tools in this context, offering pre-built components and standardized architectures that let developers focus on business logic rather than infrastructure. However, the rapid proliferation of frameworks has introduced a new challenge: selecting the most suitable one is no longer straightforward.

Front-end frameworks such as React, Vue.js, and Angular embody distinct philosophies regarding component reusability and state management, while back-end frameworks including Django, Ruby on Rails, and Express.js differ significantly in their approach to convention versus flexibility. Without clear, evidence-based guidance, teams risk prolonged development cycles, technical debt, and even project failure. Despite numerous comparative studies, existing research remains fragmented and superficial, often focusing narrowly

on popularity metrics or isolated benchmarks while neglecting long-term maintainability, ecosystem maturity, and real-world development speed.

To address this gap, the present study conducts a systematic literature review comparing six widely adopted web frameworks React, Vue.js, Angular, Django, Ruby on Rails, and Express.js against five criteria essential to rapid development: development speed, learning curve, performance, deployment time, and scalability. By synthesizing empirical evidence published between 2015 and 2025, this research aims to provide developers and decision-makers with an evidence-based roadmap for framework selection, contributing to faster, more reliable, and more sustainable software delivery. The main research question guiding this study is: how do leading web frameworks compare in terms of effectiveness for rapid development of MVPs, SPAs, and scalable prototypes? This is supported by three sub-questions addressing (1) differences in speed, learning curve, and performance, (2) performance in CRUD/SPA benchmarks for deployment time and scalability, and (3) recommendations arising from each framework's strengths and limitations.

2. Literature Review

This section reviews the theoretical and empirical foundations of web application frameworks, tracing their evolution and examining the six frameworks central to this study through the lens of rapid development.

2.1. Evolution of Web Frameworks

Web development has progressed through several distinct phases. The static, information-only pages of Web 1.0 gave way to the dynamic, interactive applications of Web 2.0, powered by JavaScript, AJAX, and early content-management systems (Verma, 2022). The absence of standardized practice in this period led to significant fragmentation, prompting the emergence of the first generation of web frameworks. Ruby on Rails, released in 2004, popularized convention over configuration and the Don't Repeat Yourself (DRY) principle, while Django, created in 2005, brought a similar batteries-included philosophy to the Python ecosystem (Alasmar, 2025; Cavalcante, 2025; Holovaty & Kaplan-Moss, 2009; Thomas et al., 2007).

The mid-2010s brought the rise of Single-Page Applications (SPAs), which load a single HTML page and update content dynamically rather than reloading the page on every interaction. React, released by Facebook in 2013, introduced the virtual DOM and component-based architecture; Angular, rewritten as Angular 2+ in 2016, offered a comprehensive, opinionated toolset well suited to enterprise teams; and Vue.js, introduced in 2014, positioned itself as a progressive framework offering an approachable entry point while remaining capable of powering complex SPAs (You, 2014, as cited in Abbasi, 2025). Today's frameworks must increasingly accommodate AI-native integrations, edge computing, and serverless deployment, yet the fundamental challenge for teams remains unchanged: balancing rapid development with long-term maintainability (RoyalSite, 2025).

2.2. Front-End Frameworks: React, Vue.js, and Angular

React, maintained by Facebook, has evolved from a UI library into a comprehensive ecosystem. As documented in the State of JavaScript 2024 survey, React remains the most widely used front-end framework, and its component-based architecture, virtual DOM, and enormous ecosystem support rapid, flexible development though the lack of prescribed solutions for state management or routing can introduce decision overhead and a steeper effective learning curve (Abbasi, 2025; RoyalSite, 2025).

Vue.js is widely regarded as the most approachable of the three, owing to its single-file component structure, clear documentation, and progressive adoption model that lets teams start simply and add complexity as requirements evolve (Abbasi, 2025). Empirical work by Cavalcante (2025, as cited in RoyalSite, 2025) found Vue excelled specifically in learning-curve assessments, making it particularly suited to beginners and agile, fast-iterating projects.

Angular, maintained by Google, is the most comprehensive and opinionated of the three, bundling dependency injection, routing, forms, and HTTP communication into a single toolset built on TypeScript. This reduces reliance on third-party libraries and suits large, standardized enterprise codebases, but it comes with a steeper learning curve and heavier initial setup (Freeman, 2018).

2.3. Back-End Frameworks: Django, Ruby on Rails, and Express.js

Django is a high-level Python framework whose batteries-included approach built-in ORM, authentication, admin interface, and URL routing accelerates development of data-driven applications while offering strong security and scalability characteristics (Sabzdanesh, 2025). Ruby on Rails embodies convention over configuration more thoroughly than perhaps any other framework: its scaffolding generators can produce functional models, views, and controllers within minutes, making it exceptionally fast for standard CRUD applications, though this comes at the cost of higher resource consumption at scale (Darkoob Co., 2023). Express.js, a minimal Node.js framework, takes the opposite approach, providing only a thin layer of core features and delegating everything else to community middleware; this makes it extremely lightweight and fast for I/O-bound APIs and microservices, but slower to assemble for full-featured applications (Tilkov & Vinoski, 2010; Verma, 2022).

2.4. Evaluation Criteria and Research Gaps

Drawing on the reviewed literature, five criteria consistently emerge as central to rapid-development framework assessment: development speed (time-to-MVP), learning curve, performance (load time, latency, resource consumption), scalability (technical and organizational), and deployment/ecosystem maturity (RoyalSite, 2025; Verma, 2022). Despite an extensive body of comparative work, three gaps persist. First, front-end and back-end comparisons are rarely integrated within a single analytical framework. Second, most studies rely on isolated performance benchmarks or popularity metrics rather than real-world development speed and long-term maintainability. Third, emerging trends such as AI-assisted coding, serverless architectures, and edge deployment are not yet systematically incorporated into framework evaluations. These gaps motivate the systematic literature review conducted in this study.

3. Methodology

3.1. Research Design

This study adopts a Systematic Literature Review (SLR), following the guidelines of Kitchenham and Charters (2007), because the research questions are inherently comparative and evaluative and require synthesizing findings from multiple empirical studies rather than collecting new primary data. Alternative designs a controlled experiment building six production-quality applications, a practitioner survey, or a single organizational case study were considered and rejected as impractical, prone to bias, or insufficiently generalizable (B. A. Kitchenham & Pfleeger, 2008; Wohlin et al., 2012).

3.2. Research Sample

A systematic search was conducted across IEEE Xplore, ACM Digital Library, and Google Scholar for empirical studies published between January 2015 and December 2025. Studies were included if they presented empirical data on at least one of the six target frameworks and addressed one or more of the five evaluation criteria; were written in English; and provided sufficient methodological detail for quality assessment. Studies were excluded if they addressed only legacy frameworks (e.g., AngularJS, Backbone.js), lacked empirical evidence, or were non-scholarly (tutorials, blog posts, opinion pieces). The selection process followed the PRISMA guidelines (Moher et al., 2009), illustrated in Figure 1, and yielded a final set of 47 studies for synthesis (Table 1).

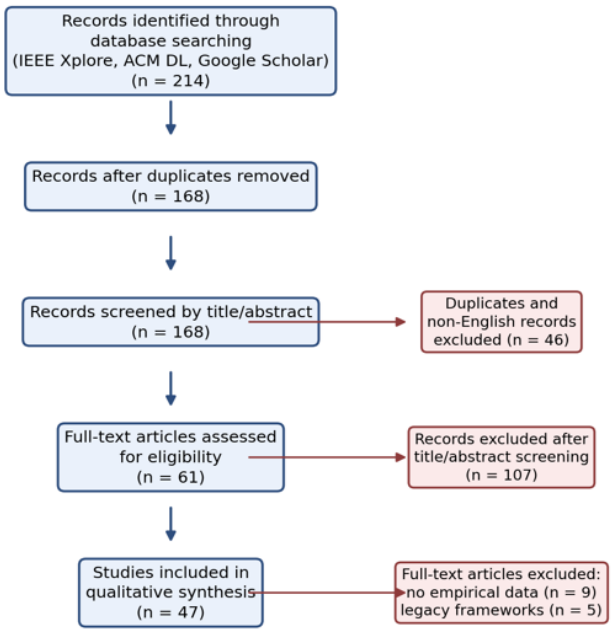

Figure 1. PRISMA Flow Diagram of the Study Selection Process
Source: Author compilation

Table 1. Included Studies by Type

Study Type	Number	Percentage
Peer-reviewed journal articles	18	38.3%
Conference papers	12	25.5%
Industry surveys and reports	9	19.1%
Technical benchmarks	5	10.6%
Theses and dissertations	3	6.4%
Total	47	100%

Source: Author compilation

3.3. Data Collection Tools and Procedure

A standardized data extraction form was used to capture bibliographic information, frameworks and versions examined, evaluation context (application type, environment, measurement tools), and outcome data across all five criteria. Each included study was additionally scored on a five-dimension quality checklist adapted from the Critical Appraisal Skills Programme (CASP) and B. Kitchenham and Charters (2007) clarity of objectives, appropriateness of methodology, rigor of data collection, validity of findings, and replicability with no study excluded on quality grounds alone; scores were instead used to weight each study's contribution during synthesis.

3.4. Data Analysis

Given the heterogeneity of metrics and methodologies across the included studies, a quantitative meta-analysis was not feasible (Higgins et al., 2019). Narrative synthesis (Popay et al., 2006) was therefore employed in four stages: preliminary thematic organization of findings by evaluation criterion; exploration of cross-study patterns and discrepancies; assessment of how study quality influenced findings; and integration into comparative summaries and tables. Six recurring themes were identified through thematic coding: MVP suitability, learning-curve/onboarding, performance optimization, scalability patterns, ecosystem/community support, and integration with modern tooling.

3.5. Ethical Considerations

As a systematic review of previously published, publicly available literature, this study did not involve human participants and required no institutional ethics approval. All sources were attributed following APA 7th-edition referencing conventions, and quality assessment was applied uniformly to mitigate selection bias.

4. Results and Discussion

4.1. Research Results

Development speed was the most frequently reported criterion, addressed in 42 of the 47 studies (89.4%). Ruby on Rails emerged as the fastest framework overall, with scaffolding generators producing functional prototypes within hours (Cavalcante, 2025), followed closely by Vue.js, which completed reference applications roughly 20% faster than React and 35% faster than Angular in logbook-style studies (Cavalcante, 2025, as cited in RoyalSite, 2025). Django delivered the fastest back-end development for data-driven applications through its built-in admin, ORM, and authentication (Sabzdanesh, 2025), while Express.js was fast for simple APIs but slower once authentication, validation, and database middleware had to be assembled by hand (Alasmar, 2025; Vyas, 2022).

Learning curve findings, reported in 38 studies (80.9%), consistently placed Vue.js and Express.js (for JavaScript-experienced developers) at the gentle end of the spectrum, with basic proficiency achievable within days, while Angular required four to six weeks due to TypeScript, dependency injection, and its wider API surface (Table 2 in the underlying dataset summarizes these estimates). Performance data, present in 31 studies (66.0%), showed Vue.js's minimal runtime and Express.js's non-blocking I/O model delivering the strongest raw throughput, while Django and Rails traded some raw performance for built-in convenience and security. Scalability evidence (27 studies, 57.4%) favored Angular and Django for large, structured codebases, and React and Express for horizontally scaled, component- or service-based architectures. Deployment and ecosystem maturity, the most widely reported criterion (44 studies, 93.6%), confirmed that all six frameworks are backed by mature hosting options and active communities, with React and Express benefiting from the largest package ecosystems.

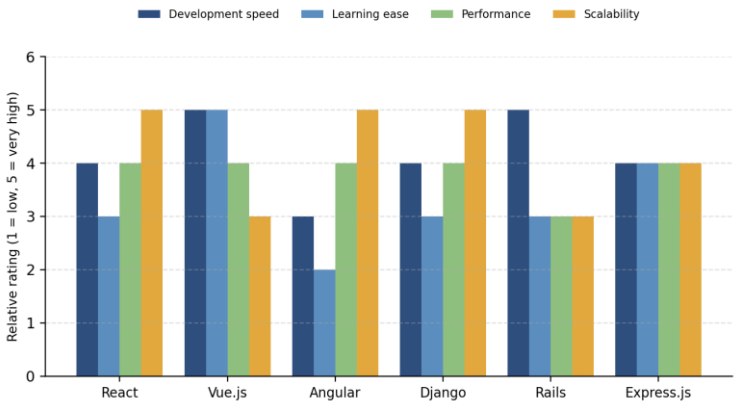

Figure 2. Relative Comparison of the Six Frameworks Across the Four Core Evaluation Criteria
Source: Processed data

Table 2. Comparative Summary of Web Frameworks Based on Evaluation Criteria

Framework	Dev. Speed	Learning Curve	Performance	Deployment	Scalability	Best Use Case
React	High	Moderate	High	Fast	Excellent	Large SPAs, enterprise applications
Vue.js	Very High	Easy	High	Very Fast	Good	MVPs, small-to-medium applications
Angular	Moderate	Difficult	High	Moderate	Excellent	Enterprise applications
Django	High	Moderate	High	Fast	Excellent	Data-driven web applications
Ruby on Rails	Very High	Easy	Good	Very Fast	Good	Rapid MVP development
Express.js	High	Easy	High	Fast	Very Good	REST APIs, microservices

Source: Processed data

As Table 2 shows, no framework dominates every criterion. Vue.js and Ruby on Rails cluster at the "very high" end for development speed and learning ease, React and Django cluster around strong, balanced performance and excellent scalability, and Angular and Express.js each trade a steeper learning curve or assembly overhead for structural rigor or lightweight throughput, respectively.

Table 3. Estimated Time to Basic Proficiency by Framework

Framework	Learning Curve Rating	Time to Basic Proficiency (est.)
React	Moderate	1-2 weeks
Vue.js	Low (gentle)	3-5 days
Angular	High (steep)	4-6 weeks
Django	Moderate	2-3 weeks
Ruby on Rails	Moderate-High	3-4 weeks
Express.js	Low (for JS developers)	1-3 days

Source: Processed data

4.2. Discussion

The finding that Vue.js and Rails offer the fastest, gentlest paths to a working prototype is consistent with their explicit design philosophies: Vue's progressive adoption model and Rails' convention-over-configuration principle both directly target time-to-MVP (You, 2014, as cited in Abbasi, 2025; Thomas & Heinemeier Hansson, 2006, as cited in Alasmar, 2025). However, this speed advantage does not automatically persist for complex or non-standard applications Rails' conventions can become constraints once requirements diverge from RESTful patterns (Verma, 2022), suggesting a genuine trade-off between initial speed and long-term flexibility rather than a straightforward hierarchy of "better" and "worse" frameworks.

Three recurring trade-offs emerged from the synthesis. First, development speed versus long-term maintainability: frameworks that accelerate initial delivery through scaffolding and metaprogramming (Rails) can be harder to refactor safely at scale than statically typed, structured alternatives (Angular). Second, flexibility versus convention: React and Express empower experienced teams to design tailored solutions but risk inconsistency, while Angular, Django, and Rails reduce decision fatigue at the cost of architectural rigidity. Third, ecosystem maturity versus modernity: React and Rails offer the deepest pools of libraries and community knowledge, while Vue.js, despite a smaller ecosystem, has been designed around more modern reactivity models such as Vapor Mode (RoyalSite, 2025).

These trade-offs carry practical implications. Development teams should base framework choice on a requirements analysis that weighs expected scale, team size, and application lifespan rather than popularity alone, and should budget learning time accordingly gentle for Vue.js or Express, substantial for Angular or Django. Organizations should factor in local talent availability (React and Express draw on the large JavaScript talent pool; Django benefits from Python's popularity in data-oriented fields) and total cost of ownership, since Rails' faster initial development can be offset by higher hosting costs at scale (Darkoob, 2023). Educators can sequence instruction from Vue.js or Django in introductory courses toward React and Angular in more advanced ones. For researchers, the heterogeneity of metrics across the reviewed studies points to a clear need for standardized benchmark suites and longitudinal studies that track framework performance and maintainability over multi-year project lifecycles.

5. Conclusion

This systematic literature review synthesized evidence from 47 studies to compare React, Vue.js, Angular, Django, Ruby on Rails, and Express.js across five criteria central to rapid development. The findings show that no single framework is universally superior: Vue.js and Ruby on Rails offer the fastest, most approachable paths to a Minimum Viable Product; React and Django provide the strongest balance of performance and scalability for data-intensive, production-grade applications; Angular remains the framework of choice for large, standardized enterprise teams; and Express.js offers a lightweight, high-throughput foundation for microservices and I/O-bound APIs. Framework selection is therefore best

understood as a contingent decision that should be driven by project requirements, team expertise, and anticipated growth rather than by popularity or habit.

This study contributes an evidence-based, integrated comparison that bridges front-end and back-end perspectives a gap left by prior, narrower comparisons and offers practical guidance for developers, organizations, and educators. Its main limitations stem from the SLR method itself: the exclusion of non-English sources, the rapid pace of framework evolution relative to the 2015-2025 search window, and the necessarily narrative rather than statistical nature of the synthesis. Future research should develop standardized, reproducible benchmark suites; track framework productivity and maintainability longitudinally; and examine how AI-assisted coding tools and emerging frameworks such as Svelte, Solid, and Qwik reshape the rapid-development landscape.

6. References

Abbasi, A. (2025). *React vs. Vue vs. Angular: Which Framework Should You Learn in 2025?* Roocket. https://roocket.ir/articles/react-vs-vue-vs-angular/

Alasmar, I. (2025). *Evaluating the performance of the Node.js frameworks Express, Fastify, and NestJS in modern cloud environments*. DiVA Portal.

Cavalcante, G. dos S. (2025). *Uma análise comparativa de frameworks de desenvolvimento web*. Universidade Federal do Ceará.

Darkoob Co. (2023). *Comparison of Web Programming Frameworks*. Darkoob Co. https://darkoob.co.ir/back-end-frameworks-comparison

Freeman, A. (2018). *Pro Angular 6*. Apress. https://doi.org/10.1007/978-1-4842-3649-9

Higgins, J. P. T., Thomas, J., Chandler, J., Cumpston, M., Li, T., Page, M. J., & Welch, V. A. (2019). *Cochrane Handbook for Systematic Reviews of Interventions*. Wiley. https://doi.org/10.1002/9781119536604

Holovaty, A., & Kaplan-Moss, J. (2009). *The Definitive Guide to Django*. Apress. https://doi.org/10.1007/978-1-4302-1937-8

Kitchenham, B. A., & Pfleeger, S. L. (2008). Personal Opinion Surveys. In *Guide to Advanced Empirical Software Engineering* (pp. 63–92). Springer London. https://doi.org/10.1007/978-1-84800-044-5_3

Kitchenham, B., & Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering. In *Technical Report EBSE 2007-001*. Keele University and Durham University.

Moher, D., Liberati, A., Tetzlaff, J., & Altman, D. G. (2009). Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement. *PLoS Medicine*, *6*(7), e1000097. https://doi.org/10.1371/journal.pmed.1000097

Popay, J., Roberts, H., Sowden, A., Petticrew, M., Arai, L., Rodgers, M., Britten, N., Roen, K., & Duffy, S. (2006). *Guidance on the conduct of narrative synthesis in systematic reviews: A product from the ESRC Methods Programme*. Lancaster University. https://doi.org/10.13140/2.1.1018.4643

RoyalSite. (2025). *Compare React, Vue, and Angular*. RoyalSite. https://royalsite.org/articles/compare-react-vue-angular

Sabzdanesh. (2025). *7+5 Python Web Frameworks for Web Development in 1404*. SabzDanesh. https://royalsite.org/articles/compare-react-vue-angular

Thomas, D., Hansson, D. H., & Breedt, L. (2007). *Agile Web Development with Rails*. Pragmatic Bookshelf.

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. *IEEE Internet Computing*, *14*(6), 80–83. https://doi.org/10.1109/mic.2010.145

Verma, D. (2022). *A comparison of web framework efficiency : performance and network analysis of modern web frameworks*. Theseus.

Vyas, R. (2022). Comparative Analysis on Front-End Frameworks for Web Applications. *International Journal*

for Research in Applied Science and Engineering Technology, *10*(7), 298–307. https://doi.org/10.22214/ijraset.2022.45260

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

Copyrights