World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(RESEARCH ARTICLE)

Check for updates

Comprehensive Performance and Scalability Assessment of Front-End Frameworks: React, Angular, and Vue.js

Mikita Piastou *

Full-Stack Developer at Emplifi Inc, University of California, Berkeley, Extension.

World Journal of Advanced Engineering Technology and Sciences, 2023, 09(02), 366–376

Publication history: Received on 10 May 2023; revised on 13 August 2023; accepted on 16 August 2023

Article DOI: https://doi.org/10.30574/wjaets.2023.9.2.0153

Abstract

This paper discusses performance and scalability opportunities of the three major modern front-end frameworks, namely React, Angular, and Vue.js. In this respect, the research tries to point out the strengths and limitations of the compared frameworks in handling large-scale and complex web applications by evaluating each against different load conditions or scalability scenarios. It does this by running a set of benchmark applications that test performance indicators such as load times, speed of rendering, memory usage, and CPU usage. Additionally, it investigates scalability through load testing-emulation of increased user interactions and complexity testing, observing how each of these frameworks handles increasing application complexity. Overall, the results obtained were good, especially those for React, where the use of its virtual DOM was exploited, though some optimization may be necessary when handling more complex state changes. Angular has great performance, though it can show increased load times and higher memory usage due to the richness of features. Vue.js has competitive performance, with lower memory usage and faster render times, balancing simplicity with scalability quite well. The study concludes by giving insight on how to select the correct framework based on an application's particular needs and thought on scalability for guidance on choosing the right tool for the development job at hand.

Keywords: React; Angular; Vue.Js; Front-End Frameworks; Performance Metrics; Load Testing; Rendering Speed; Memory Usage; CPU Usage; Web Application Performance; Scalability; Resource Optimization

1. Introduction

The front-end framework has become very crucial for performance and the user experience of web applications in the dynamically developing context of web development. When applications grow in size and complexity, the choice of front-end framework becomes a more important decision that might affect not only the development process but also the long-term performance and maintainability of the application. The scalability - the efficiency with which a framework can handle greater volumes of work or more users - starts to become one of the major headaches for a developer who wants to build robust high-performance applications. Among the popular front-end frameworks that are in wide use today, React, Angular, and Vue.js maintain separate architectural philosophies, each with its own unique set of features [1].

React is an open-source JavaScript library developed by Facebook, noted for its component-based architecture and virtual DOM to achieve dynamic and efficient updates. Angular was designed by Google and refers to a complete framework that offers tight integration with two-way data binding and dependency injection, therefore providing more of an integrated approach to application development [2].

Vue.js' creator Evan You described it as being about simplicity and ease of integration. Taking the best ideas from both React and Angular, it makes it easier on the learning curve. Another key area the study looks at is how each framework

* Corresponding author: Mikita Piastou

handles upscaling an application in complexity and user interactions, looking at things such as load times, rendering speed, memory usage, and CPU consumption [3].

In addition, it will investigate the practical implications of these case study findings for developers regarding strengths and limitations in terms of large-scale application handling, and possible consequences for development practices.

2. Methods and Procedures

2.1. Framework Selection

Each of these above mentioned frameworks brings distinct features and strengths, making them suitable for a variety of development scenario s[4]. To provide a clear comparison, the following table highlights their key characteristics, including architecture, features, and suitability for different types of projects [5].

Table 1 Framework Overview

Framework	Developer	Architecture	Key Features
React	Facebook	Component-based, Virtual DOM	Reusable components, efficient updates, large ecosystem
Angular	Google	MVC (Model-View-Controller), Two-way data binding	Comprehensive tooling, dependency injection, strong typing (TypeScript)
Vue.js	Vue Technology	MVVM (Model-View-ViewModel)	Simple integration, progressive framework, lightweight

2.2. Application Benchmarking

To evaluate the scalability of the frameworks, we developed a benchmark application focused solely on a basic CRUD (Create, Read, Update, Delete) functionality[6]. It has been selected for the test case for the comparison of the basic performance each framework is supposed to offer due to the simplicity and wide relevance: load times, rendering speed, memory usage, responsiveness, and scalability[7].

2.3. Data Collection and Analysis

We used browser-based performance profiling tools such as Chrome DevTools to get real-time load time, rendering speed, memory usage, and CPU usage[8]. Further, data analysis was done by comparing the collected data of these frameworks over a wide range of metrics and scenarios[9].

2.4. Performance Metrics

Performance has been measured based on several key metrics, comprehensively testing the scalability and efficiency of React, Angular, and Vue.js frameworks[10]. These have given an idea of how each of the frameworks handles different aspects of application performance under varied conditions[11].

Table 2 Primary Performance Metrics

Metric	Submetric	Description
Load Times	Initial Load Time	Time taken for the application to load for the first time.
	Subsequent Load Time	Time taken for subsequent page loads or views, which can be affected by caching mechanisms.
Rendering Speed	Component Rendering Time	Time required to render individual components.
	DOM Updates	Time taken to update the Document Object Model (DOM) when changes occur.

Memory Usage	Peak Memory Usage	The highest amount of memory used by the application during its execution.
	Memory Leak Detection	Assessment of whether the framework manages memory effectively without leaks over time.
CPU Usage	Average CPU Usage	The average percentage of CPU resources used by the application.
	CPU Load during High Activity	CPU usage during peak activity periods, such as when processing large amounts of data or handling complex user interactions.

3. Experiment

3.1. Application Code

Below, you will find the code for the CRUD application built with React, Angular, and Vue.js, providing a detailed view of how each framework approaches these common tasks[12].

3.2. React

3.2.1. App.js

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const App = () => {
 const [items, setItems] = useState([]);
 const [newItem, setNewItem] = useState('');
 const [editingItem, setEditingItem] = useState(null);
 const [editText, setEditText] = useState('');

 useEffect(() => {
 axios.get('/api/items')
 .then(response => setItems(response.data))
 .catch(error => console.error('Error fetching data:', error));
 }, []);

 const handleAdd = () => {
 axios.post('/api/items', { name: newItem })
 .then(response => setItems([...items, response.data]))
 .catch(error => console.error('Error adding item:', error));
 };

 const handleDelete = (id) => {
 axios.delete(`/api/items/${id}`)
 .then(() => setItems(items.filter(item => item.id !== id)))
 .catch(error => console.error('Error deleting item:', error));
 };

 const handleEdit = (item) => {
 setEditingItem(item.id);
 setEditText(item.name);
 };

 const handleUpdate = () => {
 axios.put(`/api/items/${editingItem}`, { name: editText })
 .then(response => {
 setItems(items.map(item => item.id === editingItem ? response.data : item));
 setEditingItem(null);
```

```
setEditText('');
})
.catch(error => console.error('Error updating item:', error));
};

return (
<div>
<h1>React CRUD</h1>
<input value={newItem} onChange={e => setNewItem(e.target.value)} placeholder="New Item" />
<button onClick={handleAdd}>Add</button>
<ul>
{items.map(item => (
<li key={item.id}>
{editingItem === item.id ? (
<>
<input value={editText} onChange={e => setEditText(e.target.value)} />
<button onClick={handleUpdate}>Update</button>
<button onClick={() => setEditingItem(null)}>Cancel</button>
</>
) : (
<>
{item.name}
<button onClick={() => handleEdit(item)}>Edit</button>
<button onClick={() => handleDelete(item.id)}>Delete</button>
</>
)}
</li>
))}
</ul>
</div>
);
};

export default App;
```

3.3. Angular

3.3.1. crud.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
 selector: 'app-crud',
 templateUrl: './crud.component.html',
 styleUrls: ['./crud.component.css']
})
export class CrudComponent implements OnInit {
 items: any[] = [];
 newItem: string = '';
 editingItemId: number | null = null;
 editText: string = '';

 constructor(private http: HttpClient) { }

 ngOnInit(): void {
 this.http.get('/api/items').subscribe((data: any[]) => this.items = data);
```

```
}

addItem(): void {
this.http.post('/api/items', { name: this.newItem }).subscribe(item => {
this.items.push(item);
this.newItem = '';
});
}

deleteItem(id: number): void {
this.http.delete(`/api/items/${id}`).subscribe(() => {
this.items = this.items.filter(item => item.id !== id);
});
}

editItem(item: any): void {
this.editingItemId = item.id;
this.editText = item.name;
}

updateItem(): void {
if (this.editingItemId) {
this.http.put(`/api/items/${this.editingItemId}`, { name: this.editText }).subscribe((updatedItem: any) => {
this.items = this.items.map(item => item.id === this.editingItemId ? updatedItem : item);
this.editingItemId = null;
this.editText = '';
});
}
}
}
```

3.3.2. crud.component.html

```
<h1>Angular CRUD</h1>
<input [(ngModel)]="newItem" placeholder="New Item" />
<button (click)="addItem()">Add</button>
<ul>
 <li *ngFor="let item of items">
 <ng-container *ngIf="editingItemId === item.id; else viewMode">
 <input [(ngModel)]="editText" />
 <button (click)="updateItem()">Update</button>
 <button (click)="editingItemId = null">Cancel</button>
 </ng-container>
 <ng-template #viewMode>
 {{ item.name }}
 <button (click)="editItem(item)">Edit</button>
 <button (click)="deleteItem(item.id)">Delete</button>
 </ng-template>
 </li>
</ul>
```

3.4. Vue.js

3.4.1. Crud.vue

```
<template>
 <div>
 <h1>Vue.js CRUD</h1>
 <input v-model="newItem" placeholder="New Item" />
 <button @click="addItem">Add</button>
 <ul>
 <li v-for="item in items" :key="item.id">
 <div v-if="editingItemId === item.id">
 <input v-model="editText" />
 <button @click="updateItem">Update</button>
 <button @click="cancelEdit">Cancel</button>
 </div>
 <div v-else>
 {{ item.name }}
 <button @click="editItem(item)">Edit</button>
 <button @click="deleteItem(item.id)">Delete</button>
 </div>
 </li>
 </ul>
 </div>
</template>

<script>
import axios from 'axios';

export default {
 data() {
 return {
 items: [],
 newItem: '',
 editingItemId: null,
 editText: ''
 };
 },
 created() {
 axios.get('/api/items')
 .then(response => this.items = response.data)
 .catch(error => console.error('Error fetching data:', error));
 },
 methods: {
 addItem() {
 axios.post('/api/items', { name: this.newItem })
 .then(response => {
 this.items.push(response.data);
 this.newItem = '';
 })
 .catch(error => console.error('Error adding item:', error));
 },
 deleteItem(id) {
 axios.delete(`/api/items/${id}`)
 .then(() => {
 this.items = this.items.filter(item => item.id !== id);
 })
 .catch(error => console.error('Error deleting item:', error));
 },
 editItem(item) {
 this.editingItemId = item.id;
```

```
this.editText = item.name;
},
updateItem() {
axios.put(`/api/items/${this.editingItemId}`, { name: this.editText })
.then(response => {
this.items = this.items.map(item => item.id === this.editingItemId ? response.data : item);
this.editingItemId = null;
this.editText = '';
})
.catch(error => console.error('Error updating item:', error));
},
cancelEdit() {
this.editingItemId = null;
this.editText = '';
}
}
};
</script>

<style scoped>
/* Add your styles here */
</style>
```

3.5. Measurement Procedures

To measure the performance metrics for the React, Angular, and Vue.js CRUD applications, we used a combination of browser-based tools and profiling techniques[13]. The table provides a clear overview of how each performance metric was measured, including the procedures and the specific metrics to observe for evaluation[14][15].

Table 3 Performance Metrics Measurement Procedures

Metric	**Procedure**	**Submetric**
Load Times	1. Open Google Chrome and navigate to your application. 2. Open DevTools (F12 or right-click and select "Inspect"). 3. Go to the *Performance* tab. 4. Click the *Reload* button. 5. Stop recording after the page is fully loaded.	*Initial Load Time:* Time from navigation start to full load. *Subsequent Load Time:* Time for subsequent loads, affected by caching.
Rendering Speed	1. Open DevTools and navigate to the *Performance* tab. 2. Click the *Record* button and perform actions that trigger rendering. 3. Stop recording after actions.	*Component Rendering Time:* Time to render individual components. *DOM Updates:* Time taken for DOM updates.
Memory Usage	1. Open DevTools and go to the *Memory* tab. 2. Take a *Heap Snapshot* before and after performing actions. 3. Compare snapshots.	*Peak Memory Usage:* Highest memory usage during execution. *Memory Leak Detection:* Identify uncollected or increasing memory usage.
CPU Usage	1. Open DevTools and go to the *Performance* tab. 2. Click the *Record* button and perform CPU-intensive actions. 3. Stop recording after the actions.	*Average CPU Usage:* Average percentage of CPU resources used. *CPU Load during High Activity:* CPU usage during peak activity periods.

3.6. Measurement Results

3.6.1. Load Times

React demonstrated fast load times due to its virtual DOM diffing, though performance can vary with the complexity of state management[16]. In contrast, Angular had longer initial load times, attributed to its comprehensive framework and two-way data binding. Vue.js was efficient in both loading and rendering, thanks to its lightweight design.

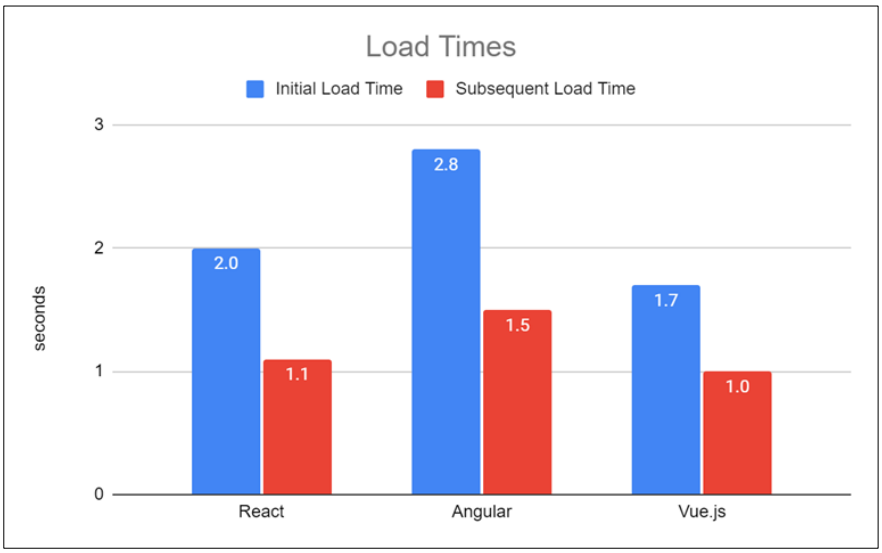

Figure 1 Load Times

3.6.2. Rendering Speed

React showed efficient rendering with its virtual DOM, but performance may be impacted by frequent state changes. Angular's rendering speed was slower, primarily due to the complexity of its two-way data binding and change detection mechanisms[17]. Vue.js provided balanced performance with simplicity, often achieving faster component rendering times.

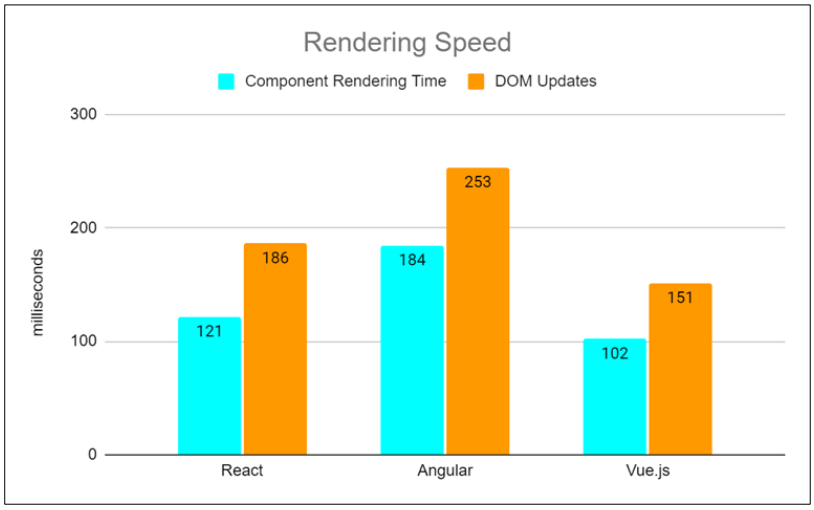

Figure 2 Rendering Speed

3.6.3. Peak Memory Usage

React was optimized but may incur higher memory usage in large applications because of the virtual DOM management. Angular consumed more memory due to its extensive framework features. Vue.js, benefiting from its lightweight nature, used less memory overall[18].

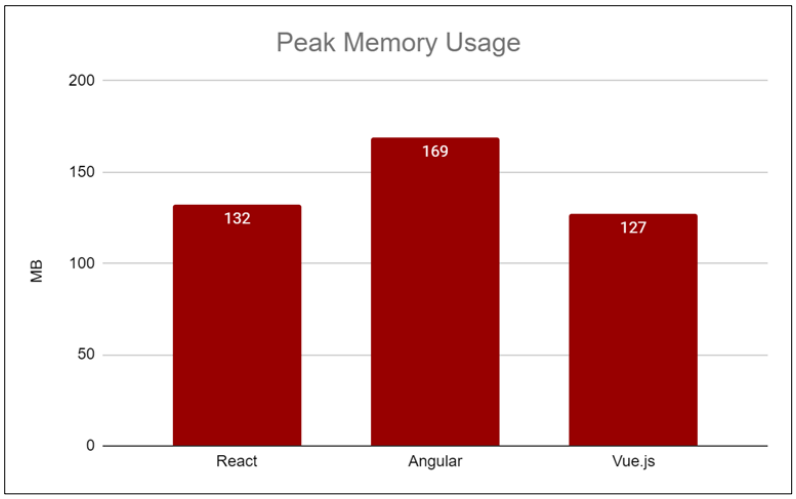

Figure 3 Peak Memory Usage

Table 4 Memory Leak Detection

Framework	Memory Leak Frequency
React	Minor leaks detected
Angular	Occasional leaks detected
Vue.js	Rarely leaks detected

3.6.4. CPU Usage

React was optimized for performance, though high CPU usage may occur during intensive rendering tasks. Angular exhibited higher CPU usage during peak activity, due to the heavy nature of its framework. Vue.js demonstrated efficient CPU usage, benefiting from its minimalistic approach[19].

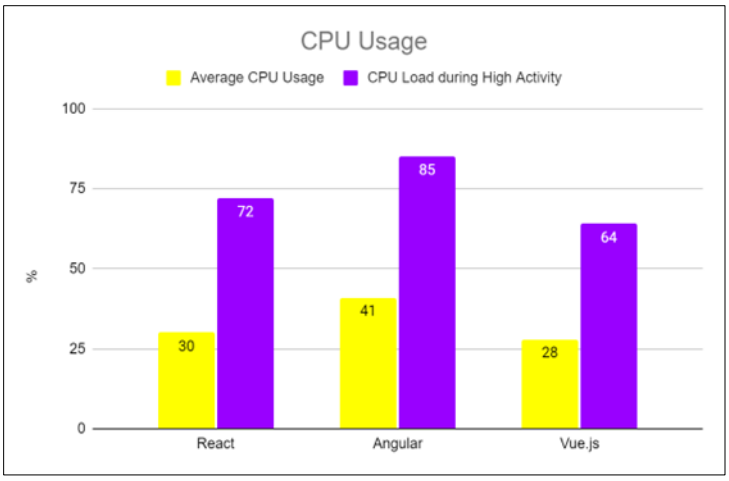

Figure 4 CPU Usage

4. Research Findings and Discussion

React demonstrated efficient performance with its virtual DOM, though it has a tendency to result in performance bottlenecks if it uses very complex state management unless appropriately optimized. It scales well for large applications and especially with advanced techniques such as code splitting and server-side rendering.

Angular performed well because of the strong framework; this is mainly with trade-offs: slower page loads and higher memory use, mainly because of the feature set and two-way data binding. The framework does a great job of scaling, considering its high modularity and dependency injection, though with the requirement of being careful with performance optimization to avoid degradation [20].

Vue.js provided a balanced approach with competitive performance, lower memory usage, and effective scalability, making it versatile for both small and large applications. It ensured effective scaling in both small and large applications with explicit needs for uncomplicated state management [21].

5. Conclusion

In this comprehensive assessment of React, Angular, and Vue.js, we have observed distinct performance characteristics and scalability potentials across these leading front-end frameworks. React excels with its virtual DOM, providing efficient rendering and scalability, though it may require optimization for complex state management. Angular, while powerful and feature-rich, faces challenges with longer load times and higher memory usage, needing careful performance optimization. Vue.js strikes an exceptional balance, offering competitive performance and lower memory usage, making it versatile for various application sizes and complexities.

Ultimately, the choice of framework should align with the specific needs of the project, taking into account factors such as performance requirements, scalability, and overall architecture.

References

[1] R. Vyas, “Comparative Analysis on Front-End Frameworks for Web Applications”, International Journal for Research in Applied Science and Engineering Technology (IJRASET), vol. 10, issue 7, pp. 298-307, Jul. 2022.

[2] React Team, “React Documentation,” React, [Online]. Available: https://react.dev/. [Accessed: Jun. 17, 2023].

[3] Angular, “Introduction to the Angular docs,” Angular, [Online]. Available: https://v17.angular.io/docs. [Accessed: Jun. 2, 2023].

[4] Evan You, “Introduction,” Vue.js, [Online]. Available: https://vuejs.org/. [Accessed: May 22, 2023].

[5] B. T. Negara, “Frontend Framework Consideration for IT Developer”, Jurnal Teknik Informatika dan Sistem Informasi (JATISI), vol. 9, no 2, 2022.

[6] M. Levlin, “DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte”, Abo Akademi University, 2020.

[7] D. Raimundo, C. Figueiredo, G. Russo, “Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue”, 2022 XVLIII Latin American Computer Conference (CLEI), Oct. 2022.

[8] P. Singh, M. Srivastava, M. Kansal, A. P. Singh, A. Chauhan, A. Gaur, “A Comparative Analysis of Modern Frontend Frameworks for Building Large-Scale Web Applications”, 2023 International Conference on Disruptive Technologies (ICDT), May 2023.

[9] C. Tsislianis, “Front-end JavaScript Frameworks: React.js & Vue.js”, University of Thessaly, School of Engineering, Department of Electrical and Computer Engineering, Feb. 2022.

[10] M. Siahaan, V. O. Vianto, “Comparative Analysis Study of Front-End JavaScript Frameworks Performance Using Lighthouse Tool”, Teknologi Informatika dan Komunikasi (Mantik), vol. 6, no 3, Nov. 2022.

[11] I. H. Madurapperuma, M. S. Shafana, M. J. A. Sabani, “State-of-art frameworks for front-end and back-end web development”, 2nd International Conference - 2022 (ICST2022), Aug. 2022.

[12] S. Abrahamsson, “A model to evaluate front-end frameworks for single page applications written in JavaScript”, Digitala Vetenskapliga Arkivet, p. 45, May 2023.

[13] S. Abrahamsson, “A model to evaluate front-end frameworks for single page applications written in JavaScript”, Digitala Vetenskapliga Arkivet, p. 45, May 2023.

[14] A. Shukla, “Modern JavaScript Frameworks and JavaScript's Future as a FullStack Programming Language”, Journal of Artificial Intelligence & Cloud Computing, vol. 2, no 4, pp. 1-5, 2023.

[15] E. Petukhova, “Sitecore JavaScript Services Framework Comparison”, Abo Akademi University, 2019.

[16] D. Verma, “A Comparison of Web Framework Efficiency - Performance and network analysis of modern web frameworks”, Turku University of Applied Sciences, 2022.

[17] A. Meredova, “Comparison of Server-Side Rendering Capabilities of React and Vue”, Haaga-Helia University of Applied Sciences, 2023.

[18] J. Wimmer, “Analysis and Evaluation of JavaScript- and WebAssembly-Based Front-End Frameworks for Enterprise Applications”, University of Applied Sciences Vienna, FH Technikum Wien, May 2020.

[19] S. Wazni, “Development of a React-Based Portfolio for Front-End Usage”, Turku University of Applied Sciences, 2023.

[20] F. Ilievska, S. Gramatikov, “Analysis and comparative evaluation of front-end technologies for web application development”, Ss. Cyril and Methodius University in Skopje, 2022.

[21] D. Wernersson, V. Sjolund, “Choosing a Rendering Framework: A Comparative Evaluation of Modern JavaScript Rendering Frameworks”, Linnaeus University, Faculty of Technology, Department of computer science and media technology (CM), p. 43, 2023.