

EVENTS@SATHYABAMA - EVENT MANAGEMENT SYSTEM

Submitted in partial fulfillment of the
requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

ARYAN AMISH

(Reg. No – 40110122)

BANDEPALLI SURYA ANJANI KUMAR

(Reg. No – 40110156)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY - I UNIVERSITY BY UGC

Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHISALAI,

CHENNAI - 600119

APRIL – 2024



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

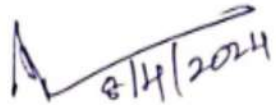
CATEGORY - 1 UNIVERSITY BY UGC

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **ARYAN AMISH (40110122)** and **BANDEPALLI SURYA ANJANI KUMAR (40110156)** who carried out the project entitled "**EVENTS@SATHYABAMA - EVENT MANAGEMENT SYSTEM**" under our supervision from November 2023 to April 2024.

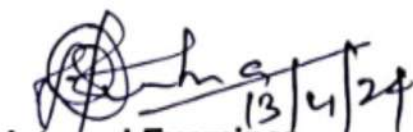

8/4/2024

Internal Guide

Dr. A. MARY POSONIA, M.E., Ph.D.

HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Jeyapalan Nagar, Gandhi Salai,
Chennai - 600 110.
Dr. E. Jeyapalan, M.E., Ph.D.

Submitted for Viva Voce Examination held on 13/04/2024


13/4/24
Internal Examiner


13/4
External Examiner

DECLARATION

We, **Aryan Amish (Reg. No – 40110122)** and **Bandepalli Surya Anjani Kumar (Reg. No – 40110156)**, hereby declare that the Project Report entitled **“EVENTS@SATHYABAMA - EVENT MANAGEMENT SYSTEM”** was done by us under the guidance of **Dr. A. MARY POSONIA, M.E., Ph.D.**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering Degree in **Computer Science and Engineering**.

DATE: 13/4/24

PLACE: Chennai

Aryan

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **BOARD OF MANAGEMENT OF SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D.**, Dean, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews

I would like to express my sincere and deep sense of gratitude to my Project Guide, **Dr. A. MARY POSONIA, M.E., Ph.D.**, for her valuable guidance, suggestions, and constant encouragement that paved the way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING** who were helpful in many ways for the completion of the project.

ABSTRACT

The Event Management System is a web-based application that simplifies the process of managing events within a college campus. It can be used to advertise events to the whole college or a specific branch easily.

Every event created must pass through three levels of acceptance, namely Head of Department, Dean and Vice Chancellor, before it is shown to students. The system has powerful smart authentication that distinguishes between a student, VC, Teacher or HOD. The application allows teachers to create and post events, and students to view and apply to events. The system provides the event organizer the option to accept or deny a student's request to participate in an event.

The data model includes tables for events, users, departments, and certificates. The events table stores details such as the name, description, date, time, location, and maximum capacity. The users table stores information about users, including their name, email, and password. The departments table stores information about the different departments within the college. The certificates table stores information about the certificates that are issued to students upon completion of an event.

The system architecture is based on the Model-View-Controller (MVC) pattern, with the backend implemented using Django and the frontend built with NextJS, React and Tailwind CSS. The application is deployed using Docker and is hosted on a cloud platform like AWS or GCP.

The Event Management System provides a user-friendly interface for teachers to create and post events and for students to view and apply to events. It also ensures that events are properly approved before they are visible to students and provides a streamlined process for issuing certificates to students after the completion of an event.

CONTENTS

CHAPTER NO.	TITLE		PAGE NO.
	ABSTRACT		i
	LIST OF FIGURES		v
	LIST OF ABBREVIATIONS		vii
1	INTRODUCTION		1 - 2
	1.1	Why Events and Event Management Systems Matter?	1
	1.2	Introduction to Web Development	1
	1.3	Introduction to Front-end Web Development	1
	1.4	Introduction to Back-end Web Development	2
	1.5	What is an API?	2
	1.6	What is a Database?	2
	1.7	What is Deployment?	2
2	LITERATURE SURVEY		3 - 5
	2.1	Existing Research and Literature	3
	2.2	Inferences from Literature Survey	4
	2.3	Open problems in Existing System	4
3	REQUIREMENT ANALYSIS		6 - 11

	3.1	Feasibility Studies	7
	3.2	Software Requirements Specification	8
4	DESCRIPTION OF PROPOSED SYSTEM		12 - 19
	4.1	User Roles and Functionalities	12
	4.2	System Features	12
	4.3	System Architecture	13
	4.4	Continuous Integration / Continuous Deployment (CI/CD)	16
	4.5	Project Budget And Scalability	17
	4.6	Project Management Plan	18
5	IMPLEMENTATION DETAILS		20 - 23
	5.1	Technology Stack	20
	5.2	Testing	22
6	RESULT AND PERFORMANCE ANALYSIS		24 - 33
	6.1	Comparing Desktop Performance	24
	6.2	Comparing Mobile Performance	25
	6.3	Comparing Desktop First Contentful Paint	26
	6.4	Comparing Mobile First Contentful Paint	27
	6.5	Comparing Desktop Largest Contentful Paint	28
	6.6	Comparing Mobile Largest Contentful Paint	29

	6.7	Analysis Of Desktop Total Blocking Time	30
	6.8	Analysis Of Mobile Total Blocking Time	31
	6.9	Analysis Of Desktop Speed Index	32
	6.10	Analysis Of Mobile Speed Index	33
7	CONCLUSION		34
8	REFERENCES		35
	APPENDIX		36
	A	SOURCE CODE	36
	B	SCREENSHOTS	50
	C	RESEARCH PAPER	53

LIST OF FIGURES

FIGURE NO.	FIGURE TITLE	PAGE NO.
3.1	Event Multi-Level Approval Process	8
4.1	Frontend (Nextjs) Architecture	13
4.2	Backend (Nextjs) Architecture	15
5.1	Illustration On How SSR Works	21
6.1	Performance Of Server-Side Rendering and Client-Side Rendering (Desktop)	24
6.2	Performance Of Server-Side Rendering and Client-Side Rendering (Mobile)	25
6.3	First Contentful Paint (Desktop)	26
6.4	First Contentful Paint (Mobile)	27
6.5	Largest Contentful Paint (Desktop)	28
6.6	Largest Contentful Paint (Mobile)	29
6.7	Total Blocking Time (Desktop)	30
6.8	Total Blocking Time (Mobile)	31
6.9	Speed Index (Desktop)	32
6.10	Speed Index (Mobile)	33
B.1	Portal Landing Page	50
B.2	Portal Home Page	50
B.3	Event Details Page	51

B.4	Approval Process	51
B.5	Event Creation Page	52
B.6	User Profile Page	52

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
API	Application Programming Interface
CDN	Content Delivery Network
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management Systems
DRF	Django Rest Framework
ES6	EcmaScript 6
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
NPM	Node Package Manager
REST	Representational state transfer
RWD	Responsive Website Development
SSR	Server-Side Rendering
CSR	Client-Side Rendering

CHAPTER 1

INTRODUCTION

Organizing events can be a daunting task, especially in large institutions like colleges. From planning to execution, it requires meticulous attention to detail, coordination, and effective communication. To simplify this process, the Event Management System is a web-based application that provides an efficient and streamlined solution for managing events within a college campus.

1.1. Why Events and Event Management Systems Matter?

Events are a critical part of student life, providing opportunities for personal growth, social interaction, and professional development. From orientation week to graduation ceremonies, colleges host various events throughout the academic year. The complexity of event management in colleges highlights the need for efficient and effective event management systems. Such systems simplify the event management process, enabling event planners to manage events seamlessly, while providing students with a platform to view and apply to events quickly and easily.

1.2. Introduction to Web Development

Web developers often work for clients who are trying to get their product or service onto the web. The work is typically very project focused and involves collaborating with a team that helps to coordinate the client's needs into the end product. The client could be an individual, a tech company, an organization, or a government.

The work could involve

- Front-end (Client-Side Development)
- Back-end (Server-Side Development) [or]
- Full-stack web development.

1.3. Introduction to Front-end Web Development

The front-end is the stuff you see on the website in your browser, including the presentation of content and user interface elements like the navigation bar. Front-end developers use HTML, CSS, JavaScript, and their relevant frameworks to ensure that

content is presented effectively and that users have excellent experience.

1.4. Introduction to Back-end Web Development

The back end refers to the guts of the application, which live on the server. The back end stores and serves program data to ensure that the front end has what it needs. This process can become very complicated when a website has millions of users. Back-end developers use programming languages like JavaScript, Java, Python, and Ruby to work with data.

1.5. What is an API?

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API. Modern APIs adhere to standards (typically HTTP and REST), that are developer-friendly, easily accessible and understood broadly.

1.6. What is a Database?

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just a database.

1.7. What is Deployment?

Deployment is the process of making software available to be used on a system by users and other programs.

CHAPTER 2

LITERATURE SURVEY

The referenced works collectively cover a range of topics pertinent to modern software development. These include comparisons between client-side and server-side rendering technologies like React.js and Next.js, insights into event management system design and implementation, discussions on designing scalable and reliable distributed systems, and principles for creating clean and maintainable software architectures. Each work contributes valuable perspectives on improving web application performance, building scalable event-centric applications, and designing robust distributed systems and software structures.

2.1. OVERVIEW OF EXISTING RESEARCH AND LITERATURE

"React Apps with Server-Side Rendering: Next.js" by Harish A Jartarghar et al [1] compares React.js client-side rendering with Next.js server-side rendering. The findings indicated that Next.js, through its server-side rendering approach, significantly enhances web page loading speed. However, one limitation of this study is the absence of specific performance metrics or case studies, which could provide more comprehensive insights into the benefits of using Next.js for server-side rendering.

"Modern Front End Web Architectures with React.Js and Next.Js" by Mochammad Fariz Syah Lazuardy [2], a thorough comparison of the advantages and disadvantages associated with these technologies. The study systematically highlights the merits and demerits of React.js and Next.js. Notably, it identifies potential challenges, including the occurrence of a white screen during the initial loading phase and the requisite utilization of additional libraries for routing and state management.

"Event Management System: A Comprehensive Guide" by John Smith [3] John Smith's guide offers a comprehensive overview of event management systems, covering aspects such as system design, user requirements, features, and implementation strategies. It provides insights into building scalable and efficient event-centric applications.

"Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services" by Brendan Burns [4]. This book by Brendan Burns focuses on designing distributed systems with an emphasis on scalability and reliability. It likely discusses architectural patterns, fault tolerance mechanisms, and scalability strategies essential for building robust distributed systems.

"Clean Architecture: A Craftsman's Guide to Software Structure and Design" by Robert C. Martin [5]. Robert C. Martin's book delves into software architecture principles with a focus on clean, maintainable designs. It covers topics such as architectural patterns, separation of concerns, and best practices for structuring software systems to facilitate maintainability and extensibility.

2.2. INFERENCES FROM LITERATURE SURVEY

The literature survey highlights significant findings regarding the use of React.js and Next.js, particularly in the context of server-side rendering (SSR) for web development. Notably, Next.js, through its SSR approach, is shown to notably improve web page loading speed. The survey systematically discusses the advantages and disadvantages of both technologies, with React.js being associated with challenges like the initial white screen and the need for additional libraries. Data fetching with SSR is seen as beneficial for improved page load times and SEO, but it raises concerns about increased server load and limitations for real-time applications. Caching techniques and data manipulation are recognized as valuable tools, yet require careful management. Moreover, SSR's impact on mobile performance shows promise in reducing loading times and bounce rates, emphasizing the importance of mobile-specific optimization while avoiding over-optimization pitfalls.

2.3. OPEN PROBLEMS IN EXISTING SYSTEM

In modern web development, the choice between Client-Side Rendering (CSR) and Server-Side Rendering (SSR) is a critical consideration that directly impacts the performance, user experience, and functionality of web applications. CSR, where rendering and processing are primarily handled on the client's browser using JavaScript, presents challenges such as slower initial page load times, potential SEO limitations, and complex client logic management. Conversely, SSR involves rendering content on the server before sending it to the client, offering benefits like

improved SEO, faster First Contentful Paint (FCP), and reduced resource consumption on client devices. This introduction sets the stage for understanding the nuances and implications of choosing between CSR and SSR in web development, highlighting the key factors that developers and stakeholders must weigh when designing and optimizing modern web applications.

- **Initial Page Load Speed:** CSR often leads to slower initial page load times, especially for larger and more complex web applications. This can result in poor user experience.
- **SEO and Indexing:** CSR can present challenges for search engine optimization (SEO) because search engines may have difficulty crawling and indexing content rendered on the client side. SSR, on the other hand, generates pre-rendered HTML content that is more accessible to search engines.
- **First Contentful Paint (FCP):** CSR may suffer from delayed FCP, meaning users see a blank or partially rendered page before the JavaScript and data are fully loaded and processed. SSR can provide a faster FCP by serving a pre-rendered page.
- **Mobile Performance:** On mobile devices, CSR can lead to sluggish performance and longer load times, which can result in higher bounce rates. SSR, when optimized for mobile, can offer better performance.
- **Resource and Bandwidth Consumption:** CSR can be resource-intensive, requiring the client to download and execute JavaScript, potentially consuming more bandwidth and device resources. SSR can reduce this burden on the client.
- **Complex Client Logic:** Managing complex client-side logic, especially in large applications, can lead to challenges in terms of code organization.
- **JavaScript Dependency Size:** CSR often involves the use of extensive JavaScript libraries and dependencies, which can increase the size of the application and impact load times. SSR may reduce the need for large client-side libraries.
- **Security Concerns:** Client-side rendering can introduce security concerns, such as potential cross-site scripting (XSS) vulnerabilities when handling user input or third-party scripts.

- **User Experience Consistency:** Achieving a consistent user experience across various devices and network conditions can be more challenging with CSR. SSR can provide a more consistent experience due to server-generated content.
- **Offline Access:** CSR applications may not work well offline or under limited network connectivity, while SSR can provide a better experience in such scenarios.

CHAPTER 3

REQUIREMENT ANALYSIS

The Event Management System aims to streamline and automate the event management process within your college. This system will have various user roles, each with specific functionalities and permissions. Here is a detailed requirement analysis for the project:

3.1. FEASIBILITY STUDIES

3.1.1. Technical Feasibility:

Technology Stack: Evaluate the chosen technologies (Next.js, React, Django, MySQL) for compatibility, scalability, and support.

Data Integration: Examine the feasibility of integrating with the college's existing ERP system, including data extraction, transformation, and synchronization.

3.1.2. Economic Feasibility:

Cost Estimation: Calculate the initial development costs, including hardware, software, and human resources.

Operating Costs: Estimate ongoing expenses, such as server hosting, maintenance, and support.

Return on Investment (ROI): Analyze potential returns, including increased efficiency and reduced administrative workload for staff.

3.1.3. Legal and Compliance Feasibility:

Compliance with College Policies: Ensure that the system aligns with the college's rules, guidelines, and academic standards.

3.1.4. Operational Feasibility:

User Acceptance: Conduct surveys or interviews with potential users (students, teachers, administrators) to gauge their acceptance of the system.

Training Needs: Identify training requirements for users to ensure smooth adoption.

Integration with Existing Systems: Evaluate the compatibility of the new system with any existing processes and tools.

3.1.5. Schedule Feasibility:

Dependencies: Identify dependencies that could affect project milestones.

Resource Availability: Ensure that required resources are available when needed.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

3.2.1. User Authentication:

- Secure login and registration for all user roles.
- One common login form for all 5 different user roles.
- User-specific dashboards with relevant options.

3.2.2 Event Creation:

- Faculty members and higher can create events.
- Specify event details (name, date, location, description).
- Attach event documents if necessary.

3.2.3 Event Approval Workflow:

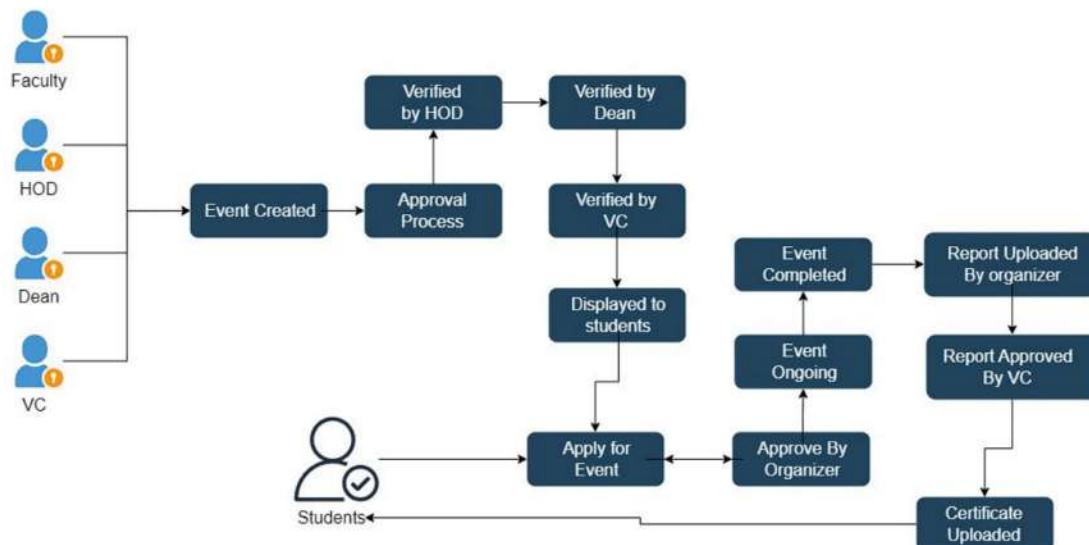


Figure 3.1 Event Multi-Level Approval Process

Events must go through a multi-level approval process as shown in Figure 3.1.

3.2.4. Event Application:

- Students can apply for events.
- Event eligibility criteria based on batch and branch.
- Students' applications can be rejected or accepted by the organizers of the event.

3.2.5. Report Submission:

- Faculty and student organizers can submit event reports.
- Reports must be reviewed and approved by the Vice Chancellor.

3.2.6. Certificate Management:

- Organizers upload event certificates.
- Students can download certificates upon approval.

3.2.7. Notifications:

- Automated email notifications for event approvals, rejections, and report approvals.
- Alerts for pending actions on the user's dashboard.

3.2.8. Security:

- Data encryption for sensitive information.
- Role-based access control to ensure users can only perform authorized actions.
- Regular security audits and updates.

3.2.9. Data Integration with ERP

- To successfully collect and utilize student data for the Event Management System, data integration with the existing ERP (Enterprise Resource Planning) system of the college is essential.

3.2.10. Data Extraction:

- Data Sources: Identify the specific data sources within the ERP system that contain student information, including register number, name, email, branch, and batch. These sources could include databases, student information systems, or other data repositories.
- Data Transfer: Develop a mechanism for extracting data from the ERP system. This could involve APIs, database queries, or other methods, depending on the ERP system's capabilities.

3.2.11. Data Transformation:

- Data Mapping: Create a mapping between the data fields in the ERP system and the corresponding fields in the Event Management System. This ensures that data is correctly transferred and aligned between the systems.

- **Data Cleaning:** Implement data cleaning and transformation processes to ensure that the collected data is consistent and accurate. This may include handling data format inconsistencies, duplicates, or missing information.

3.2.12. Data Transfer:

- **Data Transfer Frequency:** Determine how often data will be transferred from the ERP system to the Event Management System. This could be a daily, weekly, or real-time process, depending on the data update frequency in the ERP system.
- **Data Transfer Protocols:** Ensure secure and efficient data transfer protocols are in place to move data from the ERP system to the Event Management System.

3.2.13. Data Storage and Management:

- **Data Repository:** Establish a data repository within the Event Management System where the collected student data will be stored. This could be a dedicated database or storage solution.
- **Data Synchronization:** Implement mechanisms for data synchronization to keep the collected student data up-to-date with changes in the ERP system.

3.2.14. Data Security and Privacy:

- **Data Encryption:** Secure the data transfer and storage with encryption to protect student information during transit and while at rest.
- **Data Access Control:** Implement access controls to ensure that only authorized personnel can view and modify the integrated data.

3.2.15. Server Hosting:

- **Web Hosting Servers:** Deploy the Event Management System on web hosting servers that are capable of handling the application's traffic and load efficiently. Consider using cloud-based hosting services, like Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure, which offer scalability and reliability.
- **Front-End Hosting:** Host the front-end, built with Next.js, on a suitable hosting service. Vercel, a platform specializing in hosting Next.js applications, is a recommended choice due to its ease of use and integration with the framework.
- **Back-End Hosting:** The Django back-end should be hosted on a server capable of running Python applications. You can use cloud-based servers, virtual private

servers (VPS), or a dedicated server depending on your resource requirements.

- Database Hosting: Ensure the database server is set up and hosted, either on the same hosting service or on a separate server. Consider Railway for database management if applicable.
- Scalability: Configure auto-scaling capabilities on the hosting servers to handle traffic fluctuations efficiently. This ensures that the system remains responsive during peak loads without incurring unnecessary costs during low-traffic periods.

The proposed Event Management System shown in Figure 3.1 is a comprehensive and innovative software solution designed to enhance the management of events within our college. This system is engineered to streamline the event creation, approval, participation, and documentation processes, providing a structured and efficient approach for all stakeholders, including students, faculty, Heads of Departments (HODs), Deans, and the Vice Chancellor.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

The event management system described encompasses five distinct user roles, each assigned specific functionalities tailored to their position within an academic institution. Students are granted the ability to apply for events, assuming they meet eligibility requirements, and can also serve as organizers when designated by faculty members.

4.1. USER ROLES AND FUNCTIONALITIES:

The system will encompass five distinct user roles, each with specific functionalities:

- **Student:** Students will be able to apply for events, provided they meet the eligibility criteria based on their batch and branch. They can also act as organizers if appointed by a faculty member. Students will be responsible for submitting event reports and downloading event certificates upon approval.
- **Faculty:** Faculty members will have the privilege of creating events. However, their created events will undergo an approval process. They will also be responsible for approving or rejecting event reports and uploading event certificates.
- **HOD (Head of Department):** HODs will have the authority to approve events that fall within their respective departments. Once approved, events will proceed to the Dean for further evaluation.
- **Dean:** Deans will oversee the approval process for events from all departments. They will decide which events advance to the final approval stage, led by the Vice Chancellor.
- **Vice Chancellor:** The Vice Chancellor will serve as the final authority in approving events. Only those granted approval at this level will be accessible to students for participation.

4.2. SYSTEM FEATURES:

- **User Authentication:** The system will implement secure login and registration procedures, ensuring user privacy and data protection.
- **Event Creation:** Faculty members can create events with relevant details, attachments, and documents.

- **Event Approval Workflow:** Events will follow a structured multi-level approval workflow, with notifications to keep users informed about the status of their events.
- **Report Submission:** Faculty and student organizers will be able to submit event reports, which must undergo review and approval by the Vice Chancellor.
- **Certificate Management:** Event organizers will upload certificates, and students can conveniently download approved certificates.

4.3. SYSTEM ARCHITECTURE

4.3.1. Frontend Architecture

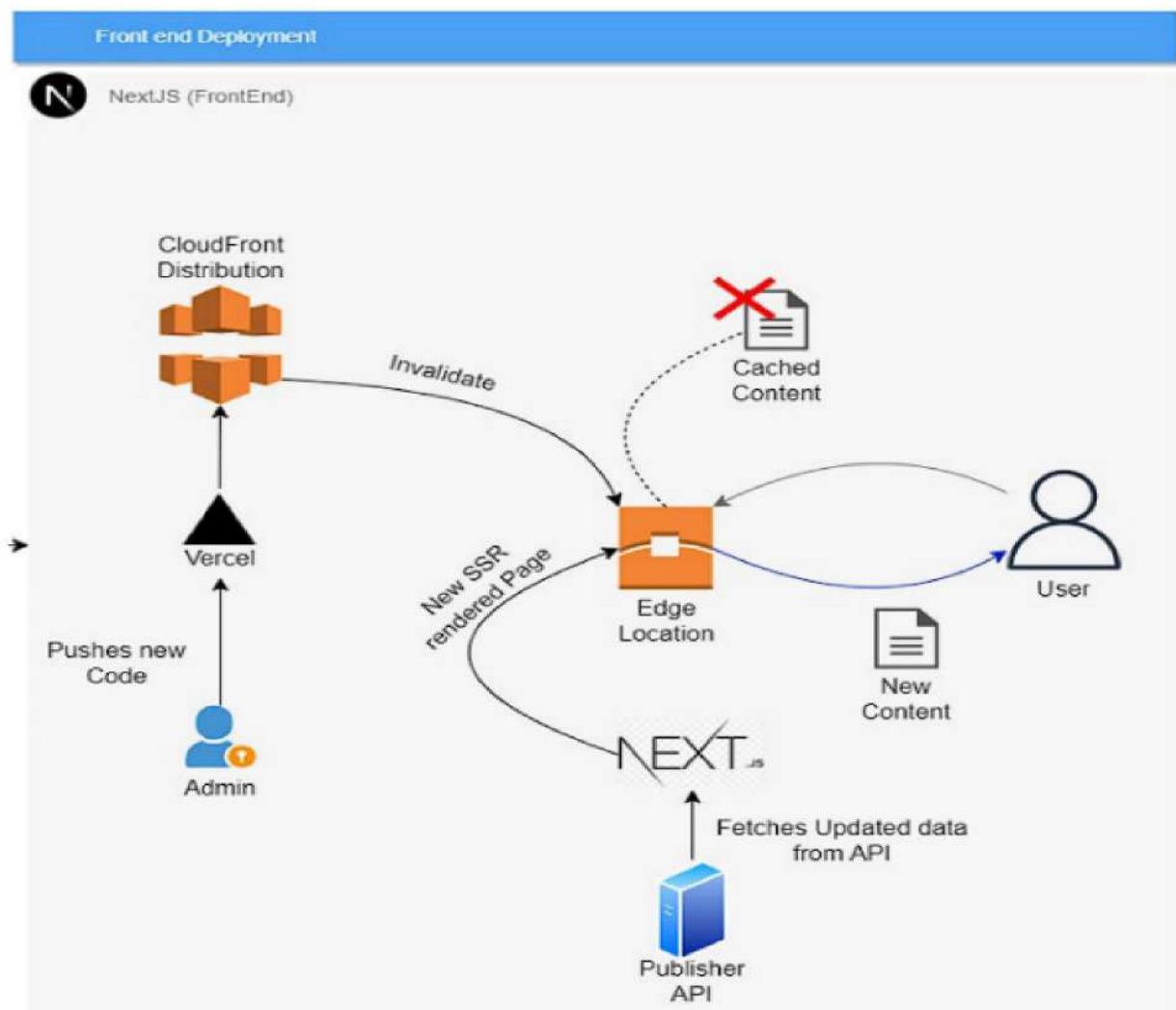


Figure 4.1 Frontend (Nextjs) Architecture

Frontend: The Next.js application itself. This is where your code lives, including the React components, pages, and API routes refer Figure 4.1.

CloudFront distribution: A CloudFront distribution is a content delivery network (CDN) that serves your static and dynamic content to users around the world with low latency and high availability.

Vercel: Vercel is a cloud platform that is specifically designed for hosting and deploying Next.js applications.

Publisher API: An API that provides the data that your Next.js application needs to render its pages.

Brief overview of how the architecture works:

- A user visits your website.
- Their request is routed to the CloudFront distribution.
- CloudFront checks its cache for the requested page. If the page is in the cache, CloudFront serves it to the user.
- If the page is not in the cache, CloudFront requests it from Vercel.
- Vercel renders the page using the Next.js runtime and the data from the Publisher API.
- Vercel then returns the rendered page to CloudFront.
- CloudFront caches the rendered page and serves it to the user.

This architecture has a number of advantages:

- **Performance:** CloudFront is a global CDN, so users can access your website quickly from anywhere in the world.
- **Scalability:** CloudFront can automatically scale to handle increased traffic.
- **Security:** CloudFront provides a number of security features, such as encryption and DDoS protection.
- **Reliability:** CloudFront is highly reliable, with a 99.99% uptime SLA.

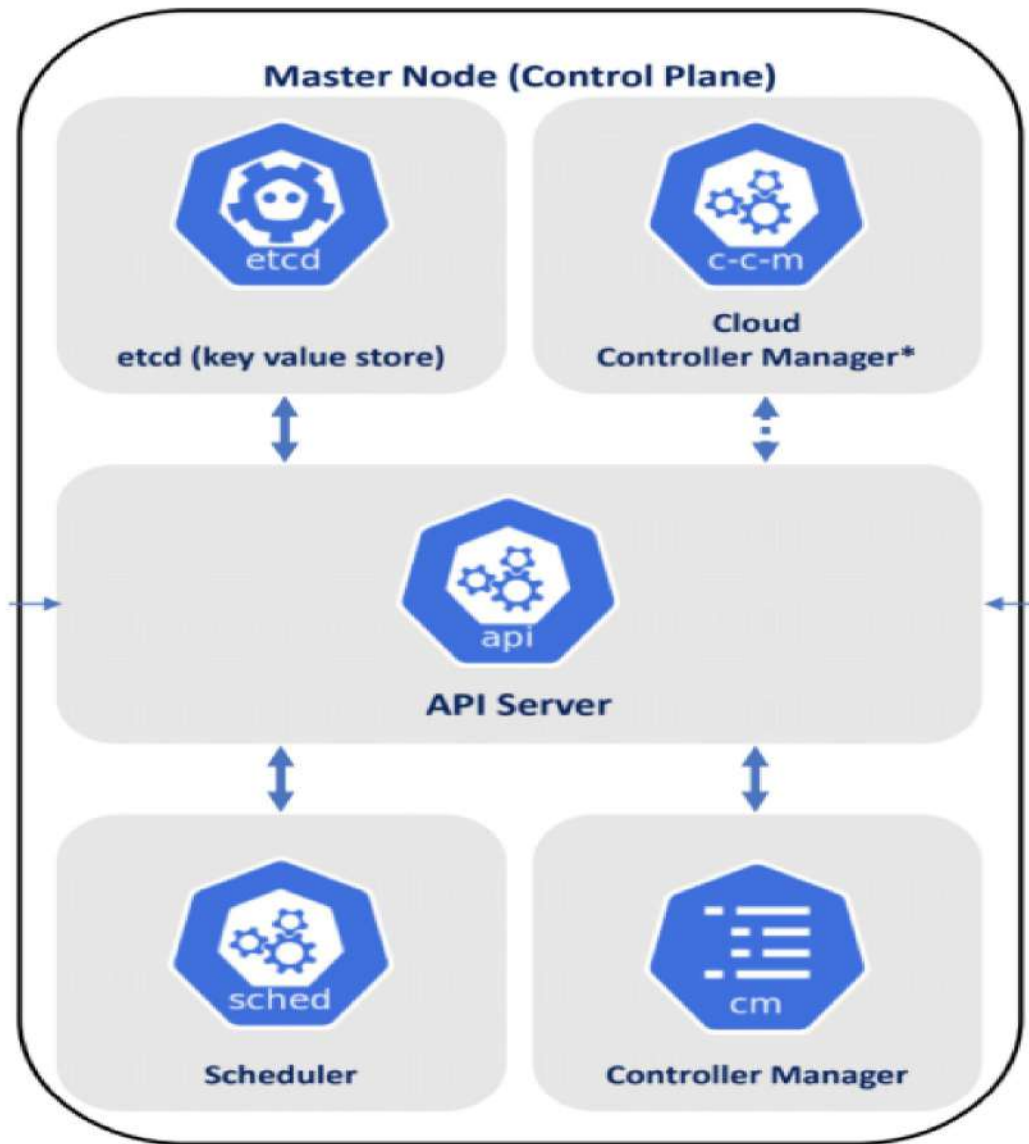


Figure 4.2 Backend Architecture

4.3.2. Backend Architecture

It is designed to handle many concurrent users and requests, while also being resilient to failures.

The architecture consists of the following components as shown in the Figure 4.2:

- **Kubernetes Cluster:** Kubernetes is a container orchestration platform that automates the deployment, management, and scaling of containerized applications. In this architecture, Kubernetes is used to manage the deployment and scaling of the Django application and its supporting services.
- **MySQL Cluster:** MySQL is a popular open-source relational database management system (RDBMS). In this architecture, MySQL is used to store the

Django application's data. The MySQL cluster is configured to be highly available, with multiple replicas of the data stored on different nodes.

- **Redis Cluster:** Redis is an open-source in-memory data structure store. In this architecture, Redis is used to cache frequently accessed data, such as database queries and session data. The Redis cluster is also configured to be highly available.
- **Django Application:** The Django application is deployed to multiple Kubernetes pods. This allows the application to be scaled horizontally by adding or removing pods.
- **Load balancer:** The load balancer distributes traffic across the Django application pods. This helps to ensure that no single pod is overloaded.
- **Reverse proxy:** The reverse proxy acts as a single point of entry for the Django application. It terminates SSL/TLS connections and forwards requests to the appropriate Django application pod.
- **Logging server:** The logging server collects and stores logs from the Django application and its supporting services.

4.3.3. User Interface:

The system will feature user-friendly interfaces tailored to each user role, ensuring that users can easily access and interact with the functions most relevant to them. A dashboard will provide quick access to actions and updates.

4.3.4. Security:

Security is a paramount concern, and the system will employ data encryption and role-based access control to safeguard sensitive information. Regular security audits and updates will be conducted to maintain system integrity.

4.4 CONTINUOUS INTEGRATION/CONTINUOUS DEPLOYMENT (CI/CD):

CI/CD Workflow:

1. **Version Control with Git:** The project source code is managed using Git, allowing for collaborative development and version control. Developers work on features and fixes in isolated branches.
2. **Continuous Integration with Vercel:** The CI/CD pipeline begins with continuous

integration provided by Vercel. Whenever changes are pushed to the version control system (Git), Vercel automatically triggers a build process. This ensures that new code changes are built, tested, and integrated into the system.

3. **Automated Testing:** During the CI phase, automated tests are executed to validate the quality and stability of the code. Front-end tests, back-end tests, and any other relevant tests are performed to catch any regressions or issues early in the development process.
4. **Deployment to Vercel:** Once the code passes the automated tests, Vercel deploys the updated application to its hosting environment. This process is automated, minimizing the need for manual intervention and reducing deployment errors.
5. **Continuous Deployment with Railway:** Railway, a database hosting service, integrates with the CI/CD pipeline to manage database migrations and updates. Any database schema changes are tracked and managed seamlessly using Railway, ensuring data consistency and schema evolution.
6. **Monitoring and Rollback:** After deployment, the system's performance and stability are monitored. If any issues or regressions are detected, automated rollback mechanisms can revert to the previous stable version, maintaining system reliability.

4.5. PROJECT BUDGET AND SCALABILITY:

The budget for the Event Management System is estimated to be in the range of *Rs.3,000* to *Rs.4,000* during peak load, where all college students are using the portal regularly. This budget is calculated to ensure the system's smooth operation and is structured to accommodate the auto-scaling feature.

One of the key features of the system is its auto-scalability, which enables it to dynamically adapt to varying levels of user demand without compromising the speed and quality of service. This ensures that the system can efficiently manage resources, scaling up when needed during peak load and scaling down during periods of lower usage to optimize costs.

4.6 PROJECT MANAGEMENT PLAN

The Project Management Plan outlines the strategy for developing and implementing the Event Management System for the college. This plan addresses the key aspects of project management, including scope, schedule, resources, communication, risk management, and quality assurance.

4.6.1. Goals:

- Design and develop a user-friendly Event Management System.
- Implement user authentication, roles, and permissions for system security.
- Enable event creation and approval workflows.
- Provide event application and registration features for students.
- Implement report submission and certificate management functionalities.
- Ensure notifications and alerts for users.
- Integrate the system with the college's ERP for student data.
- Deploy the system on web hosting servers for scalability.

4.6.2. Timeline:

Month 1 (March): Project Planning and Requirement Gathering.

Month 2 (April): System Design and Architecture Planning.

Month 3 (May): Front-End Development (Next.js, React, TypeScript, Material UI, etc.).

Month 4 (June): Back-End Development (Django, Django REST Framework, MySQL).

Month 5 (July): Integration with ERP System.

Month 6 (August): Testing and Quality Assurance.

Month 7 (September): Deployment on Google Cloud Platform (GCP) and Vercel.

Month 8 (October): Documentation and Final Presentation.

4.6.3. Hardware Requirements to Develop the Project:

1. CPU: Intel i3 10th Gen or higher
2. RAM: 4 GB or higher
3. Storage: 10 GB available space recommended
4. Internet Connection: Required

4.6.4. Software Requirements to Develop the Project:

1. VS Code – (Recommended Code Editor)

Extensions that have to be installed:

- Prettier (Code Formatter)
- Auto Close Tag
- IntelliCode
- Tailwind CSS suggestions

2. NodeJS – v18.14.2 or later

3. NPM – v9.5.0 or later

Dependencies that have to be installed:

- material-ui v5.11.11
- react v18.2.0
- next v13.2.3
- git v2.37.0

4. Tailwind CSS – v3.2.7 or later

5. Python – v3.10.5 or later

6. Django – v4.1.7 or later

CHAPTER 5

IMPLEMENTATION DETAILS AND METHODOLOGY

The implementation of the Next.js and Django project involves combining a front-end and back-end architecture to create a full-stack web application. Next.js, a React framework, is utilized for building the frontend, providing features like server-side rendering, routing, and state management. This enables efficient client-side interactions and seamless navigation within the application. On the other hand, Django, a powerful Python web framework, is employed for developing the backend. Django handles data modeling, business logic, and API endpoints, facilitating robust server-side operations. The frontend communicates with the backend through RESTful API calls, allowing data exchange and interaction between the user interface and the server. This project architecture leverages the strengths of both frameworks: Next.js for dynamic and responsive frontend development and Django for secure and scalable backend operations. The implementation encapsulates best practices in web development, including modular design, separation of concerns, and efficient data flow between the client and server components. Overall, the Next.js and Django combination forms a versatile and effective solution for building modern, feature-rich web applications.

5.1 TECHNOLOGY STACK:

- **Next.js:** Next.js is a popular React framework that provides server-side rendering (SSR), automatic code splitting, and routing for React applications. SSR (Figure 5.1) improves the initial page load time by rendering HTML on the server.

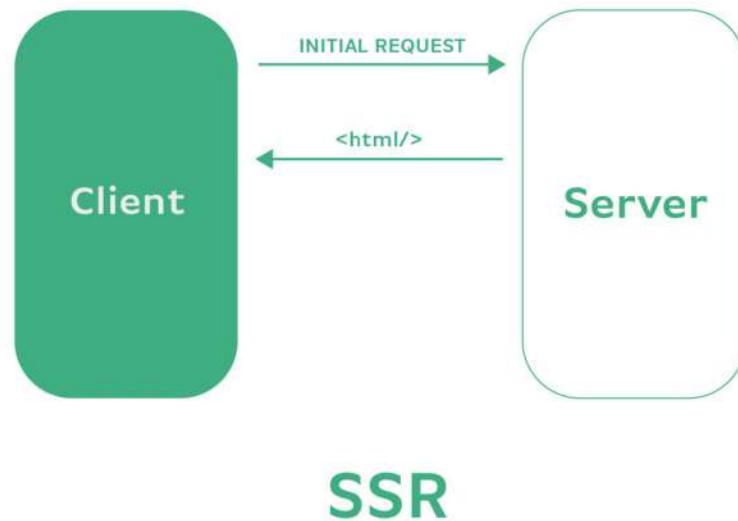


Figure 5.1 Illustration on How SSR Works

- **ReactJS:** React is a JavaScript library for building user interfaces. It's used as the primary framework for developing the front-end of the application. React allows for the creation of reusable UI components.
- **TypeScript:** TypeScript is a statically typed superset of JavaScript. It adds type checking to the code, making it more robust and maintainable, especially in larger projects.
- **ESLint:** ESLint is a static code analysis tool that helps developers identify and fix problems in their JavaScript and TypeScript code. It enforces coding standards and best practices.
- **Material UI:** Material UI is a popular React UI framework that provides pre-designed components and styles following the Material Design guidelines. It simplifies the creation of a visually appealing and consistent user interface.
- **HTML5:** HTML5 is the latest version of the Hypertext Markup Language. It's the standard markup language for creating web pages and applications.
- **CSS3:** CSS3 is the latest version of Cascading Style Sheets. It's used to style HTML elements, providing control over layout and presentation.
- **Tailwind CSS:** Tailwind CSS is a utility-first CSS framework. It offers predefined classes that can be used to style HTML elements quickly and easily. It simplifies the styling process.
- **Axios:** Axios is a popular JavaScript library used for making HTTP requests from the front-end to the back end. It's commonly used for API interactions.

- **Formik:** Formik is a library for managing forms in React applications. It simplifies form handling, validation, and submission.
- **Yup:** Yup is a JavaScript library for form validation. It's often used in combination with Formik to define validation rules for form fields.
- **Other NPM Libraries:** Various other npm libraries are used to facilitate specific functionalities or streamline development processes.
- **Django:** Django is a high-level Python web framework that provides a set of tools for building web applications rapidly. It's used as the primary framework for developing the back end of the application.
- **Django REST Framework:** Django REST Framework is a powerful toolkit for building Web APIs in Django. It facilitates the creation of RESTful APIs for communication between the front-end and back-end.
- **MySQL:** MySQL is an open-source relational database management system (RDBMS). It's used as the database for storing and managing application data.
- **Google Cloud Platform (GCP):** GCP is a cloud computing platform that provides various cloud services, including server hosting, storage, and data services. It's used for deploying and hosting the application in the cloud.
- **Vercel:** Vercel is a cloud platform designed for hosting web applications and static sites. It's often used for deploying Next.js applications due to its seamless integration.
- **Kubernetes:** Kubernetes is an open-source container orchestration platform. It's used to manage containerized applications, ensuring scalability, reliability, and efficient resource utilization.
- **Docker:** Docker is a containerization platform that allows applications and their dependencies to be packaged into containers. Containers are used for consistent deployment and scalability.

5.2 TESTING:

Prior to deployment, the system will undergo rigorous testing to evaluate its usability, security, and performance.

Testing was done in 4 phases:

1. *Functionality Testing:* It is the process that includes several testing parameters

like user interface, APIs, database testing, security testing, client and server testing and basic website functionalities. Functional testing is very convenient, and it allows users to perform both manual and automated testing. It is performed to test the functionalities of each feature on the website.

2. *Usability Testing*: It has now become a vital part of any web-based project. It can be carried out like a small focus group similar to the target audience of the web application.
3. *Interface Testing*: Three areas to be tested here are – Application, Web and Database Server.
4. *Database Testing*: Database is one critical component of your web application and stress must be laid to test it thoroughly.

Testing activities will include-

1. Test if any errors are shown while executing queries
2. Data Integrity is maintained while creating, updating or deleting data in the database.

5.2.1. *Testing Conditions*:

Configuration: Ryzen 7 5800H 8 CPU cores, Nvidia GTX 1650, 16 GB RAM

Server-Side and Client-Side were run on the local machine.

CHAPTER 6

RESULT AND PERFORMANCE ANALYSIS

React.js and Next.js are both JavaScript frameworks renowned for their performance in building modern web applications. React.js offers a flexible component-based architecture with a Virtual DOM, optimizing rendering efficiency by updating only necessary components upon data changes. This approach enhances scalability and responsiveness. Next.js builds upon React.js by integrating server-side rendering (SSR) and static site generation (SSG), reducing initial page load times and improving SEO. Next.js ability to pre-render pages at build time or on-demand further enhances perceived performance, making it ideal for projects where performance optimization, SEO, and scalability are paramount considerations. Ultimately, the choice between React.js and Next.js depends on project requirements, with Next.js offering distinct advantages for achieving superior performance in web applications.

6.1 COMPARING DESKTOP PERFORMANCE

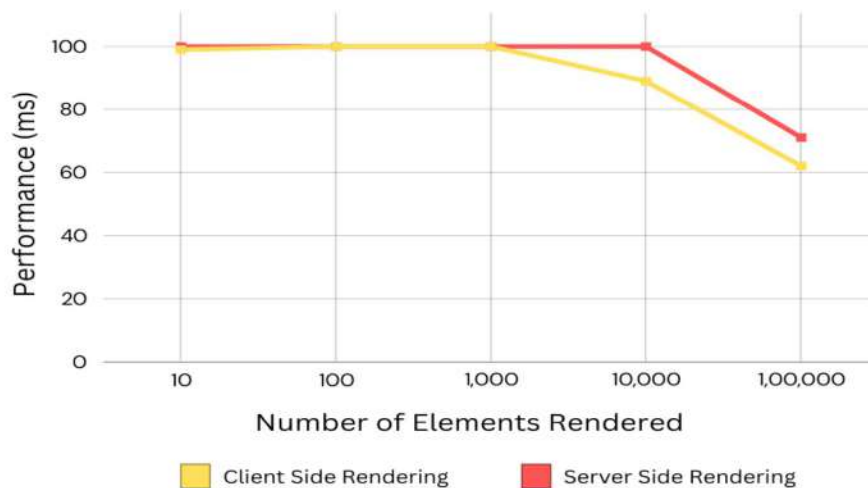


Figure 6.1 Performance of Server-Side Rendering and Client-Side Rendering (Desktop)

- The graph as shown in Figure 6.1 displays the desktop performance of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as elements increase.

- The performance of SSR and CSR is the same until 10^2 elements, but when it exceeds this number, SSR provides higher performance than CSR.

6.2 COMPARING MOBILE PERFORMANCE

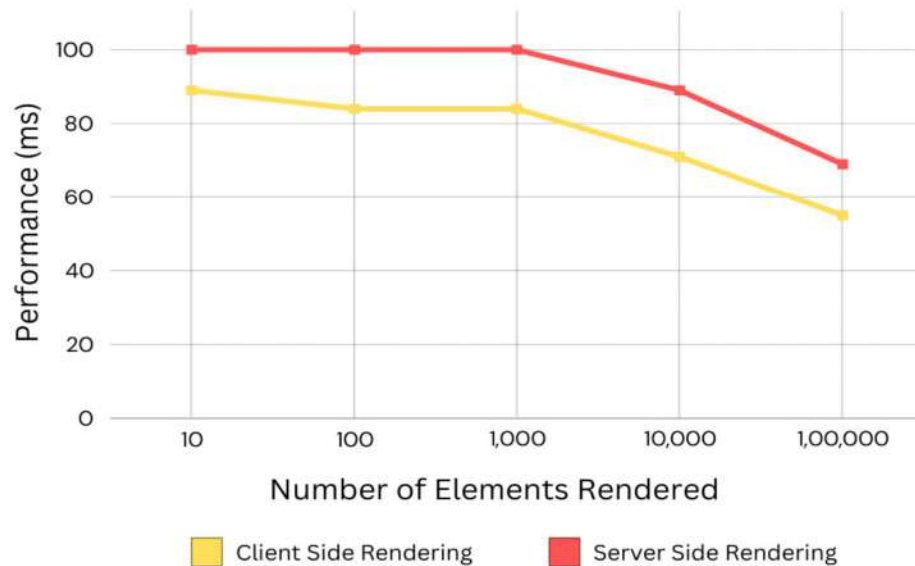


Figure 6.2 Performance of Server-Side Rendering and Client-Side Rendering (Mobile)

- Figure 6.2 displays the mobile performance of Next.js (SSR) and React.js (CSR) as the number of elements increases, as shown in Figure 6.2
- The performance of SSR is consistently higher than CSR from 10^0 to 10^4 .

6.3 COMPARING DESKTOP FIRST CONTENTFUL PAINT

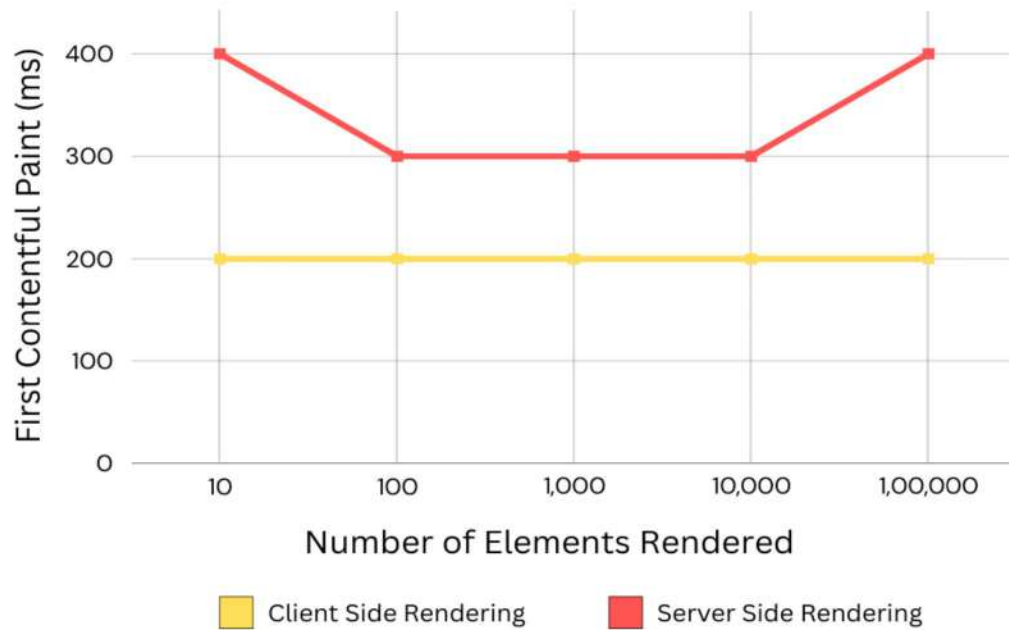


Figure 6.3 First Contentful Paint (Desktop)

- The graph in Figure 6.3 compares the desktop first contentful paint of Next.js (SSR) and React.js (CSR) as the number of elements increases.
- CSR consistently takes 200ms for the First Contentful Paint due to its client-side nature, while SSR can take longer due to server-side rendering and dynamic content generation.

6.4 COMPARING MOBILE FIRST CONTENTFUL PAINT

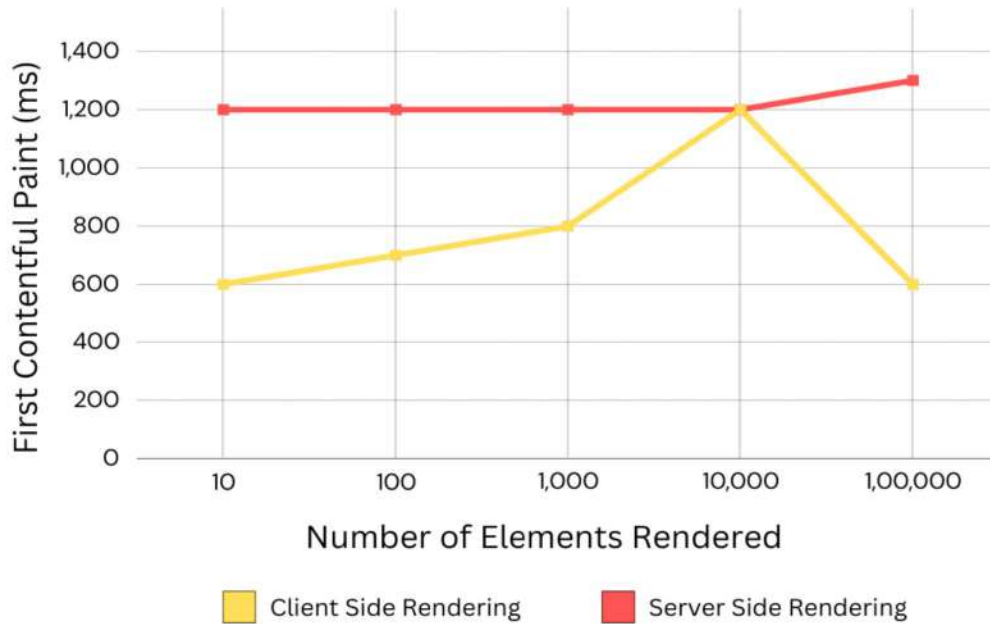


Figure 6.4 First Contentful Paint (Mobile)

- Figure 6.4 compares the mobile first contentful paint of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- In mobile, as shown in Figure 6.4 CSR takes the lead, but at 10^3 elements, it matches the time taken by SSR. However, it makes a comeback for 10^4 elements.

6.5 COMPARING DESKTOP LARGEST CONTENTFUL PAINT

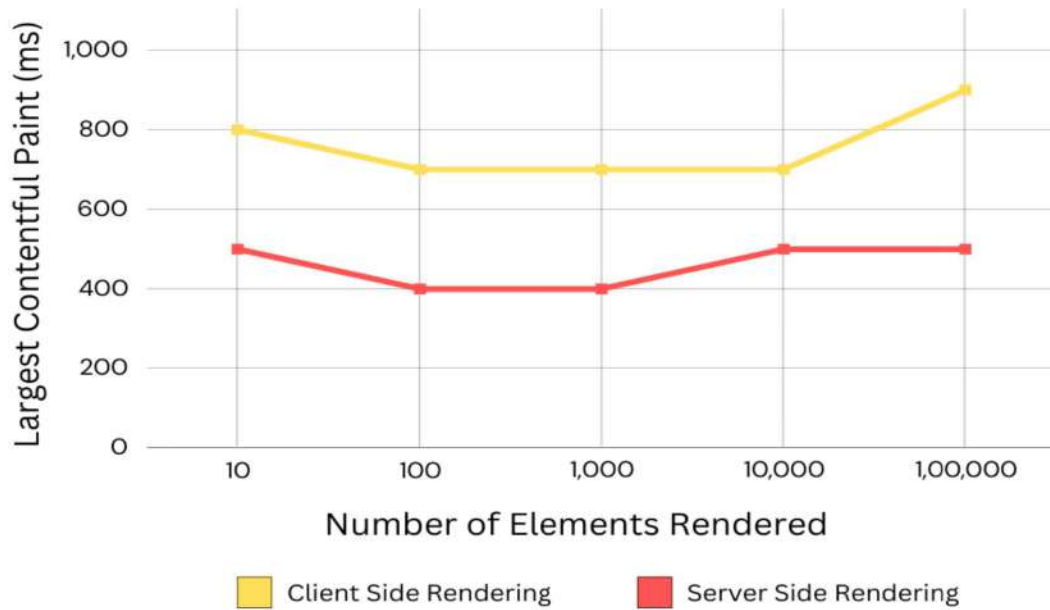


Figure 6.5 Largest Contentful Paint (Desktop)

- Figure 6.5 compares the desktop largest contentful paint of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- In terms of Largest Contentful Paint, SSR always leads on Desktop.

6.6 COMPARING MOBILE LARGEST CONTENTFUL PAINT

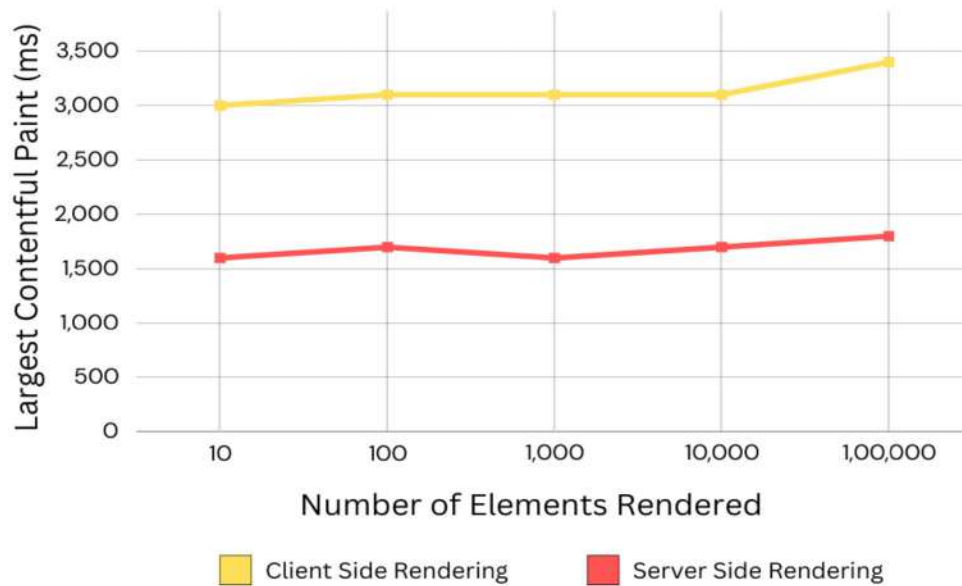


Figure 6.6 Largest Contentful Paint (Mobile)

- Figure 6.6 compares the mobile largest contentful paint of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- In terms of Largest Contentful Paint, SSR always leads on Mobile as well.

6.7 ANALYSIS OF DESKTOP TOTAL BLOCKING TIME

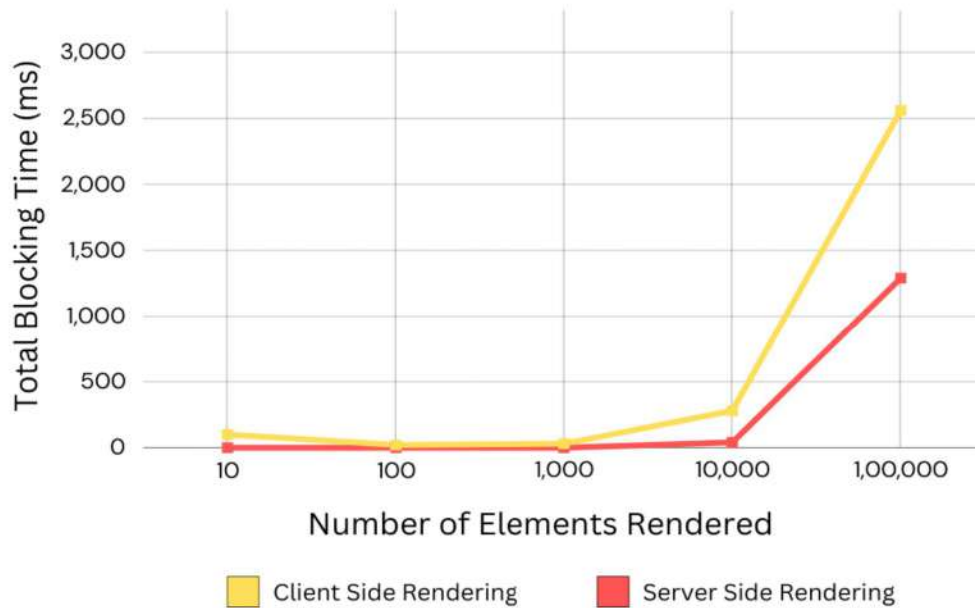


Figure 6.7 Total Blocking Time (Desktop)

- Figure 6.7 displays the desktop total blocking time of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- The total blocking time in desktop is comparatively lower in SSR than in CSR.

6.8 ANALYSIS OF MOBILE TOTAL BLOCKING TIME

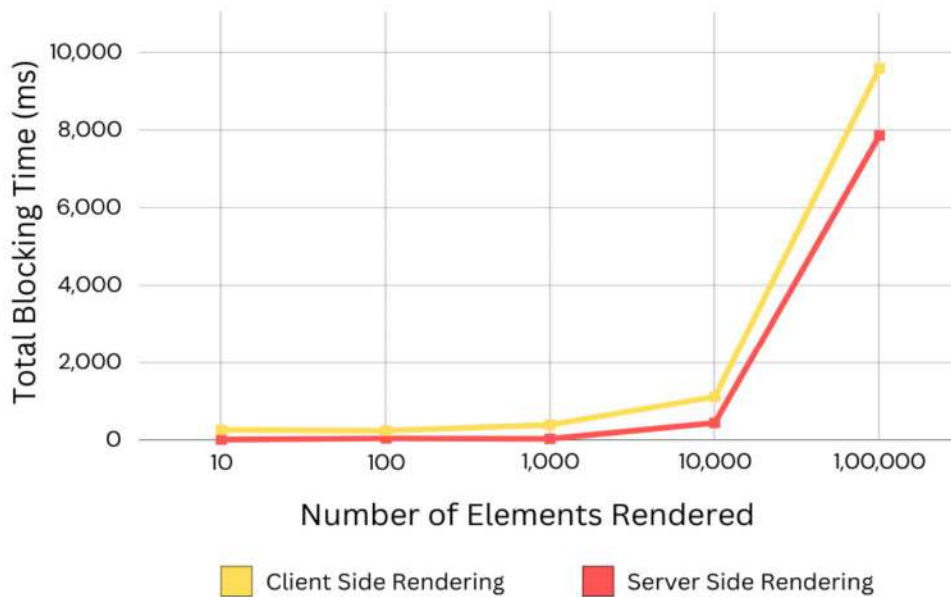


Figure 6.8 Total Blocking Time (Mobile)

- Figure 6.8 compares the mobile total blocking time of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- The total blocking time in mobile is comparatively lower in SSR than in CSR.

6.9 ANALYSIS OF DESKTOP SPEED INDEX

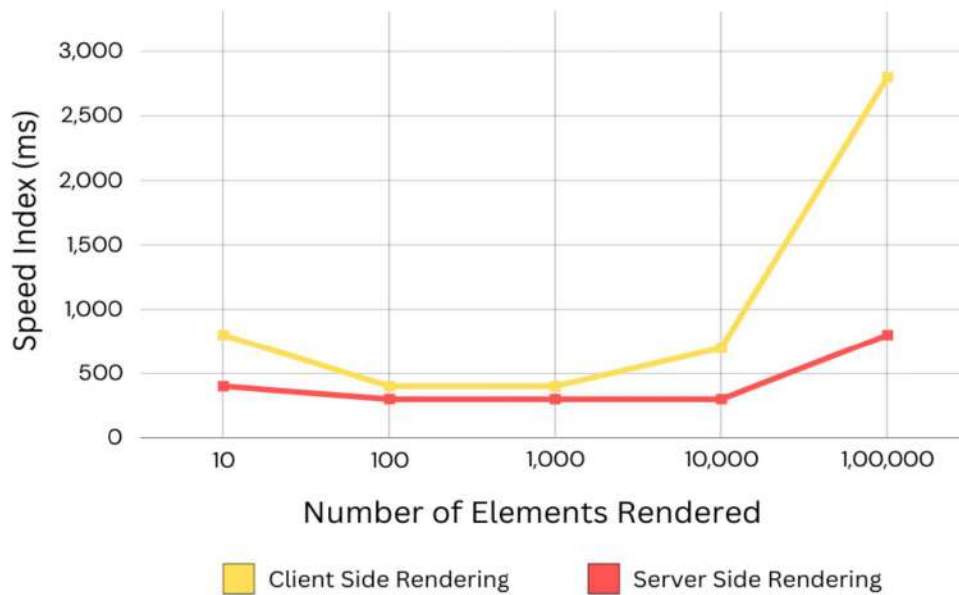


Figure 6.9 Speed Index (Desktop)

- Figure 6.9 compares the desktop speed index of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- The Desktop Speed Index also underscores SSR's superior performance over CSR.

6.10 ANALYSIS OF MOBILE SPEED INDEX

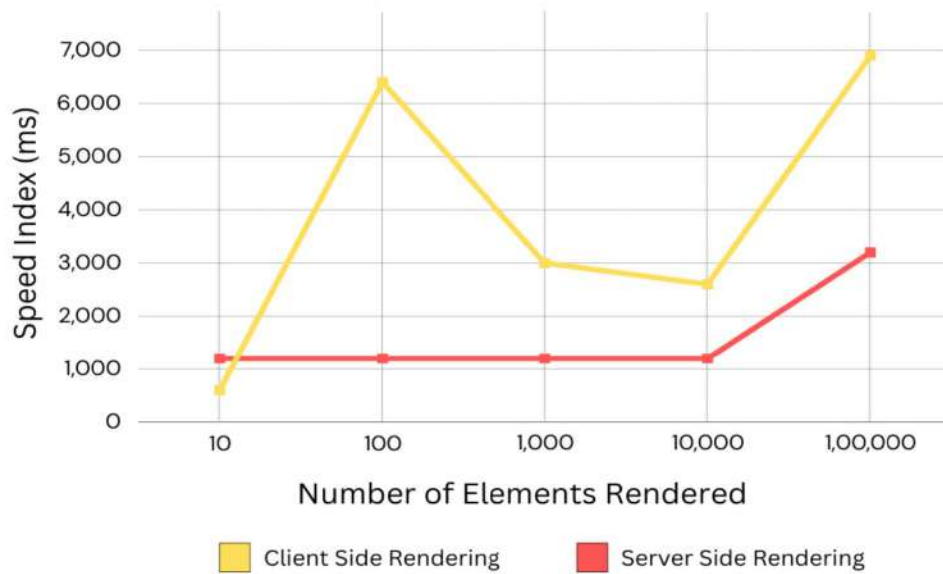


Figure 6.10 Speed Index (Mobile)

- Figure 6.10 compares the mobile speed index of Next.js server-side rendering (SSR) and React.js client-side rendering (CSR) as the number of elements increases.
- The Mobile Speed Index also underscores SSR's superior performance over CSR.

CHAPTER 7

CONCLUSION

In conclusion, the Event Management System presented in this report offers a sophisticated solution tailored to the diverse needs of event organization within our college. By addressing key challenges such as event creation, approval workflows, participant engagement, and documentation management, this system aims to revolutionize the way events are managed and executed.

Through the delineation of user roles and functionalities, the system ensures seamless collaboration among students, faculty members, Heads of Departments, Deans, and the Vice Chancellor.

The architectural overview highlights the robustness and scalability of the system, leveraging cutting-edge technologies such as Next.js, React, Django, and MySQL. With a focus on security, usability, and performance, the system's architecture is designed to meet the evolving needs of our college community.

Furthermore, the testing phases and continuous integration/continuous deployment (CI/CD) pipeline underscore our commitment to delivering a high-quality, reliable, and scalable solution. Rigorous testing ensures the system's functionality, usability, and security, while the CI/CD workflow facilitates seamless deployment and maintenance.

Overall, the Event Management System represents a significant step forward in modernizing event management practices within our college. By leveraging technology, automation, and collaborative frameworks, we aim to enhance the student experience, streamline administrative processes, and foster a vibrant and engaging campus community.

REFERENCES:

- [1] "React Apps with Server-Side Rendering: Next.js" - Harish A Jartarghar¹, Girish Rao Salanke¹, Ashok Kumar A.R¹, Sharvani G.S¹ and Shivakumar Dalali published in Journal of Telecommunication, Electronic and Computer Engineering, Vol. 14 No. 4.
- [2] "Modern Front End Web Architectures with React.Js and Next.Js" - Mochammad Fariz Syah Lazuardy published in International Research Journal of Advanced Engineering and Science: 2455-9024.
- [3] "Event Management System: A Comprehensive Guide" by John Smith, published in Event Planning Journal, vol. 20, no. 2, pp. 30-45, 2020.
- [4] "Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services" by Brendan Burns, published by O'Reilly Media, 2018.
- [5] "Clean Architecture: A Craftsman's Guide to Software Structure and Design" by Robert C. Martin, published by Prentice Hall, 201.
- [6] Django documentation.
<https://docs.djangoproject.com/en/3.2/>
- [7] Django Rest Framework documentation.
<https://www.django-rest-framework.org/>
- [8] Next.js documentation.
<https://nextjs.org/docs/getting-started>
- [9] Tailwind CSS documentation.
<https://tailwindcss.com/docs>
- [10] React documentation.
<https://reactjs.org/docs/getting-started.html>
- [11] MySQL documentation.
<https://dev.mysql.com/doc/>

APPENDIX

A. SOURCE CODE

```
@api_view(['POST'])
@is_authenticated
@is_event_organizer
def application_approval(request, event_id):
    data_dict = {data.get('pk'): '3' if data.get(
'status') == 1 else '1' for data in request.data}
    participants_to_update = []

    # Fetch the existing participants from the database
    existing_participants = EventParticipant.objects.filter(
user_id__in=data_dict.keys(), event=event_id, owner=False, organizer=False)
    participant_dict = {
participant.user_id: participant for participant in existing_participants}

    # Update the statuses and store them in the list
    for user_id, status in data_dict.items():
        participant = participant_dict.get(user_id)
        if participant:
            participant.status = status
            participants_to_update.append(participant)

    # Perform bulk update using bulk_update() method
    with transaction.atomic():
        EventParticipant.objects.bulk_update(
participants_to_update,
fields=['status'])
```

```

)

return Response({'detail': message.applicaition_approval.success, 'status': 200})


@api_view(['POST'])
@is_authenticated
@required_roles([2, 3, 4])
def approve_event(request, event_id):
    msg = message.event_approval
    user_role = request.user.role
    user = request.user
    approve_message = request.data.get('message')
    if user_role < 2:
        return Response(data={'detail': msg.forbidden}, status=403)

    event = Event.objects.filter(pk=event_id)
    response_already_approved = Response(
        data={'detail': msg.already_approved}, status=400)
    if not event.exists():
        return Response(data={'detail': msg.not_found}, status=404)
    event = event.first()
    if event.rejected:
        return Response(data={'detail': msg.event_rejected}, status=400)
    if event.status >= 2:
        return response_already_approved
    try:
        user_data = user_serializer.UserDetail(user).data
        if user_role == 2: # hod

```

```

if event.hod_verified:
    return response_already_approved
    event.hod_verified = True

if event.owner.branch:
    if user.branch and event.owner.branch.name == user.branch.name:
        pass
    else:
        return Response(data={'detail':
            msg.branch_didnt_matched.format(event.owner.branch.name)}, status=400)
    if not event.create_timeline(level=1, user=user_data, msg=approve_message,
        status=TimeLineStatus.completed):
        raise Exception
    event.save()

elif user_role == 3: # Dean
    if event.dean_verified:
        return response_already_approved
    if event.hod_verified is True:
        event.dean_verified = True
    if not event.create_timeline(level=2, user=user_data, msg=approve_message,
        status=TimeLineStatus.completed):
        raise Exception
    event.save()
    else:
        return Response(data={'detail': msg.verify_hod_first})

elif user_role == 4: # VC
    if event.vc_verified:

```

```

return response_already_approved

if event.hod_verified is False:
    return Response(data={'detail': msg.verify_hod_first})

if event.dean_verified is False:
    return Response(data={'detail': msg.verify_dean_first})

event.vc_verified = True

event.status = 2

if not event.create_timeline(level=3, user=user_data, msg=approve_message,
status=TimeLineStatus.completed):

    raise Exception

if not event.create_timeline(level=4,
user=Portal_User,
msg="",
status=TimeLineStatus.completed):

    raise Exception

event.save()

except:

    return Response(data={'detail': msg.server_error}, status=500)

return Response(data={'detail': msg.success})

```

```

@api_view(['POST'])
@is_authenticated
@required_roles([2, 3, 4])
def deny_event(request, event_id):
    msg = message.event_deny
    user_role = request.user.role
    user = request.user

```

```

deny_message = request.data.get('message')
if user_role < 2:
    return Response(data={'detail': msg.forbidden}, status=403)
event = Event.objects.filter(pk=event_id)

response_already_approved = Response(
    data={'detail': msg.event_already_approved}, status=400)
response_already_deny = Response(
    data={'detail': msg.event_already_denied}, status=400)
if not event.exists():
    return Response(data={'detail': msg.not_found}, status=404)
event = event.first()
if event.status > 1:
    return response_already_approved
if event.rejected:
    return response_already_deny
try:
    user_data = user_serializer.UserDetail(user).data
    if user_role == 2: # hod
        event.hod_verified = False
        event.rejected = True
    if not event.create_timeline(level=1, user=user_data, msg=deny_message,
        status=TimeLineStatus.rejected):
        raise Exception
    event.save()

    return Response(data={'detail': msg.success})

if user_role == 3: # Dean

```

```

event.dean_verified = False

event.rejected = True

if not event.create_timeline(level=2, user=user_data, msg=deny_message,
status=TimeLineStatus.rejected):

raise Exception

event.save()

return Response(data={'detail': msg.success})


if user_role == 4: # VC

event.vc_verified = False

event.rejected = True

if not event.create_timeline(level=3, user=user_data, msg=deny_message,
status=TimeLineStatus.rejected):

raise Exception

event.save()

return Response(data={'detail': msg.success})

except:

return Response(data={'detail': msg.server_error}, status=500)


@api_view(['POST'])
@is_authenticated
@is_event_organizer
def upload_report(request, event_id):
    msg = message.report_upload
    event = Event.objects.get(id=event_id)
    if event.status < 3:
        return Response(data={'detail': msg.ongoing_event}, status=400)
    if event.status >= 5:

```

```

return Response(data={'detail': msg.report_approved}, status=400)

report_file = request.FILES.get('file')

try:
    user_data = user_serializer.UserDetail(request.user).data
    event.report = report_file
    event.clear_timeline()

    if not event.create_timeline(level=7, user=user_data,
                                status=TimeLineStatus.completed):
        raise Exception

    event.status = 4

    event.save()

    return Response(data={'detail': msg.success, 'link':
                          request.build_absolute_uri(event.report.url)})
except:
    return Response(data={'detail': msg.error})

```

```

@api_view(['POST'])
@is_authenticated
@is_event_organizer
def upload_certs(request, event_id):
    msg = message.cert_upload

    event = Event.objects.get(pk=event_id)

    if event.status < 5:
        return Response(data={'detail': msg.report_approval_required})

    # Retrieve all EventParticipant objects for the specified event
    event_participants = EventParticipant.objects.filter(event_id=event_id)

```

```

# Check if a zip file is provided in the request
if 'file' not in request.FILES:
    return Response(data={'detail': msg.file_not_found}, status=400)

zip_file = request.FILES['file']
if not zipfile.is_zipfile(zip_file):
    return Response(data={'detail': msg.invalid_file_type}, status=400)

student_certificate = {}
invalid_files = []
zip_ref = zipfile.ZipFile(zip_file, 'r')
certificate_names = zip_ref.namelist()

for cert in certificate_names:
    fileName, file_type = os.path.splitext(cert)
    if fileName.isnumeric():
        student_certificate[fileName] = [cert, zip_ref.read(cert)]
    else:
        invalid_files.append(fileName)

participants_to_update = []
for participant in event_participants:
    # Get the user ID for each participant
    college_id = participant.user.college_id

    # Check if the image exists in the zip file
    if participant.status == '3' and college_id in student_certificate:

```

```

# Update the certificate field and add to the update list
file_name = student_certificate[college_id][0]
image_data = student_certificate[college_id][1]

# Convert binary data to an in-memory image

participant.certificate.save(file_name, BytesIO(image_data))
participants_to_update.append(participant)

if len(participants_to_update) > 0:
    with transaction.atomic():
        EventParticipant.objects.bulk_update(
            participants_to_update, ['certificate'])
        if not event.create_timeline(level=9,
            user=user_serializer.UserDetail(request.user).data,
            msg=msg.certified_by.format(
                request.user.first_name, request.user.college_id),
            status=TimeLineStatus.completed):
            raise Exception
        event.status = 6
        event.save()
        return Response(data={'detail': msg.success, 'certified_quantity':
            len(participants_to_update),
            'invalid_files': invalid_files}, status=200)
    else:
        return Response(data={'detail': msg.no_certificates_found, 'invalid_files':
            invalid_files}, status=400)

```

```

@api_view(['GET'])
@is_authenticated
@is_event_organizer
def delete_certs(request, event_id):
    msg = message.delete_cert
    event = get_object_or_404(Event, pk=event_id)
    event_participants = EventParticipant.objects.filter(
        event_id=event_id, status='3')
    if event.status < 6:
        return Response(data={'detail': msg.no_cert_to_delete}, status=400)
    participants_to_update = []
    for participant in event_participants:
        participant.certificate = None
        participants_to_update.append(participant)
    if len(participants_to_update) > 0:
        with transaction.atomic():
            EventParticipant.objects.bulk_update(
                participants_to_update, ['certificate'])
    if not event.create_timeline(level=9,
        user=user_serializer.UserDetail(request.user).data,
        status=TimeLineStatus.not_visited):
        raise Exception
    event.save()

    return Response({'detail': msg.success}, status=200)

```

```

@api_view(['GET'])
@is_authenticated

```

```

def apply_event(request, event_id):
    msg = message.apply_event
    response = Response({'detail': msg.success})
    try:
        event = Event.objects.get(pk=event_id)
        if event.status == 1:
            return Response(data={'detail': msg.pending_event}, status=400)
        if event.status > 2:
            return Response(data={'detail': msg.completed_event}, status=400)
        status, event_message = event.register_participant(user=request.user)
        if status is True:
            response.data['detail'] = event_message
            response.status_code = 200
        else:
            response.status_code = 403
            response.data['detail'] = event_message
    except:
        response = Response(status=404, data={'detail': msg.not_found})
    return response

```

```

@api_view(['GET'])
@is_authenticated
def club_branch(request):
    club = serializers.ClubSerializer(Club.objects.all(), many=True)
    branch = BranchSerializer(Branch.objects.all(), many=True)
    return Response({'club': club.data, 'branch': branch.data})

```

```

@api_view(['GET'])
@is_authenticated
@is_event_owner
def delete_event(request, event_id):
    msg = message.delete_event
    event = get_object_or_404(Event, pk=event_id)
    if event.status >= 3:
        return Response({'detail': msg.already_completed}, status=400)
    title = event.title
    event.delete()
    return Response({'detail': msg.success, 'status': 200})

```

```

@api_view(['GET'])
@is_authenticated
@is_event_organizer
def delete_report(request, event_id):
    msg = message.delete_report
    event = get_object_or_404(Event, pk=event_id)
    if event.status >= 5:
        return Response({'detail': msg.already_approved}, status=400)
    file_name = event.report
    event.clear_timeline()
    if not event.create_timeline(level=7, user=None,
        status=TimeLineStatus.ongoing) and event.create_timeline(level=8, user=None,
        status=TimeLineStatus.not_visited):
        raise Exception

```

```
event.report = None
event.status = 3

event.save()
return Response({'detail': msg.success, 'status': 200})
```

```
@api_view(['GET'])
@required_roles([4])
def deny_report(request, event_id):
    msg = message.reject_report
    event = get_object_or_404(Event, pk=event_id)
    if not event.report:
        return Response(data={'detail': message.approve_report.report_not_submitted},
            status=400)
    if event.status >= 5:
        return Response(data={'detail': message.approve_report.report_already_approved},
            status=400)

    if not event.create_timeline(
        level=8, user=user_serializer.UserDetail(request.user).data,
        msg=msg.report_reject_message.format(
            request.user.first_name, request.user.college_id),
        status=TimeLineStatus.rejected):
        return Response(data={'detail': 'Something Went Wrong'}, status=200)
    event.report = None
    event.status = 3
    event.save()
```

```
return Response(data={'detail': msg.success}, status=200)
```

```
@api_view(['GET'])
```

```
@required_roles([4])
```

```
def approve_report(request, event_id):
```

```
    msg = message.approve_report
```

```
    event = get_object_or_404(Event, pk=event_id)
```

```
    if not event.report:
```

```
        return Response(data={'detail': msg.report_not_submitted}, status=400)
```

```
    if event.status >= 5:
```

```
        return Response(data={'detail': msg.report_already_approved}, status=400)
```

```
    if not event.create_timeline(
```

```
        level=8, user=user_serializer.UserDetail(request.user).data,
```

```
        status=TimeLineStatus.completed):
```

```
        raise Exception
```

```
    event.status = 5
```

```
    event.save()
```

```
    return Response(data={'detail': msg.success}, status=200)
```

B. SCREENSHOTS

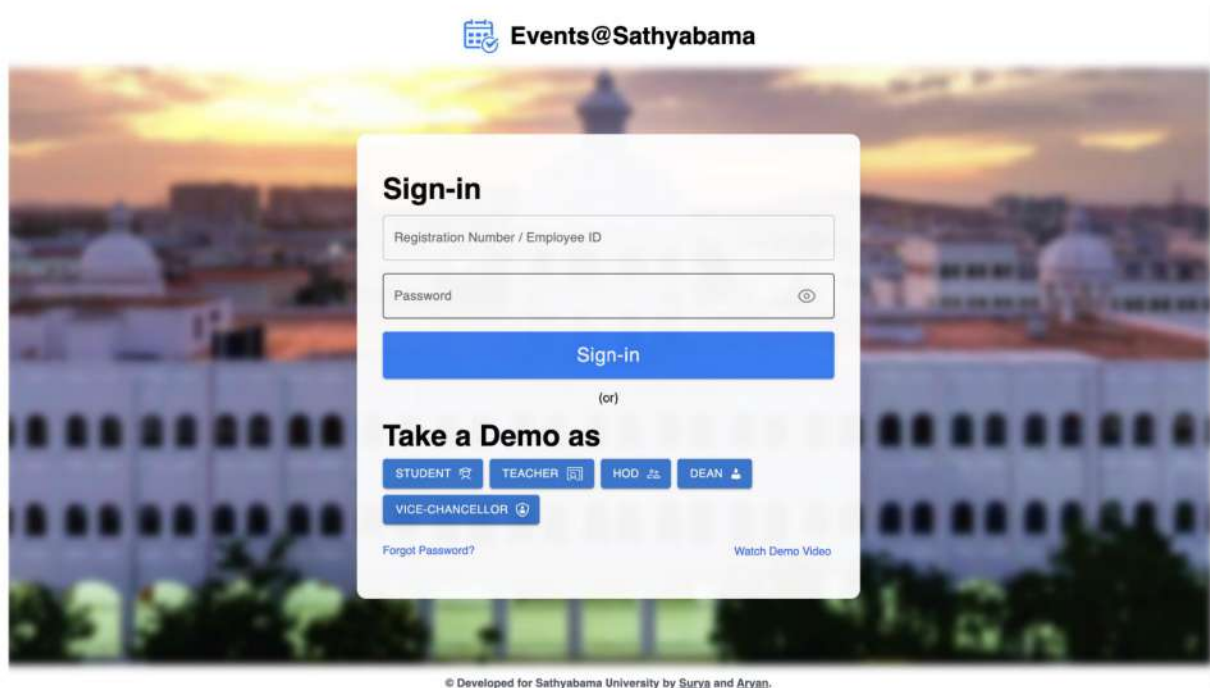


Figure B.1 Portal Landing Page

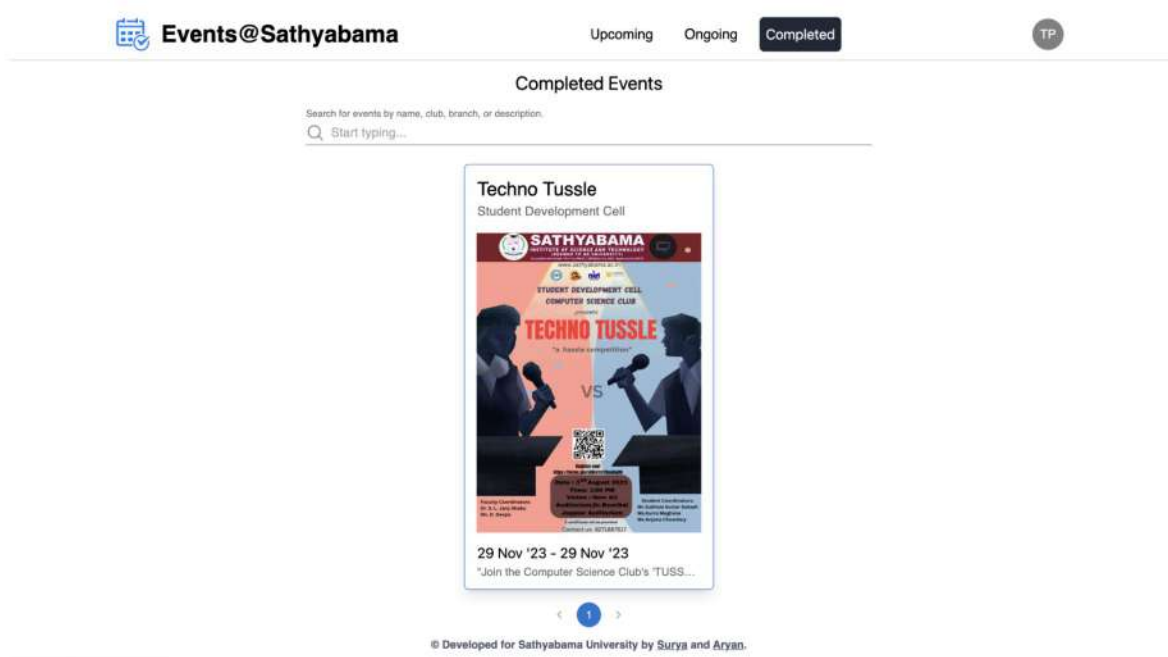


Figure B.2 Portal Home Page

Events@Sathyabama

Upcoming
Ongoing
Completed

TP

Techno Tussle

Student Development Cell

"Join the Computer Science Club's 'TUSSE,' a hassle competition.

29th November 2023

Single Day Event

Remibai Jeppiar Auditorium

"Sathyabama Institute of Science and Technology's Student Development Cell, in collaboration with the Computer Science Club, invites you to 'TUSSE'—a hassle competition poised to test your mettle and skills. This competitive event, scheduled for August 3rd, 2023, at 1:00 PM in the New AC Auditorium, promises an engaging platform for participants to showcase their abilities and compete amidst a vibrant atmosphere.

Hosted by esteemed faculty coordinators Dr. Remibai, Dr. S. L. Jany Shabu, and Ms. D. Deepa, 'TUSSE' aims to challenge and bring out the best in participants. This competition will feature thought-provoking tasks designed to assess problem-solving capabilities, technical expertise, and innovation.

Our dedicated student coordinators, Mr. Subham Kumar Aakash, Ms. Kurra Meghana, and Ms. Anjana Chowdary, are available to assist and ensure a seamless experience throughout the event.

TP

Teacher Persona

Faculty Coordinator

0/50 applicants
0%

© Developed for Sathyabama University by Surya and Aryan.

Figure B.3 Event Details Page

X
Event Admin Panel - Techno Tussle

Timeline
Event Actions
Applications
Upload Report
Upload Certificates

Event Created - Nov 30, 2023 11:01 AM

TP
Teacher Persona

Approved by the Head of Department - Nov 30, 2023 11:01 AM

HP
HOD Persona

Approved by the Dean - Nov 30, 2023 11:02 AM

DP
Dean Persona

Approved by the Vice-Chancellor - Nov 30, 2023 11:02 AM

VP
VC Persona

Displayed to Students - Nov 30, 2023 11:02 AM

ET
Events@Sathyabama Team

Event Ongoing - Nov 30, 2023 11:02 AM

ET
Events@Sathyabama Team

Event Completed - Nov 30, 2023 11:02 AM

ET
Events@Sathyabama Team

Figure B.4 Approval Process

Events@Sathyabama

Upcoming
Ongoing
Completed

TP

Create / Update your Event

The asterisk (*) indicates a required field. These fields are mandatory to be filled out.

Event's Name Here *

Please enter your event's name in this field. It will be displayed on both the homepage cards and your event's details page.

Club / Organiser's Name Here *

Please enter your club or organiser's name in this field. This will reflect on both the homepage cards and your event's details page.

UPLOAD YOUR EVENT'S POSTER HERE *

The Event's Poster will be displayed on both the homepage cards and your event's details page.

Your Event's Start Date *

Your Event's End Date *

DD/MM/YY

DD/MM/YY

If you are unsure about your event's date, please specify an approximate date. You can edit/update this later. If it is a single-day event, please specify the same date for the start and end date.

Short Description Here *

This will be reflected on the homepage cards and the header of your event's details page. (Maximum 96 characters)

Long Description Here

Please provide a detailed explanation of your event in this field. Include any important details or context about your event here.

Event's Branch and Batch Selection

Preview

Event's Name Here

Organiser/Club's Name Here

Event's Date Here

Short Description Here

Figure B.5 Event Creation Page

Events@Sathyabama

Upcoming
Ongoing
Completed

TP

TP

Teacher Persona

Role
Teacher

EID
1

Branch
IT

Organising
Completed

Virtual Mock Parliament (Displayed)

School of Law

Awaiting event completion...

Envi Topia (Displayed)

Student Development Cell

Awaiting the ongoing occurrence of the event...

Logo Heist (Displayed)

Student Development Cell

Awaiting the ongoing occurrence of the event...

© Developed for Sathyabama University by Surya and Aryan.

Figure B.6 User Profile Page

Evaluation of Rendering Strategies: Next.js SSR vs React.js CSR Performance Analysis

Bandepalli Surya Anjani Kumar
B.E, Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, India
bsuryakumar03@gmail.com

A. Mary Posonia
Associate Professor, Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, India
maryposonia.cse@sathyabama.ac.in

Aryan Amish
B.E, Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, India
aryanamish385@gmail.com

S. Revathy
Associate Professor, Computer Science and Engineering
Sathyabama Institute of Science and Technology
Chennai, India
revathy.it@sathyabama.ac.in

Abstract— React.js is a majorly adapted library in the current age of web development. Next.js, built on top of React also has gained massive popularity among the community of developers. The paper focuses on giving the developers a comparative study to decide for their next web application between these two contenders where React.js uses CSR and Next.js uses SSR. Rendering strategies were compared with metrics such as Total Blocking Time, First Contentful Paint, Largest Contentful Paint, and Speed Index which help measure their web page performance.

The awareness of these metrics concerning the number of elements rendered helps developers make factual decisions concerning user experience, scalability, and overall application performance.

Keywords: *Next.js, React.js, Client-Side Rendering, Server-Side Rendering, Performance Analysis, Largest Contentful Paint, First Contentful Paint, User Experience, Speed Index, Total Blocking Time, Scalability, Application Performance.*

1. INTRODUCTION

A major factor in a web app's performance is the rendering approach. It helps in better user experience and satisfaction. SSR delivers ready-to-use HTML directly to the client. CSR, on the other hand, focuses on rendering after receiving the data from the server.

The pros for each approach may be seen in their speed, interactivity and visual completeness. The research paper focuses on conducting a comparison between these approaches with different element counts to be rendered in various device specifications. The aim is to provide an insight for developers and decision makers for their next web application.

2. LITERATURE REVIEW

Web development frameworks have helped with the continuous progress of both SSR and CSR. Pratik Sharad Maratkar and Pratibha Adkar (2021) [1] have touched upon how React, a JavaScript library, has enhanced UI development. The component-based architecture and virtual DOM utilization have improved performance without frequent re-rendering of the actual DOM.

Johannes Stein's "ReactJS Cookbook" [2] (2017) and Eric Masiello's "Mastering React Native" [3] (2017) highlight React's adaptability and versatility in modern web development.

Adam Horton and Ryan Vice's "Mastering React" [4] (2016) have mentioned how the coordination between React and Redux has simplified state management in UI frameworks.

In Stoyan Stefanov's book titled "React; Up and Running; Building Web Applications" (2016) [5] there is a discussion about the impact of the Chrome V8 engine that was introduced by Google in 2008. This engine brought about a change in JavaScript execution speed exceeding browsers like Internet Explorer and acts as a foundation for JavaScript platforms like Node.js 6.

In a study published in 2016 by Dasari Hermitha Curie, Jaison Joyce, Yadav Jyoti and their colleagues titled "Investigation, into the Impact of Web Frameworks," [6] the effects of JavaScript frameworks like Next.js and React.js on the field of web development were highlighted. Next.js building upon React has addressed its client-side constraints of (Masiello, 2017) [3].

The World Wide Web Consortium has continuously improved the standard of web development with the introduction of APIs and semantic elements via the HTML5 specifications (W3C, 2014).

3. SUMMARY

The advancements in JavaScript from the introduction of the V8 engine which led to powerful and component-based frameworks such as React.js and Next.js which was built on top of it have demonstrated a positive slope. They highly improved performance, state management, and faster load time on slower devices giving them the same experience as a desktop user and search engine optimization in web applications. But, they have introduced their own set of challenges which require developers and decision makers to be informed of the shortcomings of these advancements.

4. FRAMEWORK INSPECTION

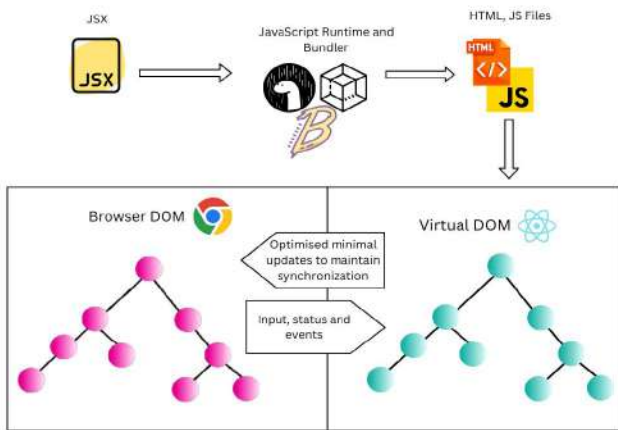


Fig. 1. Flowchart illustrating the React rendering process

React.js uses a technique for minimal updates to maintain concurrence. The flow begins with JSX, a syntax extension for JavaScript, which is transpiled into JavaScript. The JavaScript runtime and bundler then process the JavaScript code, along with other necessary files such as HTML and additional JavaScript files. The React.js framework employs a Virtual DOM to optimize rendering efficiency. When input, status changes, or events occur, React calculates the minimal updates needed and efficiently reconciles these changes with the Virtual DOM. Finally, the optimized updates are applied to the Browser DOM, resulting in synchronized and minimal changes to the rendered HTML. This streamlined process ensures efficient and performant rendering in React.js, enhancing the user experience by minimizing unnecessary updates to the DOM. This entire concept is illustrated in Fig.1.

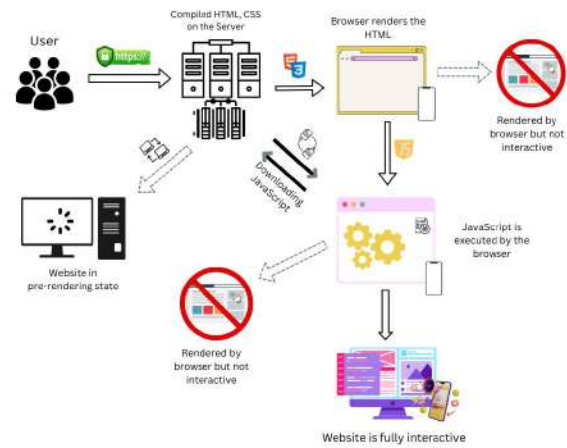


Fig. 2. Flowchart illustrating Next.js rendering algorithm

In the context of NextJS framework inspection, the user initiates the website interaction by entering a URL. Leveraging NextJS's pre-rendering capabilities, the website exists in a pre-rendered state, with HTML and CSS compiled on the server side. The initial rendering occurs as the browser receives the pre-rendered HTML and CSS, making the content visible but non-interactive. The subsequent execution of JavaScript by the browser propels the website into full interactivity, enabling dynamic updates and client-side rendering. The flowchart in Fig.2. underscores NextJS's proficiency in combining server-side rendering for quick initial load times with client-side rendering for a responsive and interactive user experience, embodying a holistic approach to web development.

5. METHODOLOGY FOR ANALYSIS

The methodology implemented for the comparative analysis between React and Next.js involved the development of two similar applications featuring an equivalent number of components and functionalities. Each application was separately built using React and Next.js frameworks, maintaining uniformity in the structure, behavior, and complexity. The flowchart provided reflects a section of the applications created for the study.

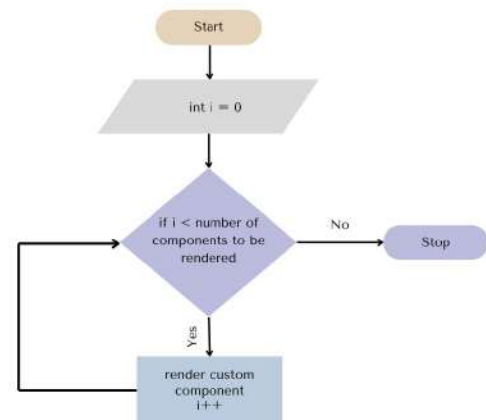


Fig. 3. Flowchart illustrating the Rendering Algorithm

The comparison focused on evaluating the performance of SSR in Next.js and CSR in React. A significant number of

components until 1,00,000 paragraphs were introduced to replicate a real-world scenario.

The metrics were audited with a performance analyzer tool named Lighthouse, which is open-source and present in Chromium browsers. The metrics monitored were First Contentful Paint, Largest Contentful Paint, Total Blocking Time and Speed Index.

The tests were performed on identical hardware, software and network conditions. Insights such as rendering speed, interactivity and performance data were extracted for both the React.js and Next.js application modules which helped us to complete and summarize this investigation.

6. RESULTS AND DISCUSSION

First Contentful Paint (FCP) - Desktop and Mobile:
FCP or first contentful paint analyzes the first visuals that the user sees, in FCP CSR consistently maintains a 200ms duration because all the rendering happens on the client side and this duration only depends on the file size. Conversely, SSR shows potential delays in its FCP because of dynamic content generation. This indicates that SSR's FCP duration can be longer because of server-side processing.

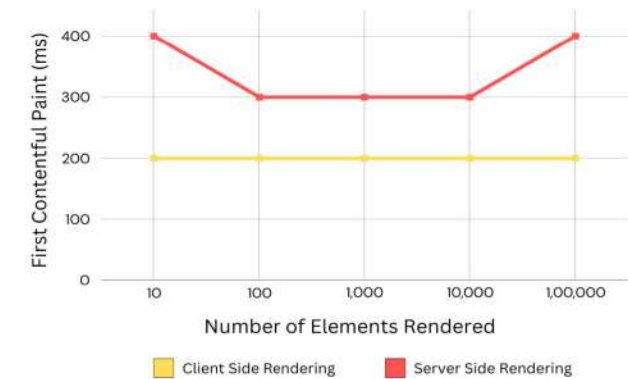


Fig. 6. First Contentful Paint (Desktop)

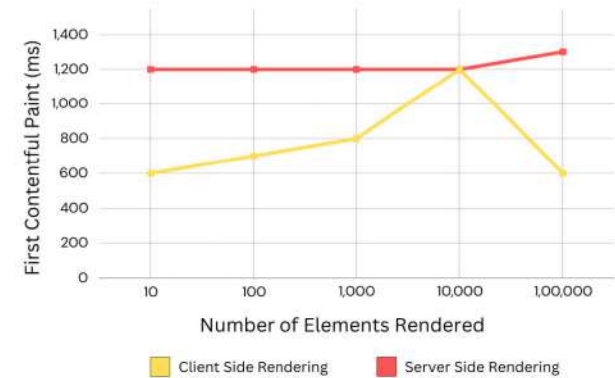


Fig. 7. First Contentful Paint (Mobile)

Fig.6 and Fig.7 indicate the First Contentful Paint on desktop and mobile respectively. The graphs indicate the

faster nature of CSR, while the slower behavior of SSR due to its server-side preprocessing.

Largest Contentful Paint (LCP) - Desktop and Mobile:
In Largest Contentful Paint (LCP), SSR outperforms CSR consistently across various element counts on desktop as well as on mobile. The outcome showcases the SSR's ability to render larger content swiftly, making a smoother user experience compared to CSR.

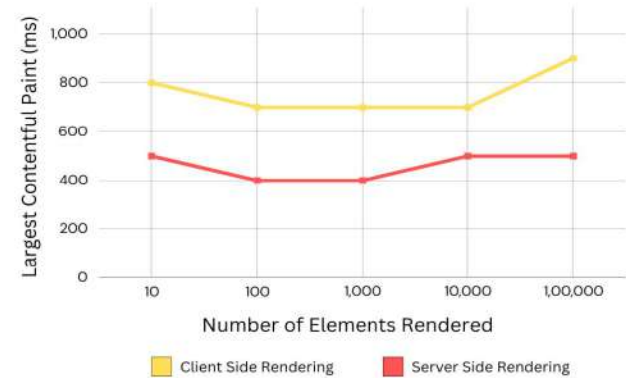


Fig. 8. Largest Contentful Paint (Desktop)

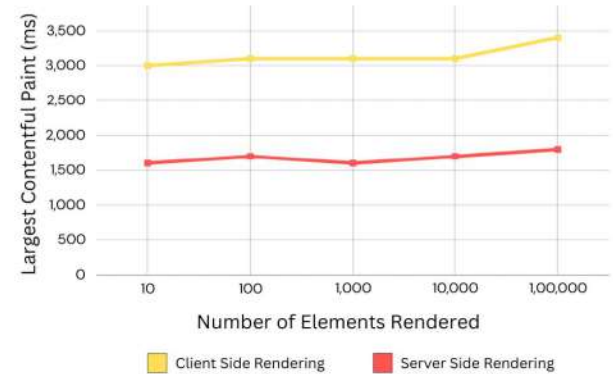


Fig. 9. Largest Contentful Paint (Mobile)

In Fig.8 we observe the extraordinary performance of SSR in doing Largest Contentful Paint on desktop, likewise we can see the same pattern for Mobile performance in Fig.9. This data shows that SSR excels in handling the largest content, making for a better User experience across both desktop and mobile platforms.

Total Blocking Time - Desktop and Mobile: Analysis of the Total Blocking Time is the time period when the site is unresponsive because the JavaScript is executing something in the background. Since SSRs handle most of the rendering on the server there is less work to do for Javascript on the client side making it faster and more interactive. Figures (Fig.10 and Fig.11) showcases the insights into the performance of Client-Side Rendering and Server-Side Rendering.

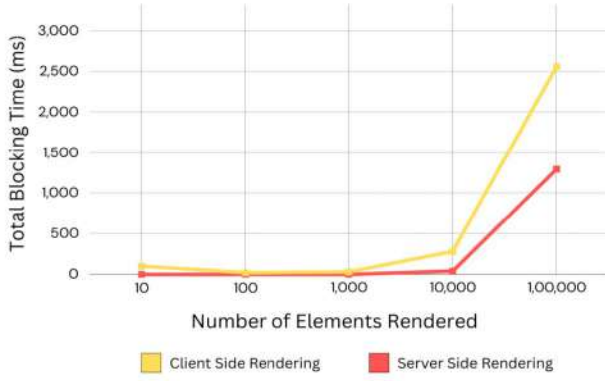


Fig. 10. Total Blocking Time (Desktop)

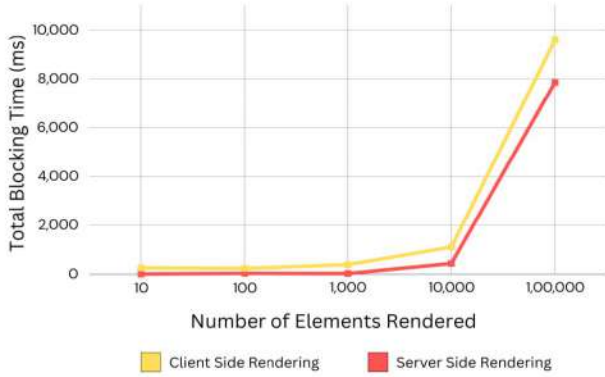


Fig. 11. Total Blocking Time (Mobile)

Speed Index - Desktop and Mobile: The evaluation of the Speed Index underscores SSR's superior performance over CSR in both desktop and mobile settings as the number of elements increases. This emphasizes SSR's efficiency in delivering quicker rendering and page load times compared to CSR, thus enhancing overall user-perceived performance.

$$\text{Speed Index} = \text{Interval time} \times \left(1 - \frac{\text{visual completeness}}{100}\right)$$

Time is represented in intervals of 500 milliseconds and Visual Completeness is the percentage of total above-the-fold content that has been loaded. Fig.12 and Fig.13 highlight Server-Side Rendering (SSR) extraordinary performance over CSR in both desktop and mobile platforms; it indicates that SSR increases the user experience by minimizing the time it takes to load the site on the browser.

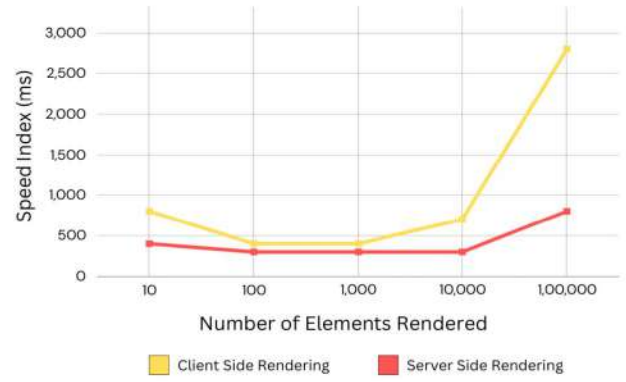


Fig. 12. Speed Index (Desktop)

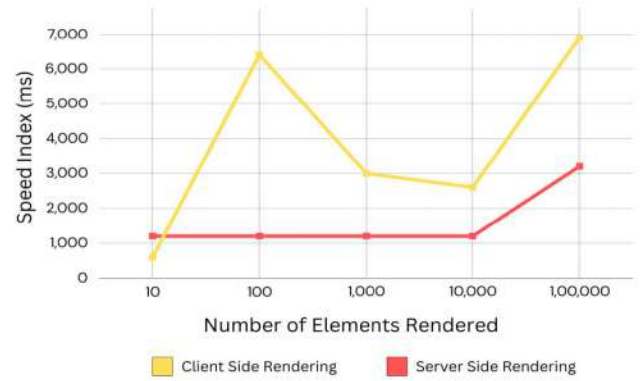


Fig. 13. Speed Index (Mobile)

Performance across Devices: The comparison between React.js (CSR) and Next.js (SSR) showcases initial performance parity until reaching 10^2 elements, after which SSR notably outperforms CSR. This distinct advantage of SSR scaling with a larger volume of elements highlights its enhanced scalability and rendering efficiency in both desktop and mobile environments. SSR consistently demonstrates superior efficiency across varying element quantities from 10^0 to 10^4 in mobile environments, positioning it favorably for mobile applications.

The performance of the webpage is calculated using the below formula.

$$\frac{1}{10} \left[FCP + SI + \left[\frac{5}{2}(LCP + CLS) + 3TBT \right] \right]$$

FCP = First Contentful Paint

SI = Speed Index

LCP = Largest Contentful Paint

CLS = Cumulative Layout Shift

TBT = Total Blocking Time

In Fig.4, the performance of CSR and SSR in desktop environments is illustrated. Simultaneously, Fig.5 provides a visual representation of the performance on mobile devices.

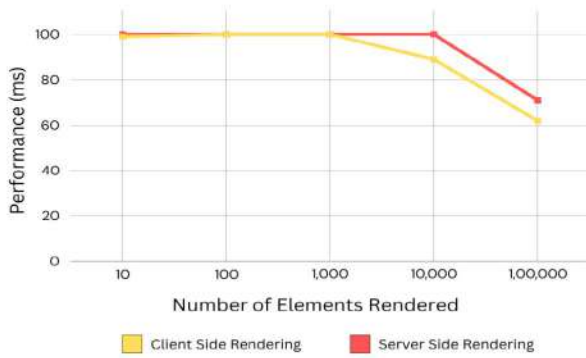


Fig. 4. SSR and CSR Performance (Desktop)

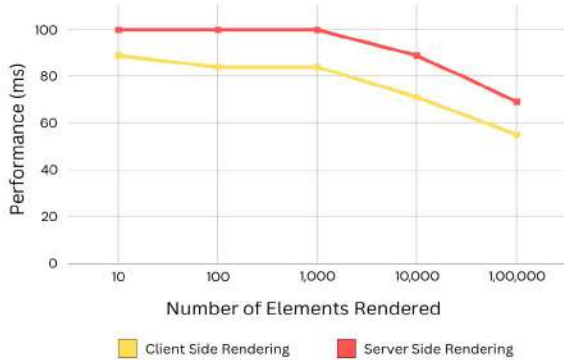


Fig. 5. SSR and CSR Performance (Mobile)

Metric	NextJS Desktop	React Desktop	NextJS Mobile	React Mobile
Avg Performance	94.2	90	91.6	76.6
Average FCP	0.34s	0.2s	1.2s	0.78s
Average LCP	0.46s	0.76s	1.68s	3.14s
Average TBT	266ms	598ms	1676ms	2318ms
Average Speed Index	0.42s	1.02s	1.6s	3.9s

Table. 1. Framework Performance Summary

Table.1 depicts that the average performance index indicates NextJS outperforms React on both desktop (94.2 vs. 90) and mobile (91.6 vs. 76.6). Notably, NextJS demonstrates faster loading times with lower Average *FCP* and *LCP* values for both desktop and mobile. In terms of responsiveness, NextJS

showcase lower TBT whereas React shows higher TBT values, this indicates the speed advantage of server-side rendering solutions.

7. CONCLUSION

The Study compared React.js which uses Client-side rendering with Next.js which uses server-side rendering across multiple matrices such as number of elements and on different device specifications.

Through analyzing key performance indicators such as Speed Index, Largest Contentful Paint and Total Blocking Time. SSR showcases better performance than CSR as the number of elements increases, SSR also decreases the Total Blocking time giving a better user experience.

These findings enforces SSR's claims in reducing rendering delays and making the site load faster by doing the heavy lifting on the server side rather than the client side, SSR's performs exceptionally well when rendering a higher number of elements and on slower devices such as mobile devices.

8. REFERENCES

- [1] Pratik Sharad Maratkar, Pratibha Adkar, "React JS - An Emerging Frontend JavaScript Library", Iconic Research And Engineering Journals, Vol. 4, Issue 12-2021, pp. 99-102.
- [2] Stein Johannes, "ReactJS Cookbook", December 6, 2017, pp. 150-170
- [3] Masiello Eric, "Mastering React Native", January 11, 2017.
- [4] Horton Adam. Vice Ryan, "Mastering React", February 23, 2016.
- [5] Stefanov. S, "React: Up and Running: Building Web Applications", 14 July, 2016.
- [6] Dasari Hermitha Curie et al, "Analysis on Web Frameworks", November 17, 2019.
- [7] Harish A Jartarghar, Girish Rao Salanke, Ashok Kumar A.R, Sharvani G.S and Shivakumar Dalali, "React Apps with Server-Side Rendering: Next.js", Journal of Telecommunication, Electronic and Computer Engineering, Vol. 14, 30 December, 2022.
- [8] Mochammad Fariz Syah, Dyah Anggraini, "Modern Front End Web Architectures with React.Js and Next.Js", International Research Journal of Advanced Engineering and Science: 2455-9024.
- [9] <https://nextjs.org/docs>
- [10] <https://react.dev/learn>



SATHYABAMA



INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

CATEGORY-1 UNIVERSITY BY UGC

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE



www.sathyabama.ac.in

SCHOOL OF COMPUTING

Department of Computer Science and Engineering

Certificate of Presentation

This is to certify that

Bandepalli Surya Anjan Kumar

of

Sathyabama Institute of Science and Technology

has presented a paper entitled

Evolution of Rendering Strategies

in the

International Conference on Cognitive Computing and Artificial Intelligence (ICCCAI-2024) in association with
Taylor's University, Malaysia held on 7th & 8th March 2024.

Dr. Sayan Kumar Ray
Head, School of Computer Science
Taylor's University

Dr. L. Lakshmanan
HOD, CSE Dept
Sathyabama Institute of Science and Technology

Dr. F. Sasikala
Dean, SOC
Sathyabama Institute of Science and Technology



SATHYABAMA



INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

CATEGORY-I UNIVERSITY BY UGC

Accredited 'A++' Grade by NAAC B2B Status by UGC Approved by AICTE



www.sathyabama.ac.in

SCHOOL OF COMPUTING

Department of Computer Science and Engineering

Certificate of Presentation

This is to certify that

Aryan Amish
of

Sathyabama Institute of Science & Technology
has presented a paper entitled

Evaluation of Learning Strategies
In the

International Conference on Cognitive Computing and Artificial Intelligence (ICCCA1-2024) in association with
Taylor's University, Malaysia held on 7th & 8th March 2024.

Dr. Sayan Kumar Ray
Head, School of Computer Science
Taylor's University

Dr. L. Lakshmanan
HOD, CSE Dept
Sathyabama Institute of Science and Technology

Dr. T. Sastikumar
Dean, SCET
Sathyabama Institute of Science and Technology