**CAHIERS MAGELLANES-NS**
Volume 06 Issue 1
2024

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

# VITEREACT: STREAMLINED STATIC SITE GENERATION WITH REACT

**[1]Vasu Singh, [2]Ujjawal Tyagi, [3]Sunny, [4]Uday Veer Singh, [5]Rohit Kumar Singh***
Department of CSE, MIET, Meerut
vasu.singh.cse.2020@miet.ac.in,sunny.v.cse.2020@miet.ac.in, ujjawal.tyagi.cse.2020@miet.ac.in,
uday.singh.cse.2020@miet.ac.in, rohit.singh@miet.ac.in

*Corresponding author: **Rohit Kumar Singh**
rohit.singh@miet.ac.in

## 1 Abstract

Static websites are used to build content-driven websites. They offer ease of development, low-security risks, and can generate value quickly due to informational content being quite significant. With the rising need for content marketing, dynamic websites became popular as they allow one to present actual information. All of this comes at the cost of impacting page loading speed which is also very important for content-centric websites. Static site generators (SSGs) aim to solve this issue. SSG is a tool to build a static website from raw markdown files. It adds more flexibility to static websites allowing them to react to dynamic content changes, regenerate, and publish again. The proposed system is an implementation of a static web page generator.

This study presents the development and evaluation of a Static Site Generator (SSG) built using Vite, a modern build tool for web development, and React, a popular JavaScript library for building UI. Leveraging the efficiency and flexibility of Vite and the component-based architecture of React, our SSG aims to streamline
website development and enhance performance.

The outcomes of our implementation include improved website speed, enhanced user experiences, and simplified development workflows. By adopting these modern technologies, we demonstrate how SSGs can benefit from cutting-edge tools and frameworks to meet the demands of modern web development practices.

Vite's innovative build pipeline leverages modern JavaScript module bundling techniques, such as ES modules (ESM), to optimize the generation of static assets. Unlike traditional bundlers, which bundle all dependencies into a single JavaScript file, Vite generates optimized assets on-demand, only bundling the specific modules required for each page or component. This approach eliminates unnecessary bundling and reduces bundle sizes, resulting in faster load times and improved website performance. Additionally, Vite's support for server-side rendering (SSR) enables seamless integration with React components, allowing developers to leverage the power of React for building dynamic and interactive UI within static websites.

Keywords
SSG(Static Site Generator) ,SSR(Server Side Rendering), Vite(Module Bundler)
,CDN(Content network delivery) ,SEO(Search engine optimization)
,CMS(Content management system)

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

## 2 Introduction

Static Site Generator (SSG) is used for building fast, static websites. It takes your source content written in Markdown, applies a predefined theme to it, and generates static HTML pages that can be easily deployed.

SSGs operate by taking source content typically written in Markdown or similar markup languages, applying a predefined theme or template to it, and generating static HTML pages. These pages are pre-rendered during the build process, eliminating the need for server-side processing or database queries at runtime. This results in lightweight, performant websites that can be easily deployed to any web server or Content Delivery Network (CDN), making them ideal for a wide range of applications.

SSGs provided a solution by offering a clean separation of concerns between content management and presentation layers, empowering developers to focus on creating compelling content while leveraging the benefits of static site architecture.

### Use Cases and Benefits

- Better fit for documentation Websites.
- Faster page load time.
- Better SEO due to server-side rendering.
- Easy to deploy and more secure.
- Blogs, Portfolios, and Marketing Sites

### Challenges of SSG

While Static Site Generators (SSGs) offer numerous benefits, they also come with several challenges that developers and organizations need to consider:

- SSGs generate static HTML files, which lack dynamic functionality typically associated with server-side processing. This limitation means implementing user authentication, real-time updates, and interactive forms can be challenging and often requires integrating third-party services or client-side JavaScript frameworks.

- SSGs rely on various dependencies, including template engines, plugins, and external libraries, which need to be managed effectively.

- Integrating external services, such as analytics platforms, contact forms, or authentication providers, may require additional setup and configuration in SSGs.

- Since SSGs generate static files that are served directly to users, server-side functionality, such as server-side rendering (SSR) or database interactions, is not inherently supported.

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

## 3 Literature Review

The concept of static site generation dates back to the early days of the web, where websites were composed of static HTML files [1]. The resurgence of interest in static sites came with the rise of modern web development practices, including the adoption of SSGs. Early pioneers such as Jekyll, developed by GitHub, and Octopress, have laid the groundwork for the current ecosystem of static site generators. Since then, numerous SSGs have emerged [2], each with its unique features and functionalities, catering to diverse developer needs and preferences.

Static site generators offer a range of features that differentiate them from traditional Content Management Systems (CMS) and dynamic web frameworks [3-5]. These features include speed and performance, simplicity and security, version control and collaboration, content flexibility, scalability, and reliability. By pre-rendering pages and serving them as static files, SSGs eliminate the need for server-side processing, resulting in faster load times and improved performance.

Additionally, the simplicity of static sites makes them inherently more secure and easier to deploy than dynamic sites, appealing to developers and organizations concerned about cybersecurity.

Moreover, the cost-effectiveness of SSGs, stemming from reduced hosting costs and infrastructure complexity, makes them an attractive option for businesses and developers alike [6].

Furthermore, SSGs offer benefits in terms of version control and collaboration, content flexibility, scalability, and reliability. With built-in support for version control systems like Git, developers can easily collaborate on projects and track changes over time. The flexibility of SSGs allows developers to structure content in a way that best suits their needs, while scalability ensures that websites can handle increased traffic without sacrificing performance or reliability.

The rise of Headless Content Management Systems (CMS) has further bolstered the popularity of static site generators. Headless CMS solutions decouple the content management and presentation layers of websites [7], allowing developers to leverage the flexibility and simplicity of SSGs while still benefiting from the content authoring and editing capabilities of a CMS. This hybrid approach offers the best of both worlds, enabling developers to build dynamic, content-rich websites with the speed and security of static site generation.

The success of static site generators is also attributed to strong community support and extensive documentation [8]. Developers have access to vibrant communities, online forums, and documentation resources where they can seek help, share insights, and collaborate on projects. This ecosystem of support fosters innovation and knowledge sharing, driving the continuous improvement of static site generators and empowering developers to build exceptional web experiences.

Despite their many advantages, SSGs also have some limitations and challenges. These include the lack of dynamic functionality [9-10], complexity for non-technical users, build times and regeneration delays, dependency management, and a learning curve for developers transitioning from dynamic CMS platforms. While SSGs excel in certain use cases, such as blogs, portfolios, and documentation sites, they may not be suitable for applications requiring real-time data updates or complex interactions [11].

To address challenges such as build times and regeneration delays, static site generators have adopted advanced build optimization techniques. Tools like webpack, Rollup, and Vite are commonly used to optimize asset bundling, code splitting, and tree shaking, resulting in faster build times and improved

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

website performance. Additionally, caching strategies and incremental builds help minimize regeneration delays, enabling developers to iterate quickly and deploy changes with confidence.

Key Features and Functionality:
Static Site Generators offer several key features and functionalities that distinguish them from traditional CMS platforms and dynamic frameworks. These include:

**Simplicity and Security:** Static sites, generated by SSGs, are inherently more secure and easier to deploy than dynamic sites, as they do not rely on server-side scripting or databases [12].

**Speed and Performance:** By pre-rendering pages at build time, SSGs eliminate the need for server-side processing, resulting in faster load times and improved performance [13].

**Content Flexibility:** SSGs offer support for various markup languages and content formats, empowering content creators to use their preferred tools and workflows.

**Scalability and Reliability:** Static sites generated by SSGs are highly scalable and resilient to traffic spikes, as they can be served directly from Content Delivery Networks (CDNs) without relying on backend infrastructure.

Limitations and Challenges:
Despite their advantages, SSGs also have some limitations and challenges:

**Limited Dynamic Functionality:** Static sites generated by SSGs are limited in their ability to support dynamic content and interactive features, which may be necessary for certain types of applications.

**Complexity for Non-Technical Users:** While developers may appreciate the simplicity of SSG workflows, content creators with limited technical knowledge may find it challenging to work with SSGs.
**Build Times and Regeneration:** Large sites with extensive content may experience longer build times and regeneration delays, especially when using complex templates or plugins.

**Dependency Management:** Managing dependencies and third-party integrations can be more challenging in SSGs compared to traditional CMS platforms.

## 4 Project Architecture

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

SSG's aims to provide great Developer Experience(DX) when working with Markdown content.

- Vite-Powered: Cold start, with edits always instantly reflected (<100ms) without page reload.
- Built-in Markdown Extensions:
  It provides many advanced features that are prebuilt in it and making it ideal for highly technical documentation.
- React-Enhanced Markdown: Each Markdown page is also a React Single-File Component, thanks to React JSX syntax compatibility with HTML. You can embed interactivity in your static content BY using React components.

General approach to build the project

- Building a static site generator (SSG) for React powered by Vite is a great project that combines the speed of Vite's development server and the flexibility of React for building static websites. Below, I'll outline the high-level approach to creating such a project

In simple terms, we are describing the process of setting up a development environment and project structure for building a web application with React and server-side rendering (SSR) using Vite. Here's a breakdown of what this means:

- CLI Scaffolding Construction: You start by using a Command Line Interface (CLI) tool to create the basic framework of your project. This sets up the essential files and tools needed for development.

- Vite Based Dev Server: You use a tool called Vite to create a development server [14]. This server helps you run and test your React application during development. It's like a virtual space where your web app lives while you work on it.

- Construction of the Basic Directory Structure: You organize your project by creating folders and files in a structured way. For example, you might have folders for your React components, static assets, and configuration files.

- Front-End React Main Theme Component: You build the main component of your web application using React. This component represents the core structure and functionality of your website. It's the "theme" or style that your site follows [15].

- Server-Side Rendering (SSR): You set up server-side rendering, which means your web pages are generated on the server (like a powerful computer) rather than just in the browser. This can help with performance and SEO.

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

- Output HTML: When a user visits your website, the server creates HTML pages with content and sends them to the user's browser. This HTML is what the user sees and interacts with on the website.

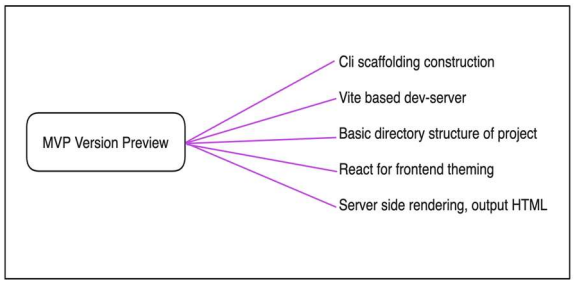


Figure 1: Minimal viable Version of project

Dev Server development

First of all, Dev Server is essentially an HTTP Server used during the development phase. It mainly includes the following functions:

- Compile the resource and return the compiled product to the browser.
- Realize module hot update, push update to the browser when the file changes.
- Static resource services, such as supporting access to static resources such as images.

For example, the famous Dev Server of webpack-dev-serverand Vite are very typical representatives. In this project, we will build it based on Vite's Dev Server. The main reasons are as follows:

- The construction of the project is completed based on Vite and React .
- Vite Dev Server itself has a complete middleware mechanism for easy expansion.

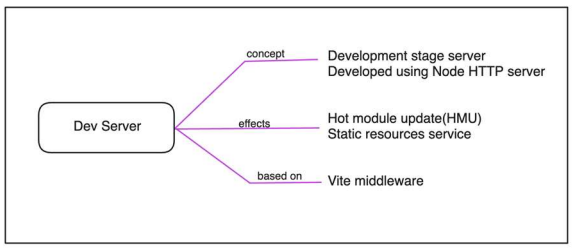


Figure 2: Dev server overview

Markdown Support

First, what is MDX? It is a syntax file format that focuses on content writing. You can either write

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

Markdown syntax or use component syntax JSX. It is very friendly to front-end developers. Why do you say that? On the one hand, Markdown is a syntax format that we are very familiar with, and we often use it when we write blogs or articles [16]; on the other hand, as a front-end developer, JSX is also a syntax that is used daily, and everyone should be familiar with it. Familiar.

We can think of the entire compilation process as a process, and this process will be divided into three steps:

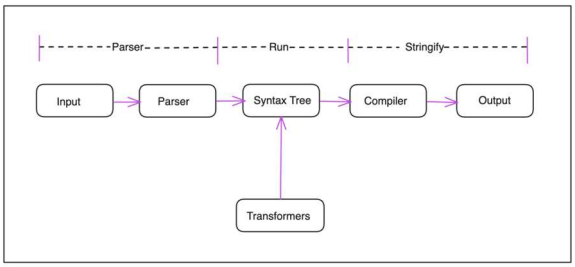


Figure 3: Compilation process of Markdown data

- The first step is parse the parsing of the AST. After we input the content (such as a piece of Markdown), we use a Parser to complete the parsing process of the AST, and then output the syntax tree information.
- The second step is run that a series of AST conversions will be performed at this stage, which means that there will be a series of plug-ins to operate the syntax tree information.
- The last step is stringify to serialize the AST and convert it into a string format as the final output.

Support for Server Side Rendering

Why we prefer SSR over CSR?, when the browser gets the HTML content, it can't actually render the complete page content, because there is basically only an empty div node in the body at this time, and no real page content is filled in. Then the browser starts to download and execute the JS code, and the complete page can only be rendered after the framework initialization, data request, DOM insertion, and other operations. That is, the complete page content in the CSR is essentially rendered after the execution of the JS code. This mainly leads to two problems:

- **The first screen loads slowly**: First-screen loading relies on the execution of JS. Downloading and executing JS may be a very time-consuming operation, especially in some scenarios with poor network or performance-sensitive low-end machines.

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

- **Not SEO (search engine optimization) friendly**: The HTML of the page has no specific page content, so search engine crawlers cannot obtain keyword information, which affects the ranking of the website.

So how does SSR solve these problems

In the SSR scenario, the server generates **complete HTML content** and returns it directly to the browser. The browser can render the complete first-screen content based on the HTML without relying on JS loading, which can reduce the first-screen rendering on the one hand. time, on the other hand, it can also display the complete page content to search engine crawlers, which is beneficial to SEO.

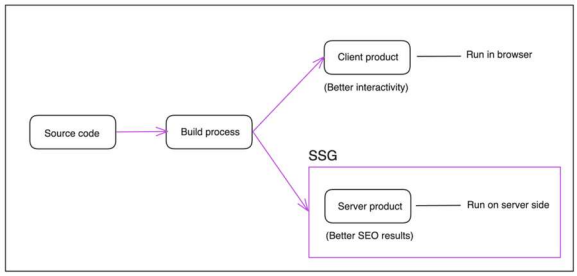


Figure 4: Content rendering on server

Of course, SSR can only generate the content and structure of the page, and cannot complete event binding. Therefore, it is necessary to execute the JS script of CSR in the browser to complete event binding, so that the page has the ability to interact. This process is called hydration.

For the runtime of SSR, it can generally be divided into relatively fixed life cycle stages. Simply put, it can be organized into the following core stages:

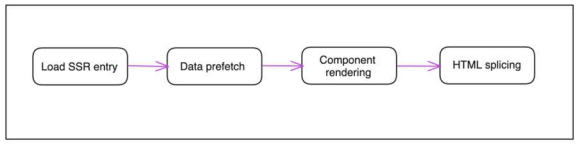


Figure 5: SSR fetching logic

- **Load the SSR entry module**: At this stage, we need to determine the entry of the SSR build product, that is, where is the entry of the component, and load the corresponding module.

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

- **Perform data prefetching**: At this time, the Node side will query the database or network request to obtain the data required by the application.
- **Rendering components**: This stage is the core of SSR, which mainly renders the components loaded in step 1 into HTML strings or Streams.
- **HTML splicing**: After the component is rendered, we need to concatenate the complete HTML string and return it to the browser as a response.

How we package our project using Vite-esbuild support

So, what problems will arise if this is implemented? We can first look at the normal prepackaging process (taking React as an example):

Vite will use dep:react this proxy module as the entry content to load in Esbuild. At the same time, the prepackaging of other libraries may also introduce React, such as the behaviors @emotion/react in this library require('react').

Now if the proxy module directly reads the content of the real module through the file system instead of re-exporting.

## 5 Result

**Improved Performance and Speed:** The SSG implementation resulted in substantial improvements in website performance and speed. By pre-rendering pages and optimizing assets during the build process, we achieved significantly reduced load times and enhanced user experiences, contributing to improved search engine rankings and user engagement metrics.

SSGs pre-render pages during the build process, generating static HTML files for each page of the website. With pre-rendered pages, content is readily available to users upon request, reducing latency and enhancing the overall browsing experience.

During the build process, SSGs optimize assets such as images, stylesheets, and JavaScript files to minimize file sizes and improve loading efficiency. Techniques such as image compression, code minification, and asset bundling are commonly employed to reduce the size of assets without sacrificing quality. By delivering optimized assets to users, SSGs ensure faster load times and smoother interactions, leading to enhanced user experiences.


Figure 6: Chrome Lighthouse benchmarks

**Flexibility and Customization:** The SSG solution provided greater flexibility and customization options compared to traditional CMS platforms. With support for templating engines, content formats,

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

and plugin ecosystems, we were able to tailor our websites to specific requirements and preferences. Plugins can extend the core functionality of the SSG, adding features such as SEO optimization, image processing, form handling, and content management. Developers can leverage plugins to tailor their websites to specific requirements and preferences, without the need for custom development. Additionally, plugin ecosystems foster community collaboration and innovation, with developers contributing new plugins and sharing them with the community.

**Enhanced SEO and Metadata Support:** Improving support for search engine optimization (SEO) and metadata management could be a focus for future development. This could include implementing automated SEO analysis and optimization tools, enhancing metadata configuration options, and integrating with SEO monitoring and reporting services.
Develop tools or plugins that automate the analysis and optimization of SEO elements within the website. These tools can scan the website for SEO deficiencies, such as missing meta tags, duplicate content, or broken links, and provide recommendations for improvement. Additionally, they can offer on-page optimization suggestions, such as keyword usage, title tag optimization, and internal linking strategies, to enhance search engine visibility and ranking.
Provide granular control over meta tags, including titles, descriptions, keywords, and social media tags, allowing developers to customize metadata for each page or content item. Additionally, support dynamic metadata generation based on content attributes or templates, ensuring that metadata remains accurate and relevant as the website evolves.

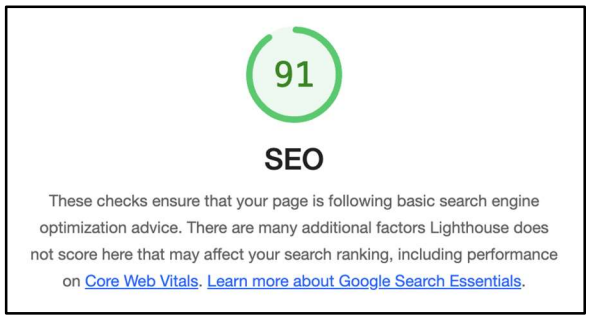


Figure 7: SEO result

**Enhanced Security and Reliability:** The transition to a static site architecture improved the security and reliability of our web assets. With no backend infrastructure or server-side vulnerabilities, the SSG solution offered enhanced resilience against cyber threats and reduced the risk of security breaches, ensuring the integrity and confidentiality of our website data.
Unlike dynamic websites that rely on databases to store and retrieve content, static sites do not require database interaction. This eliminates the risk of database-related vulnerabilities, such as SQL injection attacks or data leakage through misconfigured database permissions. Without a database backend, sensitive user data is not exposed to potential security threats, ensuring the integrity and confidentiality of website data.
This eliminates the risk of server-side scripting vulnerabilities, such as remote code execution (RCE) or

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

file inclusion attacks, which can compromise server security and lead to data breaches or system compromise.


Figure 8: Security and Safety

**Reduced Server Load and Resource Consumption:** By serving pre-rendered static files directly to users, the SSG solution reduced server load and resource consumption. This resulted in lower server costs, reduced energy consumption, and improved environmental sustainability, aligning with our organization's commitment to eco-friendly practices.

By reducing the demand for server resources such as CPU, memory, and bandwidth, SSGs enable organizations to operate with smaller and more cost-effective server infrastructure. With fewer server resources required to handle website traffic, organizations can downsize their hosting plans or opt for more affordable hosting providers, resulting in lower monthly hosting expenses and overall operational costs.

The decreased server load and resource consumption associated with SSGs translate to lower energy consumption in data centers and server facilities.


Figure 9: Content rendering time by server

**Ease of Maintenance and Updates:** The SSG solution simplified the process of website maintenance and updates. With content stored as plain text files and version-controlled using Git, we achieved greater transparency, traceability, and ease of rollback, facilitating seamless content updates and site management tasks.

Static Site Generator solution simplifies the process of website maintenance and updates by storing content as plain text files, version-controlling content with Git, leveraging automated deployment pipelines, providing content preview and drafts functionality, and supporting content reuse and templating. These features and practices enhance transparency, traceability, and ease of rollback, facilitating seamless content updates and site management tasks for developers and content authors.

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

## 6 Conclusion

As we conclude this project, we can reflect on the journey we've undertaken and the accomplishments we've achieved. We've successfully built a modern web application powered by React and server-side rendering (SSR) using Vite. This project represents the dedication and hard work of our team, and it marks a significant milestone in our development journey.
Throughout this project, we've harnessed the power of React to create dynamic and responsive user interfaces. The use of Vite as our build tool and development server has greatly streamlined our development process, offering speed and efficiency.

The goals of this study is to give an overview of the current SSG available for developing, managing, and hosting static websites, to analyze the differences between dynamic and static websites and to dig deeper into static website development tools.

Website types – static and dynamic – were introduced and their differences were explained. Static websites are like compiled programming languages they will need time for compilation but will run fast afterward. Dynamic sites are like interpreted languages no compilation is needed, but performance-related and other problems might occur at runtime due to interactivity.

SSGs pre-render pages during the build process, generating static HTML files for each page of the website and serve them to browser. With pre-rendered pages, content is immediately available to users request, reducing overhead and enhancing the overall browsing experience.

We found that static tools are modern but developer-centric. Easy-to-use UI for website content editors, bloggers, etc. But are still lacking the interactivity part and need custom setups to combine them with different tools.
Wider adoption of static technologies for content-centric websites would simply make the Internet a better place and reduce the possibilities of network security due to pre-generated content on servers and provide a better user experience..

### Future work

Focus on further optimizing the performance of the SSG solution. This may involve exploring advanced caching mechanisms, implementing lazy loading for assets, and optimizing asset delivery for improved website speed and responsiveness.
Improving the content management capabilities of the SSG solution could be a priority. This might involve developing intuitive user interfaces for content creation and editing, and implementing content versioning and rollback functionalities.
Improving support for search engine optimization (SEO) and metadata management. This includes implementing automated SEO analysis and optimization tools, enhancing metadata configuration options, and integrating with SEO monitoring and reporting services.
Continuously improving the user experience (UX) of the SSG solution could be a key area, this may involve gathering user feedback, conducting usability testing, and iteratively refining the user interface

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

and interaction design to enhance usability and accessibility.

7 References

[1] Petersen, H. (2016). From Static and Dynamic Websites to Static Site Generators. University of TARTU, Institute of Computer Science.

[2] Camden, R., & Rinaldi, B. (2017). *Working with Static Sites: Bringing the Power of Simplicity to Modern Sites (1st)*. Reilly Media, Inc.

[3] R. Kumar, R. K. Ratnesh, J. Singh, R. Chandra, G. Singh and V. Vishnoi, Recent Prospects of Medical Imaging and Sensing Technologies Based on Electrical Impedance Data Acquisition System, Journal of the Electro Chemical Society, 2023, 170, 117507.
https://doi.org/10.1149/1945-7111/ad050f

[4] R. Kumar, R. K. Ratnesh, J. Singh, A. Kumar and R. Chandra, IoT-Driven Experimental Framework for Advancing Electrical Impedance Tomography, Journal of Solid-State Science and Technology, 2024, 13, 027002.
https://doi.org/10.1149/2162-8777/ad2331

[5] Mathias Biilmann Christensen. (2015, November) Static Website Generators Reviewed: Jekyll, Middleman, Roots, Hugo.
https://www.smashingmagazine.com/2015/11/static-website-generators-jekyll-middleman-roots-hugo-review/

[6] Eduardo Bouças. (2015, May) An Introduction to Static Site Generators.
https://eduardoboucas.com/blog/2015/05/21/an-introduction-to-static-site-generators.html

[7] Wikimedia Foundation. Static web page.
https://en.wikipedia.org/wiki/Static_web_page

[8] Wikimedia Foundation. Dynamic web page.
https://en.wikipedia.org/wiki/Dynamic_web_page

[9] A. Garg, R. K. Ratnesh, Solar Cell Trends, and the Future: A Review, Journal of Pharmaceutical Negative Results, 2022, 13, 2051-2060.
https://doi.org/10.47750/pnr.2022.13.S06.268

[10] Messenlehner, B., & Coleman, J. (2019). *Building web apps with wordpress* (2nd ed.). O'Reilly Media.

[11] Harwani, B. (2015). *Foundations of Joomla!* (2nd ed.) [PDF]. https://doi:10.1007/978-1-4842-

*ISSN:1624-1940*
DOI 10.6084/m9.figshare.26090392
http://magellanes.com/

0749-9

[12] W3Techs. Usage of reverse proxy services for websites. https://w3techs.com/technologies/overview/proxy

[13] Chris Bach. (2015, September) Instant Cache Invalidation
https://www.netlify.com/blog/2015/09/17/continuous-deployment

[14] Chris Bach. (2015, September) Continuous Deployment.
https://www.netlify.com/blog/2015/09/17/continuous-deployment/

[15] Alan Shimel. (2013, June) 7 of 10 leading WordPress plugins are vulnerable.
https://www.networkworld.com/article/745486/opensource-subnet-7-of-10-leading-wordpress-plugins-are-vulnerable.html

[16] Sean Work. (2011) How Loading Time Affects Your Bottom Line.
https://blog.kissmetrics.com/loading-time/