Tampere University

Thao Huynh

WEB TECHNOLOGIES COMPARISON BETWEEN NUXT.JS AND ASP.NET

Master's thesis
Faculty of Information Technology and Communication Sciences
Examiner: Antti Sand
Examiner: Timo Poranen
February 2024

ABSTRACT

Author : Thao Huynh
Master's thesis
Tampere University
Master's Degree Programme in Computing Sciences, Software, Web & Cloud
2.2024

Web technology is increasingly developing and optimising with various new frameworks. Choosing web technology is suitable for the company's project and platform plays a key role in allocating money and human resources economically for the company while avoiding difficulties when scaling up the system. This thesis presents a comparison between two popular full-stack web frameworks, Nuxt.js and ASP.NET, focusing on their performance, features, and developer experience. Nuxt.js is a JavaScript framework based on Vue.js, predominantly used for building server-side rendered websites. ASP.NET, on the other hand, is a full-stack web framework developed by Microsoft, offering a wide range of tools and features for building robust and scalable web applications. The benchmark tests reveal that the ASP.NET application outperforms the Nuxt.js application in terms of loading, scripting, rendering, painting, and system times, as well as first contentful paint, largest contentful paint, and speed index. However, the generated code for the ASP.NET application is larger. Nuxt.js is more cost-effective for smaller projects, while ASP.NET is suitable for enterprise-level applications. The choice between Nuxt.js and ASP.NET depends on the team's skillset and project requirements. Nuxt.js offers a familiar syntax and component-based development, while ASP.NET provides stability and extensive documentation with its mature C# language. Error handling and security also differ between the two frameworks, with Nuxt.js focusing on client-side handling and requiring more attention to detail in security implementation, while ASP.NET emphasizes server-side error handling and provides a more robust security foundation. For the community support, Nuxt.js having an active open-source community and ASP.NET being backed by Microsoft with a more extensive resources and support. In conclusion, both Nuxt.js and ASP.NET can meet the demands of most projects and are suitable for different scales of projects. Nuxt.js is more agile and flexible for smaller projects, while ASP.NET's structure and tooling can handle larger, enterprise-level applications. In terms of control level, Nuxt.js offers more flexibility and customization, while ASP.NET provides a structured framework with built-in features. About performance, ASP.NET provides better performance than Nuxt.js. However, the selection between the two frameworks relies on the precise needs of the project and the team's expertise.

Keywords: comparison, web technologies, Nuxt.js, ASP.NET

ACKNOWLEDGEMENTS

I would like to thank my supervisor Antti Sand for his great support, feedback and advice during this thesis.

Tampere, 14 February 2024

Thao Huynh

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

API	Application Programming Interface
CSS	Cascading Style Sheets
ES6	ECMAScript 6
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
LESS	Leaner Style Sheets
ORM	Object-relational mapping
RQ	Research question
Sass	Syntactically Awesome Style Sheets
SEO	Search Engine Optimization
SQL	Structured query language

1. INTRODUCTION

The Internet has reshaped how people connect, communicate, and conduct business. The centre of this digital transformation is web applications enabling us to accomplish a broad spectrum of tasks and access information through web browsers. These applications are built using a combination of programming languages, databases, and programming frameworks that work together to provide interactive and dynamic user experiences.

A web application refers to a software application that provides an interactive experience to users. Unlike traditional desktop applications, web applications run on remote servers and are accessed over the internet. They are designed to be independent platforms, and users can access to them using a web browser or an internet connected device. Web applications leverage a combination of front-end and back-end technologies to deliver dynamic content, handle user interactions, and interact with databases or external APIs. They can range from simple websites with basic functionality to complex, feature-rich applications such as online stores, social media platforms, or collaborative tools. The key advantages of web applications include easy accessibility from anywhere with an internet connection, cross-platform compatibility, scalability, and the ability to provide real-time updates and collaboration [1].

From the static sites of the early web to the dynamic, data-driven applications of today, the evolution of web development mirrors the rapid pace of technological advancement. Fundamental technologies such as HTML, CSS, and JavaScript have paved the way for a new generation of frameworks, libraries, and tools. These frameworks offer various components, libraries, and tools that simplify the coding process and improve productivity. The continuous evolution of web tools and the advent of responsive design have not only transformed the visual aspects of the web but have also heightened the demand for seamless, user-centric experiences.

Web frameworks have become an indispensable tool for developers in building robust and efficient web applications. Two of the most recent full-stack frameworks are ASP.NET and Nuxt.js. By using these frameworks, developers can focus on building

innovative and feature-rich applications while relying on established patterns, best practices, and a supportive community. This results in better efficiency, time-saving, abstraction of complexity, scalability, maintainability, and faster development cycles [2].

1.1 Thesis goal

ASP.NET and Nuxt.js frameworks provide different capabilities aligns with different project requirements.This thesis, therefore, aims to compare these two frameworks on a specific case study in various aspects. The result of this thesis may be helpful for software architects and developers to gain valuable insights into which framework suits better for their purpose.

1.2 Research method

The thesis analyses the execution time, performance, code generated, cost, development complexity, error handling, security, and community support of concerned frameworks by developing a news application including front-end side, back-end side and database. Users can login to post new news, delete and modify old ones. This web application was built with the same functionality across two frameworks, as the means of comparison between them.

By examining various criteria, the thesis offers developers valuable insight to determine which framework can provide the necessary tools and functionalities to yields the best results for their specific project.

2. FULL-STACK WEB FRAMEWORKS

In the world of web development, full-stack frameworks play a pivotal role in streamlining the development of dynamic and interactive web applications. These frameworks are comprehensive sets of tools, libraries, and technologies that enable developers to work on multiple layers of the application. By providing a unified and consistent architecture, full-stack frameworks promote code reusability and reduce the complexity of managing multiple technologies and dependencies.

Full-stack web frameworks typically include components for front-end development, allowing developers to create visually appealing and engaging user interfaces. These frameworks offer different tools for development, including robust templating systems, responsive design capabilities, and client-side scripting frameworks [3]. With these features, developers can build dynamic and interactive web experiences that enhance user engagement and satisfaction .

However, full-stack web frameworks go beyond front-end development. They also provide a comprehensive set of back-end capabilities that are essential for building robust and scalable web applications. These capabilities include server-side routing, which allows developers to define endpoints and handle HTTP requests efficiently. Full-stack frameworks also facilitate seamless integration with databases, enabling teams to interact with operations like querying, updating, and deleting records. Additionally, these frameworks offer built-in authentication mechanisms, ensuring secure access to web applications. They also provide tools for API development, allowing developers to expose functionalities and interact with external systems or client applications [2].

One of the primary advantages of full-stack web frameworks is that they offer engineers a standardized approach to developing websites. They often follow well-defined patterns, like Model-View-Controller (MVC) or Model-View-ViewModel (MVVM), which enables the structuring of code [3]. These frameworks, together with their features, diminish the necessity for developers to start from scratch and accelerate the pace of the development process. By adhering to these architectural patterns, developers can easily navigate and understand the program structure, making it easier to collaborate with team members and maintain the application over time.

Full-stack web frameworks boast a rich ecosystem of plugins, extensions, and libraries. This ecosystem empowers developers by streamlining common tasks and extending the

framework's functionality. For example, frameworks often provide form validation mechanisms, session management, caching, and error handling functionalities [1]. By utilizing these pre-existing features, developers can save time and energy by avoiding the need to create everything from scratch for each project. This not only accelerates the development process but also guarantees that applications are constructed on a strong foundation, benefiting from the combined expertise and experience of the framework's developers.

Moreover, web frameworks often have active and supportive communities, providing resources, documentation, and community-driven updates and improvements. This vibrant developer community offers invaluable support. Here, developers have the opportunity to get assistance, exchange knowledge, and collaborate with fellow professionals tackling similar challenges.

The strong communities surrounding full-stack web frameworks encourage knowledge sharing and innovation. Developers can engage in discussions, attend meetups and conferences, and participate in open-source projects to enhance their skills and stay informed about the latest technology.

Furthermore, the community-driven updates and improvements for full-stack web frameworks ensure that they stay updated with the rapidly changing web development. These updates often introduce new features, performance optimizations, and security enhancements, allowing developers to leverage the latest advancements without having to start from scratch. The active development and maintenance of full-stack frameworks provide developers with confidence in the longevity and stability of their chosen technology stack.

While full-stack web frameworks offer numerous benefits, they also have their limitations. One potential drawback is the learning curve associated with mastering the framework and its associated technologies. Engineers may be required to allocate time for mastering the framework's architecture, conventions, and best practices. However, once developers become proficient with the framework, they can reap the benefits of increased productivity and efficiency.

The full-stack web development landscape is brimming with new frameworks. Among them, ASP.NET and Nuxt.js stand out for their robust feature sets, thriving developer communities, and exceptional scalability. ASP.NET, a framework developed by Microsoft, focuses on creating web applications using the .NET platform. It offers a robust development environment, extensive tooling, and various features tailored for enterprise-grade applications. With ASP.NET, teams can leverage the power of the .NET ecosys-

tem, including the C# programming language, to build scalable and performant web applications. ASP.NET Core takes full advantage of modern web development practices, including cross-platform compatibility and containerization while Nuxt.js is constructed on the foundation of Vue.js, catering to developers seeking a modern and flexible JavaScript-based full-stack solution. It offers server-side rendering (SSR), static site generation (SSG), and single-page application (SPA) modes, granting developers the flexibility to select the rendering approach that aligns with their project needs. As a result, developers can focus more on building their applications instead of being burdened with complex setup and configuration processes [4].

3. NUXT.JS

3.1 Background

Nuxt.js is a versatile full-stack JavaScript framework constructed on the foundation of Vue.js. It aims to simplify the development of server-rendered Vue applications and offers a seamless transition between frontend and backend development. Nuxt.js provides a convention-based approach, empowering developers to concentrate on application development rather than configuring complex setups. With Nuxt.js, developers can create universal applications that can be rendered on both the server and the client. It offers features like server-side rendering (SSR), routing, state management, and an extensive plugin ecosystem [5]. Nuxt.js is known for its flexibility, performance, and ease of use, attracting developers who prefer the Vue.js ecosystem to build modern web applications.

One of the prominent characteristics of Nuxt.js is its capacity to generate static websites. With Nuxt.js, developers can pre-render applications during the build process, resulting in HTML files that can be served statically. This approach not only improves the initial loading time of the website but also allows enhanced search engine optimization (SEO) [5].

Nuxt.js enables developers to easily structure their codebase into reusable modules. These modules can be used to enhance the application with features such as routing, state management, authentication, and more. Nuxt.js embraces the Vue ecosystem and integrates seamlessly with popular Vue plugins, making it a versatile choice for web development [6].

Furthermore, Nuxt.js offers a range of functionalities, including automated code separation, server-side rendering, hot module replacement, and static file serving. It also supports various development modes, enabling users to switch between development, production, and testing environments effortlessly [5].

The Nuxt.js community is active, with numerous plugins and modules available to extend the functionality of web applications. The official Nuxt.js documentation is comprehensive and provides detailed guides, examples, and API references.

3.2 Node.js server

Node.js is a software environment that allows Nuxt.js code to run outside of a web browser, enabling its use for server-side scripting and command-line tools. Node.js harnesses the powerful V8 JavaScript engine, known for its speed and efficiency, to execute JavaScript code quickly. Node.js demonstrates exceptional performance in managing simultaneous requests and efficiently handling I/O-bound tasks, making it an excellent choice for constructing highly scalable and responsive applications. Node.js code can be executed on multiple operating systems without requiring any modifications [7].

Node.js provides multiple advantages as a runtime environment for server-side applications. It employs a single-threaded that efficiently handles a large volume of concurrent connections. This characteristic makes it ideal for real-time data applications. It can swiftly handle requests and return responses without waiting for I/O operations to finish. Moreover, Node.js also has various open-source packages and modules available through its package manager, npm. This extensive library facilitates easy access to prebuilt functionalities, accelerating development time. Since Node.js employs JavaScript as its programming language, developers can use the same language for full-stack development, promoting code reuse, library sharing, and seamless transitions between client-side and server-side development [8].

Node.js is a versatile runtime environment that finds application in various use cases across different industries. One common use case of Node.js is building web applications and APIs. The non-blocking I/O paradigm of Node.js allows for effective handling of multiple simultaneous requests, making it a good option for creating scalable and high-performing web services [9]. Node.js is also commonly used in real-time applications, such as chat applications, collaboration tools, and multiplayer games, as it excels in handling real-time data and facilitating bidirectional communication. Another popular use case is building microservices and serverless architectures, where Node.js enables developers to create lightweight, modular services that are easily deployed independently. Additionally, Node.js is often employed in building streaming services, IoT applications, and data-intensive applications, thanks to its efficient handling of asynchronous tasks and its integration with various data storage systems [10].

3.3 Vue.js

Vue.js, a JavaScript framework, that is open-source and commonly employed for developing single-page applications. It was first released in 2014 by Evan You, a former

Google engineer, and has since gained popularity among developers due to its simplicity, flexibility, and performance [4].

Vue.js employs a component-based architecture, enabling developers to construct reusable UI components and effortlessly handle the application's state. This approach makes it easier to create complex UIs and maintain large codebases, as each component can be developed and tested independently [6].

A notable feature of Vue.js is its reactivity system, which allows for efficient and automatic updating of the user interface in response to changes in data. It uses a virtual DOM to track changes to the application's state and efficiently update the UI. This approach leads to improved performance and a smoother user experience compared to traditional JavaScript frameworks [11].

Vue.js also offers an uncomplicated and user-friendly API for managing events., data binding, and animations. It supports Bidirectional data syncing, which enabes developers to quickly synchronize data between the user interface and application state. Vue.js also provides a built-in transition system, which makes it easy to add animations and transitions to components [12].

One of the benefits of Vue.js is its flexibility, which allows for easy integration with a variety of libraries and frameworks, such as React and Angular. It can also be used as a standalone library or incorporated into larger projects. This makes it a versatile tool for developers who need for the development of diverse applications, from basic web pages to complex SPAs [13].

Vue.js features a lively and active community of developers, offering abundant resources and support. It has a comprehensive documentation website, as well as a large number of tutorials, examples, and plugins available online. It also has an active GitHub repository, which is regularly updated with new features and bug fixes.

Vue.js is a lightweight framework, with a size of only 20-30KB, being excellent choice for mobile and fast load web page and efficient performance. It also supports server-side rendering, which improves SEO and allows for faster initial load times [11].

Vue.js is an effective and flexible JavaScript framework that is optimal for creating UIs and SPAs. Its component-based architecture, reactivity system, and visceral API make it comfortable to build complex UIs and manage large codebases. Its flexibility and lightweight nature make It is a highly versatile tool applicable to a diverse range of uses, and The large and active community associated with this tool offers a treasure trove of resources and robust support for developers [14].

4. ASP.NET

4.1 Background

ASP.NET developed by Microsoft is a powerful full-stack framework. It enables programmer to develop high-performance applications using the .NET platform. ASP.NET offers several frameworks, including .NET Core to developing modern fullstack systems. With ASP.NET Core, developers have the opportunity to harness the capabilities of C# to create efficient backend logic and seamlessly integrate it with frontend components. The framework provides features like MVC (Model-View-Controller) architecture, routing, authentication, and support for various data storage options [15]. ASP.NET is reliable in its scalability, security, flexibility, and extensive community support, causing it a preferred choice for enterprise-level systems.

ASP.NET is a versatile web framework that offers several features and benefits for developers. One of its key advantages is server-side scripting, which allows developers to write code using popular languages like C# or Visual Basic. This code is executed on the server, ensuring efficient processing and delivering the results to the client's browser [16]. Additionally, ASP.NET supports object-oriented programming principles, enabling developers to build modular and maintainable web applications.

To enhance the development experience, ASP.NET provides an integrated development environment (IDE) called Visual Studio. This powerful IDE offers a exhaustive set of equiments for building, testing, deploying , and streamlining the development process of fullstack development.

ASP.NET offers two main approaches for building web applications. ASP.NET Web Forms follows a traditional form-based model, abstracting HTML and client-side scripting. This approach allows developers to create web applications with a structure similar to Windows-based applications, simplifying development for those familiar with desktop application development [17].

On the other hand, the architechture of Model-View-Controller (MVN) of ASP.NET is a more modern and flexible approach. It configures fullstack applications with three main constituents: models, views, and controllers. Data and algorithmic logics are handled in models, while the views display user interfaces of the app. The controllers have responsibilities to access and update models. This architectural pattern promotes code organization, maintainability, and scalability [18].

ASP.NET Core is highlighted as an ideal choice for developers who require a technology that can work across different platforms (like Windows, Linux, and macOS) while delivering high-performance results. It is an open-source framework, meaning its source code is freely available for developers to inspect, modify, and contribute to. It's designed to be modular and lightweight, which allows developers to choose and use only the necessary components for their applications, enhancing efficiency and flexibility. ASP.NET Core is also well-suited for modern cloud-based and containerized applications. These types of applications are designed to be scalable, easily deployable, and manageable across distributed environments, such as cloud platforms and container orchestration systems like Docker and Kubernetes. ASP.NET Core offers a notable benefit in its capability to operate on various platforms. This adaptability allows developers to target a wider audience and support diverse deployment scenarios without significant modifications to the codebase [19].

4.2 ASP.NET Core SDK (.Net SDK)

The .NET SDK is the software development kit (SDK) for building web applications with ASP.NET Core. Various features and improvements of .NET SDK that enhance productivity and enable the creation of high-performance, scalable, and secure applications. With .NET SDK, web apps developed by .NET with improved support by Blazor can run fastly on Windows, Linux, and macOS using C# instead of JavaScript. Additionally, the SDK includes enhancements to the API development experience, real-time communication capabilities with SignalR, and support for building microservices using Docker containers [18]. With an active community and extensive documentation, the ASP.NET Core SDK allows developers to build modern, secure cross-platform applications efficiently.

An advantage of ASP.NET Core Runtime is its enhanced performance and scalability. With optimizations and improvements in the runtime environment, .NET applications can run faster and monitoring increased loads efficiently. This means that developers can build high-performing web applications that can manipulate a lot of concurrent users and deliver a seamless user experience. Additionally, ASP.NET Core Runtime offers improved security features, including support for industry-standard authentication protocols and protection against typical web vulnerabilities like cross-site request forgery (CSRF) or cross-site scripting (XSS). This ensures that applications built on ASP.NET Core have robust security measures in place. Furthermore, ASP.NET Core Runtime is cross-platform, which means that developers can deploy their applications on Windows, Linux, and macOS environments, providing flexibility and accessibility [15].

4.3 Entity Framework Core (EFCore)

EFCore is the common object-relational mapping (ORM) package for .NET. It supplies a powerful and efficient way to work with databases in .NET applications. With EFCore, developers can easily map database tables to object-oriented models, allowing them to interact with the database using familiar object-oriented programming concepts. It supports complex data types, improves query performance, and enhances support for database migrations. EFCore also offers better integration with ASP.NET Core, causing it even easier to create data-driven applications. Additionally, it supports multiple database providers, allowing developers to work with different database systems seamlessly. EFCore simplifies database access and manipulation, improves performance, and provides a powerful tool for building data-driven applications in .NET [4].

EFCore common use case is its utilization in building data-driven web applications. Developers can easily map database tables to object-oriented models, enabling seamless integration between the application and the underlying database. This causes it an perfect option for developing CRUD (Create, Read, Update, Delete) applications to operate persistent data. Additionally, EFCore is well-suited for developing enterprise-level applications that require complex data modeling and relationships. It supports advanced querying capabilities, allows developers to handle sophisticated data structures and perform complex database operations efficiently [16]. Another use case for EFCore is in the development of cross-platform applications. Since it supports multiple database providers and can run on various operating systems, developers can build applications that can be deployed on different platforms without major modifications. EFCore is a strong package for building data-driven applications, spanning from simple web applications to extensive enterprise systems with intricate data handling.

EFCore is widely used by developers for various reasons. The important advantage of EFCore is its capability to simplify database access and eliminate the need for writing complex SQL queries. With its ORM capabilities, EFCore allows developers to manipulate database entities as regular objects in source code, abstracting away the underlying database interactions. This abstraction makes database operations more intuitive and reduces the amount of repetitive boilerplate code, leading to increased productivity and faster development cycles [18].

EFCore supports multiple database providers. It offers compatibility with various database systems such as SQLite, PostgreSQL, MySQL... This flexibility allows developers to collaborate with their preferred database system or seamlessly switch between different providers without significant code modifications. It also promotes database portability

and allows applications to be deployed across different environments or migrated to different databases easily [17].

EFCore offers features like database migrations, which greatly simplify the process of evolving the database schema over time. With migrations, developers can easily manage and apply changes to the database structure while preserving exist-ing data, ensuring smooth updates without data loss or disruption to the application [15].

.NET ecosystem integrates well with other frameworks and libraries, such as ASP.NET Core, enabling developers to leverage its capabilities throughout the entire application stack. This tight integration enhances code consistency, maintainability, and overall development efficiency [20].

Furthermore, EFCore incorporates performance optimizations, such as lazy loading, caching, and query optimization techniques, to improve overall application performance. These optimizations help minimize unnecessary database queries, enhance data retrieval efficiency, and reduce latency, resulting in faster response times and improved user experience [21].

5. RESEARCH METHODOLOGY

5.1 General introduction

The research method performs in shaping the outcome and validity of a study. Researchers must carefully select a method that aligns with their research goals, questions, and the nature of the study. In the realm of research, two primary approaches are commonly employed: quantitative and qualitative. Each approach offers distinct methodologies for data collection, analysis, and interpretation, providing researchers with different lenses through which to explore and understand phenomena. Depending on the research objectives, questions, and nature of the phenomenon under investigation, researchers may choose to adopt either a quantitative or qualitative research method, or even combine both approaches to gain a comprehensive understanding .

Quantitative research involves the structured collection and analysis of numerical data in a systematic manner. It bases on the principles of objectivity, statistical measurement, and analysis to uncover patterns, relationships, and trends. Through methods such as surveys, experiments, or the analysis of existing datasets, quantitative researchers aim to generate precise and replicable results. This approach allows for the examination of large sample sizes and the identification of statistically significant associations. Quantitative research is particularly well-suited for studying phenomena that can be quantified and analyzed using statistical techniques [22].

Furthermore, qualitative research focuses on the exploration of subjective experiences, meanings, and social contextsIt encompasses gathering and analyzing data that is non-numeric, including text, images, and observations. Qualitative researchers utilize methods like focus groups, observations, interviews, or analyzing documents to collect comprehensive and detailed data. By examining individual perspectives, social interactions, and cultural contexts, qualitative research seeks to reveal the complexities and nuances of a phenomenon. It aims to understand the "why" and "how" questions [23].

While quantitative research emphasizes generalizability and statistical inference, qualitative research prioritizes a deep understanding of the research subject and the exploration of diverse perspectives. Both approaches offer unique strengths and limitations. Quantitative research provides rigor, objectivity, and the ability to draw statistical conclusions, while qualitative research allows for in-depth exploration and contextual understanding [24].

At times, researchers may adopt a mixed methods approach, integrating both quantitative and qualitative methodologies. This combination enables data triangulation, leading to a deeper and more thorough comprehension of the research subject. By combining the strengths of both approaches, researchers can capture both the breadth and depth of a phenomenon, enriching their findings and interpretations [25].

5.2 Research method

In this study, I employ the qualitative research approach, to compare these two frameworks on a specific case study. The result of this thesis may be helpful for software architects and developers to gain valuable insights into which framework suits better for their purpose.

The qualitative approach is used to explore development experiences and gain a rich understanding of the research topic. While the study incorporates quantifiable data, it cannot be classified as quantitative because the characteristic essential to the quantitative approach involves the use of large data samples, which is not the case in this study.

Additionally, the qualitative approach is utilized to acknowledge the subjective nature of evaluation criteria such as code generated, cost, development complexity, error handling, security, community support which cannot be solely measured quantitatively. By embracing qualitative methods, this study facilitates the exploration of subjective observations.

Qualitative research approaches provides flexibility in adapting the research design and data collection methods throughout the study. Given the dynamic changing of the full-stack development field, the method allows this study to capture meaningful perspectives and experience with Nuxt.js and ASP.NET frameworks.

5.3 Research questions

This thesis aims to answer the research questions as follows

RQ1: What are criteria for comparation between the two frameworks Nuxt.js and ASP.NET?

RQ2: How to evaluate these criteria?

RQ3: Based on the analysis, which framework is suggested to the development team to yield the better results for their project?

For the initial research question, there is a comparison conducted between the ASP.NET and Nuxt.js frameworks in two categories: benchmarks and features. The criteria for this comparison is explained in the following section, "Criteria for Comparison" (5.4), for more comprehensive details.

To evaluate the benchmark criteria, including Loading Time, Scripting Time, Rendering Time, Painting Time, System Time, First Contentful Paint, Largest Contentful Paint, Speed Index, Performance, Total Blocking Time, Accessibility, Best Practices, Cumulative Layout Shift, and SEO, tools such as Lighthouse and the Inspect option in Google Chrome are employed. The feature criteria, on the other hand, are assessed based on the actual development experience with the chosen frameworks.

Based on the obtained results, a recommendation will be made between the two platforms to answer the third research question.

5.4 Criteria for comparison

There are various comparison criteria used to compare the two frameworks ASP.NET and Nuxt.js in different aspects to draw a conclusion. These criteria are divided into two categories: benchmarks and features.

5.4.1 Benchmarks

The following benchmarks have been measured to evaluate the efficiency and effectiveness of each framework

- Loading Time: indicates the duration for a web page to completely load and be ready for interaction. It includes the time to download HTML, CSS, JavaScript files, images, and other resources. This metric provides information on the total time taken to load the page.
- Scripting Time: measures the time taken to parse and execute JavaScript code on a web page. It includes the time to load external JavaScript files and execute inline JavaScript. This benchmark used to examine the timeline and identifying JavaScript execution durations.
- Rendering Time: measures the time taken to render and display the content of a web page. It includes parsing HTML, applying CSS styles, and rendering the visual elements. It analyzes the rendering time by examining the different stages of the rendering process.

- Painting Time: refers to the time taken to paint the visual elements of a web page on the screen. It includes operations such as layout, paint, and composite.
- System Time: measures the time spent by the browser's underlying system processes during web page loading and rendering. It includes activities such as network communication, disk I/O, and CPU utilization. It can offer information on the performance timeline and identifying resource utilization.
- First Contentful Paint denotes the duration required for the initial content element to appear on the screen. It provides an indication of how quickly users see visual feedback from the web page. A faster First Contentful Paint indicates a better user experience, as it reduces perceived loading time and provides a sense of progress.
- Largest Contentful Paint evaluates the duration it takes for the most significant content element to be visible on the screen. A speedier Largest Contentful Paint is desirable because it ensures that users can access and interact with the most important content quickly.
- Total Blocking Time signifies the cumulative duration of time that is blocked or delayed during the page load process. It quantifies the time when the main thread is occupied with long tasks, preventing user input and interactivity. A lower Total Blocking Time is preferable, as it indicates fewer interruptions to user interactions and a more responsive web page.
- Cumulative Layout Shift refers unexpected layout shifts or content movement that occurs during page load. It quantifies the visual stability of the page by tracking the cumulative impact of layout shifts. A lower Cumulative Layout Shift score suggests a more stable and visually pleasing experience, with reduced chances of users encountering sudden content movement or elements shifting unexpectedly.
- The Speed Index evaluates the speed at which the visual content of a web page is displayed during its loading process. It calculates the average time when visible portions of the page become painted. A lower Speed Index score indicates a quicker perceived loading time and a more responsive web page.
- Performance represents the speed and responsiveness of a web page. This criterion sums up some previous criteria to evaluates various aspects. A higher score calculated by Lighthouse tool indicates better performance in terms of how quickly the page loads and becomes usable for the user.

- Accessibility assesses how well a web page adheres to accessibility standards and guidelines. It evaluates the page's design and structure to ensure that it is accessible to users with disabilities. This benchmark checks for elements such as proper semantic markup, alternative text for images, screen readers, keyboard navigation support, and sufficient color contrast. A higher score indicates better accessibility, demonstrating that the web page is configured to be usable by a broaden range of users, including those with visual impairments or mobility limitations.
- Best Practices evaluates the adherence of a web page to recommended web development practices and standards. It checks for common issues, security vulnerabilities, and deprecated or inefficient practices. This benchmark promotes the use of modern web technologies, secure connections, and optimized code. A higher score indicates better adherence to best practices, ensuring that the web page follows industry standards and guidelines to deliver a reliable and secure user experience.
- SEO (Search Engine Optimization) measures how well a web page is optimized for search engines. It evaluates the page's structure, content, and metadata to assess its visibility and ranking in search engine results. This benchmark checks for elements such as proper use of meta tags, keyword usage, structured data markup, and mobile-friendliness. A higher score indicates better optimization for search engines, increasing the likelihood of the page being discovered and ranked higher in relevant search queries.
- Size of generated code refers to the size of the compiled, bundled, or optimized code that is produced as a result of building a web application or website. It signifies the ultimate size of the code transmitted to the user's browser upon webpage loading. A smaller code size is generally desirable for optimal web performance.

5.4.2 Features

These following features have been compared between two frameworks, provides developers with important insights to make well-informed decisions that align with the project's objectives.

- Cost: Cost refers to the expenses associated with adopting and utilizing a web framework. This includes any licensing fees, subscription costs, hosting charges, and potential costs for additional services or integrations. Evaluating the cost

helps determine if the framework aligns with the project's budget and financial constraints.

- Development Complexity: Development complexity measures the level of difficulty or ease involved in building applications using a web framework. It encompasses factors such as the learning curve, available documentation, community support, and the simplicity or complexity of the framework's architecture. Choosing a framework with a suitable development complexity level ensures that developers can work efficiently and effectively throughout the project's lifecycle.
- Error Handling: Error handling assesses how well a web framework handles and manages errors and exceptions that occur during application execution. This includes providing informative error messages, logging mechanisms, and mechanisms for gracefully handling and recovering from errors. Opting for a framework with robust error handling capabilities simplifies debugging, troubleshooting, and maintaining application stability.
- Security: Security pertains to the framework's inherent features and protocols designed to safeguard web applications. This criterion includes aspects such as protection against common vulnerabilities, authentication and authorization mechanisms, secure session management, and adherence to security best practices. Prioritizing a framework with strong security measures helps protect sensitive data and ensures a secure application environment.
- Tools refers the availability and quality of development tools, utilities, and integrations provided by the framework's ecosystem. This includes IDE support, command-line interfaces, build systems, package managers, and third-party integrations. Opting for a framework with a rich set of tools can enhance developer productivity and streamline the development process.
- Community support revolves around the presence and involvement of a developer community that actively contributes, offers assistance, shares knowledge, and collaborates to enhance the framework. A strong community can greatly benefit developers in terms of learning resources, troubleshooting assistance, and the availability of external sources and extensions.

5.5 Tools used

5.5.1 Tools used for developing the Nuxt.js app.

- Node.js Runtime

- Visual Studio Code (VS Code):

5.5.2 Tools used for developing the ASP.NET app

- ASP.NET Core SDK 8.0:
- Visual Studio 2022
- Entity Framework Core 8.0 (EFCore 8.0)
- ASP.NET Core Identity (ACI) for Entity Framework Core 8.0

5.5.3 Tools used for evaluation

- Lighthouse: is a freely available automated tool created by Google to assist developers in enhancing the quality and performance of web pages. It conducts audits and offers recommendations on different aspects of web development, covering performance, accessibility, best practices, and search engine optimization (SEO). With Lighthouse, developers can easily assess their web pages against a set of predefined metrics and guidelines, gaining valuable insights into areas that need improvement. Whether it's optimizing page load times, ensuring accessibility for all users, or implementing SEO best practices, Lighthouse allows developers to enhance their web applications and deliver a better user experience. By using this tool, developers can identify and address potential issues, ultimately creating faster, more accessible, and highly performant websites.
- The above Inspect option in Google Chrome is a powerful tool for evaluating web performance. It equips developers with a robust toolkit and valuable insights to evaluate and improve the performance of their web sites. Performance tabs offer insights into the rendering process, highlighting potential issues.

6. APPLICATION FUNCTIONALITY

The implemented application is a news application that includes the front-end, back-end, and database components. There are three types of users: visitors, customers, and staff. Only the admin can assign the staff role to users. Visitors have the option to register as customers and add their favorite news articles to their baskets for later reading. Staff members have extensive rights, including the ability to post new news articles, delete and modify existing ones that they have created after logging in. However, a staff member cannot modify or delete news articles written by other users. When a staff member posts a new article, a separate window opens to enhance the user experience. News articles can be created by staff members or fetched through APIs and is categorized into various areas such as general, entertainment, health, science, sports, technology, and business.

6.1 Registration

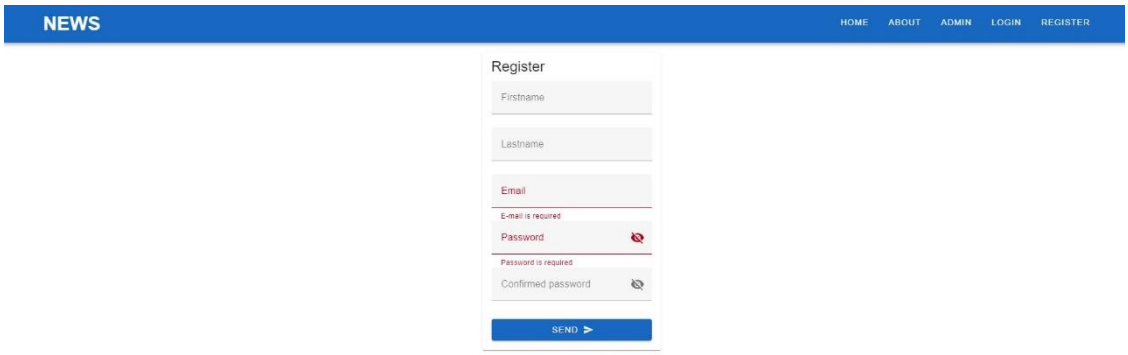

Figure 1. *Registration screenshot*

The register button is at the top right of the home page. It allows visitors and anonymous users to register as new customers. To register, information of first name, last name, email, password, confirmed password needs to be provided. After completing the form, users have to click on the "Send" button to become users of the website.

6.2 Log in

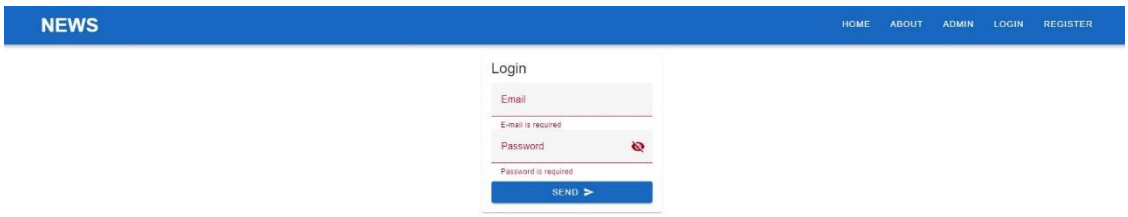

Figure 2. *Login screenshot*

The log in button is for authenticated users after their registration. It is next to the register button on the navigation bar. After pressing it, users are led to a different page where they need to type their email and password. After logging in, register button becomes invisible and login buton changes to logout.

6.3 News page

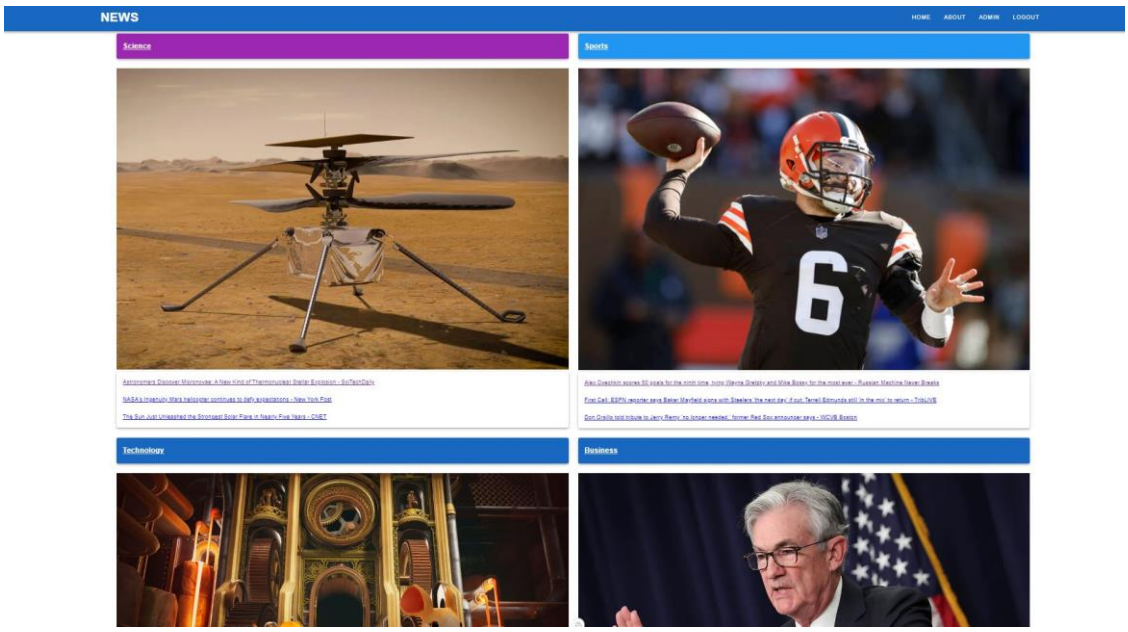

Figure 3. *News page screenshot*

The Home button directs all users to this web application's main page, which is a collection of news stored in database or fetched through APIs. News is categorized into various areas such as general, entertainment, health, science, sports, technology, and business. All users including visitors, customers and staff can view the news of all users on this page. When they click on the title of each category, they are directed to a new page which displays all the news in that category. There is a button named "Add To Feed" visible to authenticated users below each piece of news allows registered users to add their favorite news articles to their baskets for later reading.

6.4 Favorite news

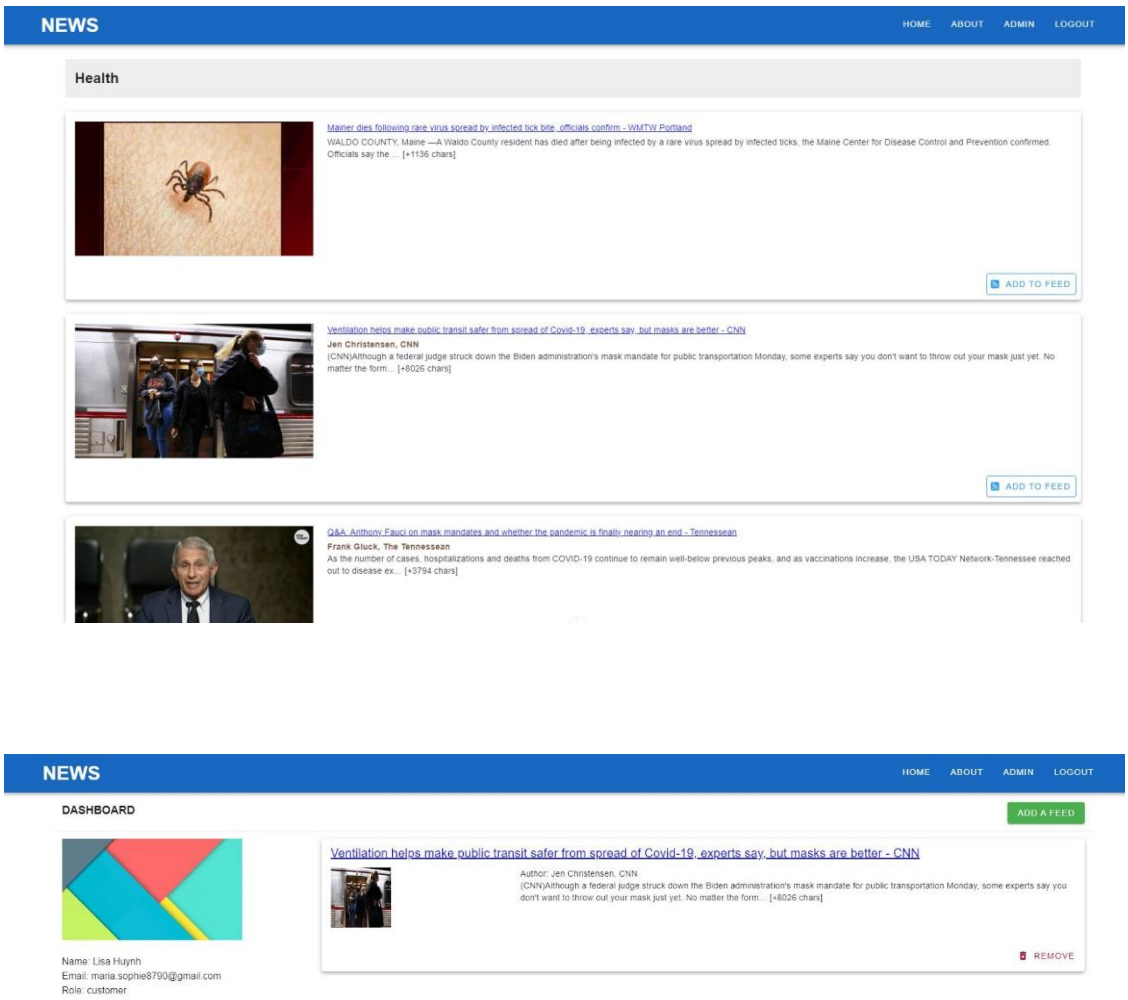

Figure 4. *Manage favorite news screenshot*

The Admin page allows authenticated users to manage their favourite news. Users need to log in to their account to view data. To add a piece of news to the favorite basket, users need to click on the "Add To Feed" button located below each one. The newly added post will be displayed in a table , and this table lists all the favorite news personally chosen by each user. There is also a Remove button to delete the news in case users are no longer interested in reading it.

6.5 Staff dashboard

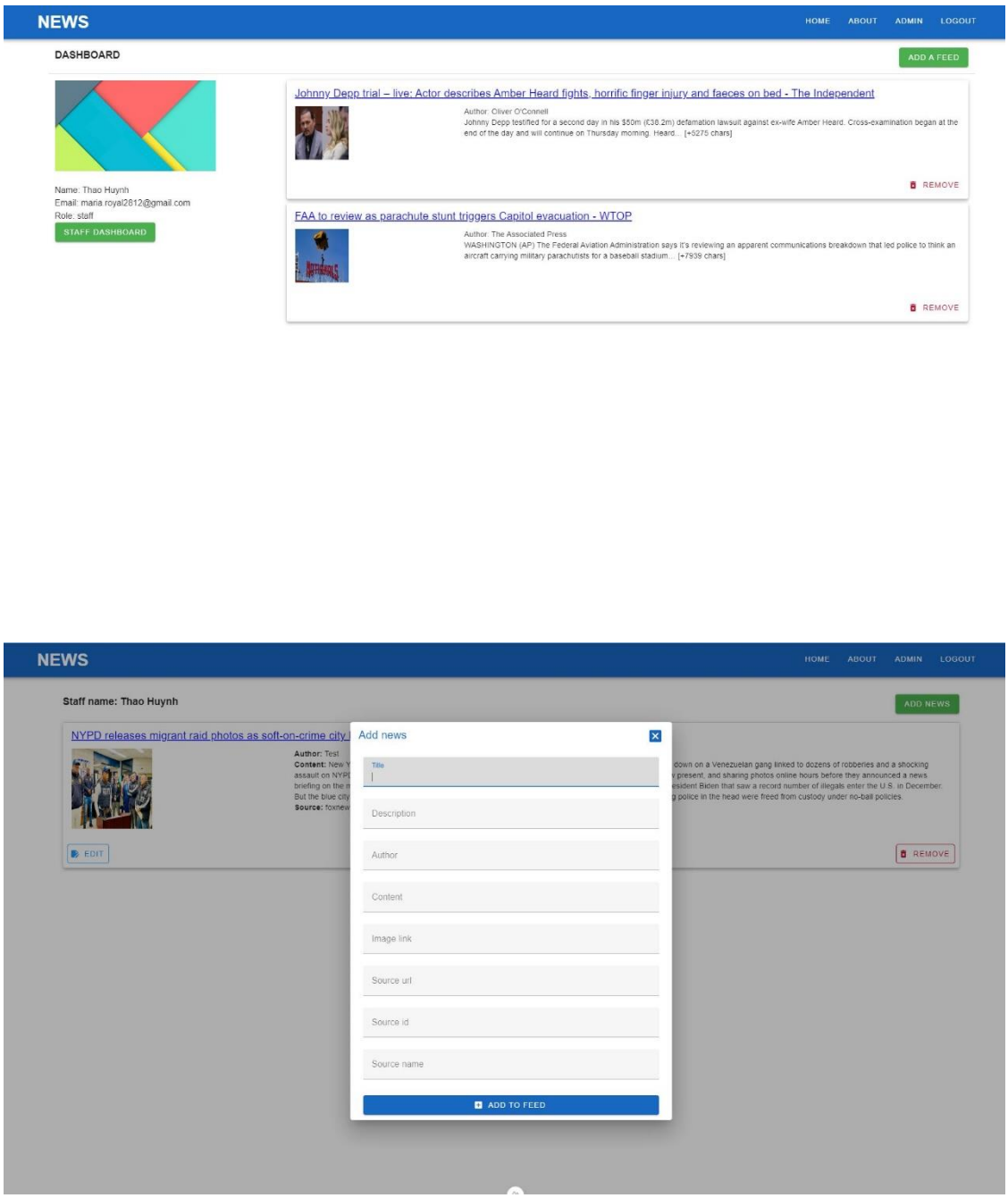

Figure 5. *Staff dashboard screenshots*

This web page enables staff users to upload their new piece of news. Users must log in as staff to see the button Staff Dashboard. After clicking on it, users need to fill in the form with required fields like Title, Description, Author, Content, Image link, Source url, Source id, Source name. The newly created post will be displayed in a table with related

information and 2 buttons for Edit and Remove the piece of news. However, a staff member cannot modify or delete news articles written by other users.

6.6 Edit the post

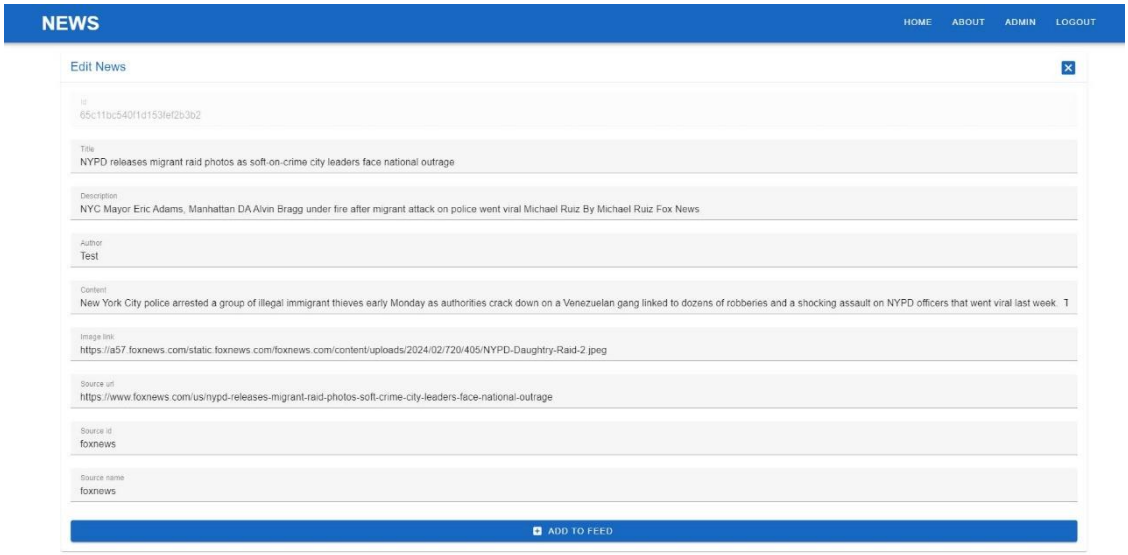

Figure 6. *Edit the posts creenshot*

The Edit button allows users to modify the posts they created. After clicking it, a new page will be displayed for users to modify the data. If users do not want to change the information anymore, they can simply press the close button at top right corner. Otherwise, the modification will be saved to the database.

6.7 Database

The database consists of three tables: posts, staffposts, and users.

Table posts: stores the favorite news added by customers and staff.

Table staffposts: This table contains the news created by staff.

Table users: information about the users is saved in this table.

6.7.1 MongoDB for the Nuxt.js app

MongoDB is a NoSQL database that employs a document-oriented data model. Unlike conventional relational databases that organize data into tables, MongoDB saves data in flexible documents using key-value pairs. This makes it ideal for storing and querying

complex, unstructured data. It provides the necessary flexibility, efficiency, and scalability to effectively handle dynamic content, simplify data retrieval, and power the backend of Nuxt.js applications [26].

MongoDB was chosen because it is well-suited to Nuxt.js. MongoDB's flexible data model aligns well with Nuxt.js's dynamic nature. Nuxt.js supports server-side rendering, allowing the generation of dynamic content in response to user requests. MongoDB's schema-less approach aligns well with the evolving data needs of Nuxt.js applications, enabling developers to store and retrieve data without strict data structures or schemas. This flexibility is especially advantageous in situations where the data structure may change often or where dynamic content requires efficient management [27].

Secondly, MongoDB's JSON-like document storage format, integrates seamlessly with Nuxt.js's handling of JSON data. Nuxt.js provides good support for JSON-based data manipulation, causing it more convenient for developers to work with MongoDB's storage. This compatibility simplifies the handling and manipulation of data between the frontend and backend, enhancing the development experience and reducing the need for complex data transformations [28].

Furthermore, MongoDB's powerful query language and indexing capabilities complement Nuxt.js's data retrieval needs. Nuxt.js applications often require efficient querying and retrieval of data from the database, and MongoDB's rich query language and flexible indexing options allow developers to optimize data access and retrieval according to their application's specific requirements. This helps improve performance and responsiveness, resulting in a smoother user experience [29].

Moreover, MongoDB's ability to scale horizontally and provide high availability makes it well-suited for scaling Nuxt.js applications. As Nuxt.js applications expand with increasing user base and data volume, MongoDB's scalability allows for distributing data across multiple servers, enabling effective management of large datasets and high traffic. Moreover, MongoDB's built-in replication supports automatic failover and data redundancy, ensuring high availability and resilience for Nuxt.js applications [30].

6.7.2 SQLite for the ASP.NET app

SQL is a prevalent and extensively utilized relational database management system (RDBMS) recognized for its straightforwardness, lightweight design, and user-friendly interface. It was first developed as a file-based database system for use in embedded systems, but it has since grown to become one of the widely used database platforms in use today [31].

An important characteristic of SQLite is its self-contained design. Unlike traditional client-server database systems, SQLite does not require a separate server process or runtime environment. Instead, it stores the entire database within a single file on disk, simplifying management and deployment. This also makes SQLite highly portable, as the database file can be easily copied or moved between different systems without the need for any special tools or utilities [32].

Another benefit of SQLite is its compact size and minimal resource usage. The entire SQLite library can be compiled into a single file of less than 500KB, making it straightforward to integrate into applications with constrained resources. This also makes SQLite highly scalable, as it can handle small to medium-sized databases with ease [33].

SQLite offers extensive functionality, including transactions, indexing, and foreign key constraints. It also provides a full-featured SQL implementation, including support for subqueries, aggregate functions, and common table expressions. This makes SQLite an excellent option for developers who require handling intricate data structures and executing advanced queries [34].

Alongside its features and performance, SQLite is known for its strong reliability and security. It utilizes a journaling file format to maintain data integrity, even in scenarios like system crashes or power failures. SQLite also provides strong encryption support, allowing developers to secure their data with industry-standard encryption algorithms [35].

SQLite is compatible with numerous platforms, such as Windows, macOS, Linux, and various mobile operating systems. It benefits from a sizable and engaged development community that consistently delivers updates and bug fixes. Being open-source, SQLite is freely accessible and can be customized to accommodate the requirements of specific applications.

Hence, SQLite is a compelling option for developers seeking a lightweight, self-contained, and user-friendly database system. Its compact size, minimal resource footprint, and extensive feature set render it suitable for applications of varying scales, ranging from small embedded systems to large-scale enterprise applications.

6.7.3 The impact of different database platforms on the applications' performance

The choice between MongoDB and SQLite can indeed have an impact on the performance of Nuxt.js and ASP.NET applications. However, it is important to note that in development process, MongoDB is commonly used for Nuxt.js applications, while SQLite

is commonly used for .NET applications. Therefore, using the same database for both Nuxt.js and ASP.NET applications might not be a sensible comparison since this is not the way applications developed in real-world scenarios. MongoDB and SQLite represent NoSQL and SQL database in fullstack system in production state.

In the industry, MongoDB is often chosen for Nuxt.js applications because MongoDB's JSON-like document format is native to JavaScript, making it a natural fit for Nuxt.js applications. The seamless integration between MongoDB and Nuxt.js allows for easy conversion of data between the two formats, simplifying development and reducing the need for complex data transformations [26]. MongoDB's schema-less approach enables developers to store and retrieve data without requiring predefined schemas or migrations. This flexibility is particularly beneficial for rapid development and prototyping, as it enables developers to iterate quickly and make changes to the data structure as needed. The simplicity and familiarity contribute to MongoDB's popularity among Nuxt.js developers [30].

On the other hand, SQLite is commonly used for ASP.NET applications due to its compatibility with the .NET framework. ASP.NET applications are built using the .NET framework, and SQLite provides a seamless integration with .NET through its ADO.NET data provider. This enables developers to apply their existing knowledge and skills in ADO.NET to interact with the SQLite database within their ASP.NET applications [32]. SQLite is also known for its efficiency and low resource consumption. It is engineered to be fast and lightweight, which makes it well-suited for applications with limited resources or those that demand quick response times. In scenarios where the application and database are co-located, such as desktop applications or small-scale web applications, SQLite's performance can be particularly advantageous [29].

Given the prevalent usage patterns in the industry, comparing the performance of both Nuxt.js and ASP.NET applications using the same database platform might not provide a meaningful or accurate comparison of their capabilities. The goal of the thesis is to provide developers in the industry with valuable insights into the practical usage and performance of these programs. By focusing on the commonly adopted approaches and technologies in the industry, the comparison can offer meaningful information that aligns with real-world scenarios and helps developers make informed decisions based on industry practices.

7. IMPLEMENTATION OF THE APPLICATIONS

The implemented application is a news application consisting of front-end, back-end, and database components with different roles of users. The news articles can originate from staff members or be fetched through APIs, and they are categorized into various areas such as general, entertainment, health, science, sports, technology, and business.

7.1 App implementation in Nuxt.js

7.1.1 Project structure

The news app created with Nuxt.js framework contains standard directories as follows

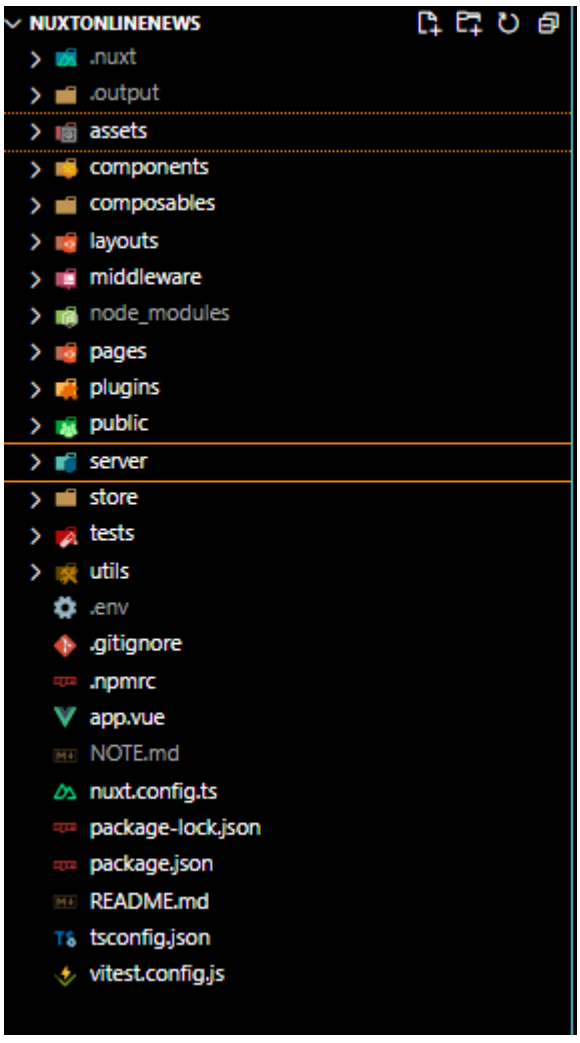

Figure 7. *Nuxt.js project structure*

The server, pages, components, layouts, and plugins folders store the main elements of the program. The server folder contains code that executes on the server-side during the

initial rendering process (SSR) or in response to client-side requests. The 'pages' directory in Nuxt.js is essential for defining the views and routes of the application. The components are Vue components located within the pages directory, which define the visual and structural content of my application's routes. The layouts directory in Nuxt.js plays a vital role in organizing and reusing UI components across multiple pages. The plugins directory in Nuxt.js serves as a central hub for integrating and utilizing third-party libraries, custom functionality, and plugins before the main Vue application initializes.

7.1.2 Server directory

The structure of the server is defined as follows:

Figure 8. *Nuxt.js server directory*

I organized the server directory into an api folder for the main REST API functions of the web application. Additionally, I created separate folders named model, plugin, and utils to provide support for the web app's REST API. Specifically, the model folder holds the schemas for the data collections used in the web app, while the plugins and utils folders contain middlewares that facilitate integration and enable the reusability of the web server functionality. Each handler for the server API is defined using the defineEventHandler() method

```
import PostModel from "~/server/models/posts";
import { tokenDecryption } from "~/server/utils/auth";

export default defineEventHandler(async (event) => {
  try {
    const body = await readBody(event);
    const auth = event.node.req.headers.authorization;
    // return { body, auth };
    if (!body || !auth) {
      setResponseStatus(event, 204);
      return { msg: "Request is wrong!" };
    } else {
      const token = auth.split(" ")[1];
      const userData = await tokenDecryption(token);
      if (userData) {
        const payload = {
          ...body,
          createdBy: {
            id: userData.id,
            username: userData.username,
            email: userData.email,
            role: userData.role,
          },
        };
        const newPost = await PostModel.create(payload);
        setResponseStatus(event, 201);
        return {
          msg: `The post is created with id: ${newPost._id}`,
        };
      } else {
        setResponseStatus(event, 204);
        return {
          msg: "Unauthorization!",
        };
      }
      //   return { token };
    }
  } catch (error) {
    return error;
  }
});
```

The server directory acts as the fundamental framework of the application, managing the essential logic and data operations. It acts as a bridge between the frontend and backend, processing requests from the frontend and delivering appropriate responses.

Within the server, three models: posts, staffPosts, and users, represent the different types of data stored in the application, allowing for efficient organization and retrieval of information.

To facilitate the operations on these models, controllers are setup, implementing the fundamental CRUD (Create, Read, Update, Delete) functionalities. These controllers are neatly organized within the api folder, ensuring a structured and easily maintainable codebase. Additionally, the server incorporates two specialized controllers dedicated to fetching external news data from other APIs, expanding the range of information available to the application.

To establish a seamless connection with the underlying database, a dbconnection middleware is placed in the plugins folder. This middleware ensures reliable and efficient interaction with the database, enabling smooth data transactions. Furthermore, the server employs hash and authentication middlewares, located in the utils folder, to enhance security and safeguard user data from unauthorized access.

```
// the middleare for database connection

import mongoose from "mongoose";

export default async () => {
  const config = useRuntimeConfig();
  try {
    const connectingOptions = {
      useNewUrlParser: true,
      // useFindAndModify: false,
      useUnifiedTopology: true,
    };
    await mongoose.connect(config.dbUrl, connectingOptions);
    console.log("The database is connected successfully!");
  } catch (error) {
    console.log(error);
  }
};
```

The server directory plays a vital role in managing data, processing requests, and ensuring the overall functionality and security of the application. It acts as the central hub for data manipulation and communication, contributing to a reliable backend infrastructure.

7.1.3 Pages

The pages directory in a Vue.js application plays a role in defining the routes and components that make up the frontend of the web app, enabling the creation of a dynamic

and user-friendly interface. The pages directory in a Vue.js application serves as a container for defining the routes and corresponding components that make up the frontend of the web app. Each folder within the pages directory represents a specific route, and the index.vue file within each folder defines the component to be rendered for that route. The index.vue file in the pages directory typically defines the home page of the web application. It contains the component that is in charge of displaying the content of the home page. In addition to the index.vue files, the pages directory may also contain dynamic pages, represented by files with names like [id].vue or [name].vue, where [id] or [name] represents a parameter that can be passed to the route. These dynamic pages enable the creation of routes that can display varied content based on the value of parameters.

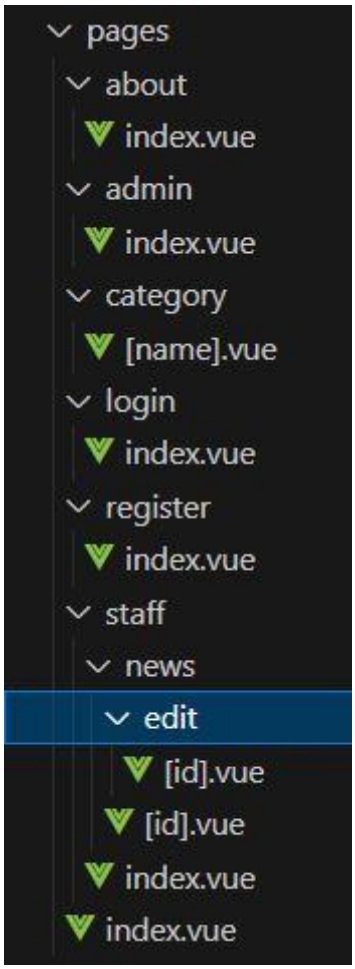

Figure 9. *Pages directory*

7.1.4 Components

The components directory stores the Vue components that will be used to display various web pages. These components include the main pages of the website, such as Home,

About, Admin dashboard, Category News, Login, Register, and Staff Dashboard. Each Vue component consists of three parts: template, script, and optional style. The template contains the HTML element tag with static content and variables, while the script contains JavaScript code that implements variables, methods, and packages of the component. The script retrieves data from the database, sends requests to the database, and displays data, facilitating dynamic and interactive user experiences. By splitting code into different components, code can be easily reused and modified throughout the project, speeding up the development process and ensuring a consistent user interface.

```
<template>
  <div>
    <async-home />
  </div>
</template>

<script setup>
import { defineAsyncComponent } from "vue";
definePageMeta({
  middleware: "auth",
});
const AsyncHome = defineAsyncComponent(() =>
  import("~/components/home/Home.vue")
);
</script>
<style scoped></style>
```

7.1.5 Layouts

The layouts subdirectory contains the main layouts of the application. Each layout is a separate file that defines the structure and design of a specific page or section of the application. It allows developers to define reusable templates that contain shared elements like headers, navigation menus, and footers, while page components focus on the unique content for each route.

The template includes a navigation bar at the top, which contains various menu items based on the user's authentication status. The useAuthStore hook is used to access the authentication store, and the storeToRefs function is used to retrieve reactive references from the store.

7.1.6 Plugins

Plugins folder contains 2 files as follows

The alert is defined as the Vue Alert package.

By importing and using the "vue-simple-alert" package as a plugin, the Vue application gains access to the features and components provided by the package. These features could include displaying customizable alerts, dialog boxes, or notifications within the Vue application.

Vuetify is the global configuration needed for the setting up UI of the client side.

7.1.7 Authentication

The authentication of the app is implemented with pinia store state in the following code

```
import { defineStore } from "pinia";

export const useAuthStore = defineStore("auth", {
  state: () => ({
    authenticated: false,
    loading: false,
  }),
  actions: {
    async authenticateUser({ username, password }) {
      const { data, pending } = await useFetch("/api/login", {
        method: "post",
        headers: { "Content-Type": "application/json" },
        body: {
          username,
          password,
        },
      });
      this.loading = pending;
      if (data.value) {
        const token = useCookie("token");
        token.value = data?.value?.token;
        this.authenticated = true;
      }
    },
    logUserOut() {
      const token = useCookie("token");
      token.value = null;
      this.authenticated = false;
    },
  },
});
```

The action authenticateUser({ username, password }) is responsible for authenticating the user. It receives an object with username and password as parameters. Inside the action, an asynchronous request is made to a login API endpoint using the useFetch function. The response is destructured to access data and pending properties. data.value is checked to determine if the authentication was successful. If it is, the user's token is

saved in a cookie using the useCookie function, and this.authenticated is assigned as true.

The action logUserOut() loges the user out. It clears the user's token stored in the cookie using the useCookie function and sets this.authenticated to false.

7.1.8 Data schema

Database system: The program uses MongoDB community for database system. There are 3 schemas defined by mongoose named post, staffPost, and user

Post schema

```
const postSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  sub: { type: String, required: true, trim: true },
  author: { type: String, required: true, trim: true },
  createdDate: { type: Date, default: new Date() },
  editDate: { type: Date, default: new Date() },
  content: { type: String, default: "" },
  img: { type: String, default: "" },
  urlToImage: { type: String, default: "" },
  url: { type: String, default: "" },
  source: { type: Object, required: true },
  createdBy: { type: Object, required: true },
});
```

Staff post schema

```
const staffPostSchema = new mongoose.Schema({
  title: { type: String, required: true, trim: true },
  sub: { type: String, required: true, trim: true },
  author: { type: String, required: true, trim: true },
  createdDate: { type: Date, default: new Date() },
  editDate: { type: Date, default: new Date() },
  content: { type: String, default: "" },
  img: { type: String, default: "" },
  urlToImage: { type: String, default: "" },
  url: { type: String, default: "" },
  source: { type: Object, required: true },
  createdBy: { type: Object, required: true },
});
```

User schema

```
const userSchema = new mongoose.Schema({
  firstName: { type: String, required: true, trim: true },
  lastName: { type: String, required: true, trim: true },
  email: { type: String, required: true, trim: true },
  username: { type: String, required: true, trim: true },
  password: { type: String, required: true },
  role: { type: String, required: true, trim: true },
});
```

7.2 App implementation in ASP.NET

7.2.1 Project structure

ASP.NET Web App is a web application framework created by Microsoft that adopts the design pattern known as model-view-controller (MVC), dividing an application into three fundamental components: Model, View, and Controller, along with optional elements such as Infrastructure, Migrations, or Repositories.

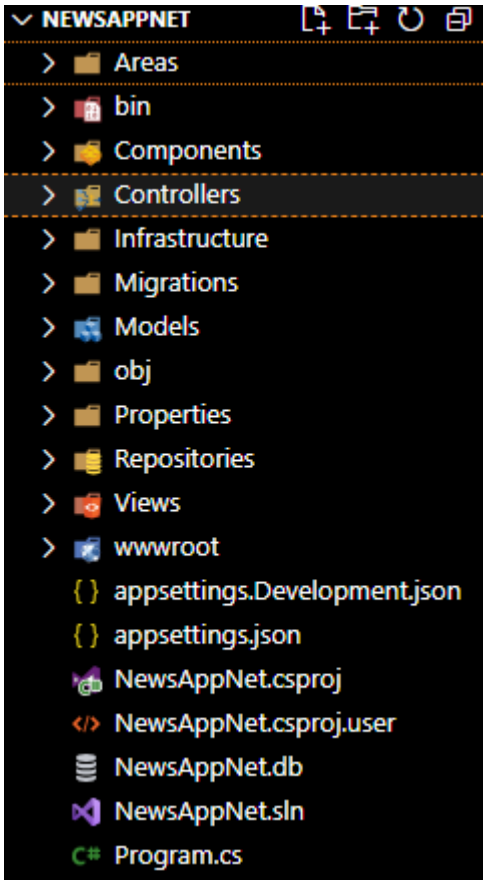

Figure 10. *ASP.NET project structure*

7.2.2 Model

In the context of ASP.NET, the Model component encompasses the data and logic of the application. This includes classes that represent various entities such as articles, external articles, users, posts, and listings. These classes would encapsulate the data and behavior of the application, such as external or own articles, list of articles, user, user role, and token information and preferences.

The Models directory contain data schemas including: BreakingNewsModels, FeedModels, OwnNewsModels, and ViewModels.

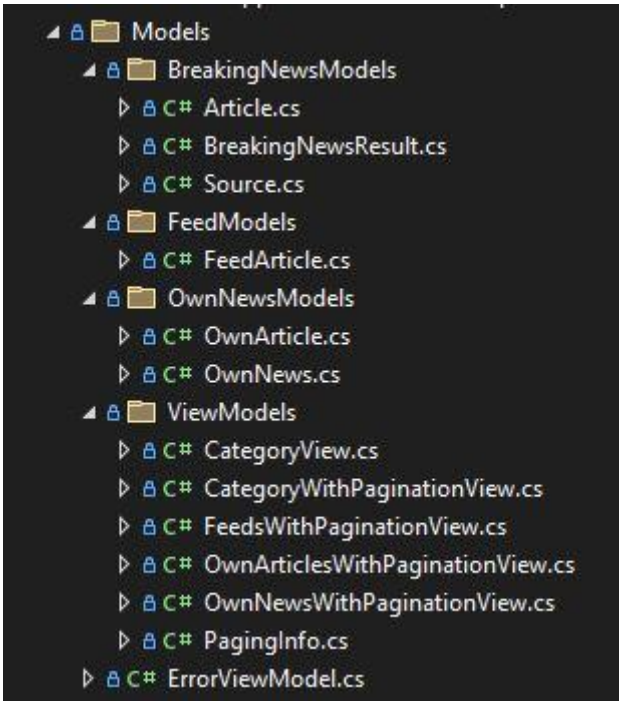

Figure 11. *Models directory*

7.2.3 View

View is responsible for rendering the user interface. In the case of APS.NET, View includes pages for displaying news, staff dashboard, user registration, user login, user log out, and managing the favorite news. View files use the data passed from Controller to display the appropriate interface to the user.

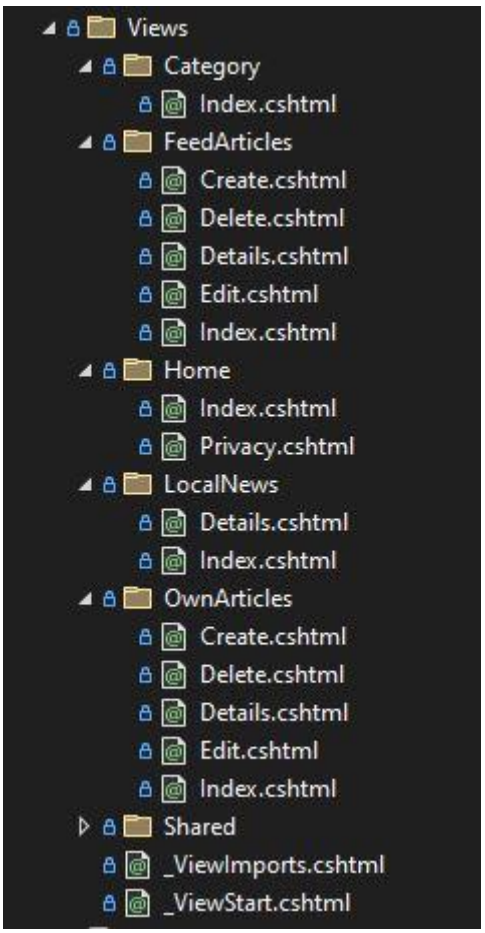

Figure 12. *Views directory*

7.2.4 Controllers

The controller plays a crucial role in handling user interactions and directing the application's flow and holds controllers of Views in the App. The Controller manages user input, engages with the Model to execute required operations, and then passes the data to the View for rendering. In the case of APS.NET, the Controller could handle user input from the login and registration forms, as well as requests to view practice listings, search for practices, and manage user accounts.

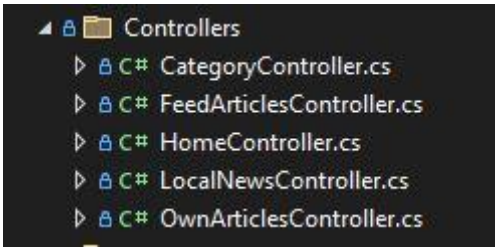

Figure 13. *Controllers directory*

Home Controller class is responsible for handling Home view:

```
using Microsoft.AspNetCore.Mvc;
using NewsAppNet.Models;
using System.Diagnostics;

namespace NewsAppNet.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Cur-
rent?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

This controller class manages requests for the home page (Index and Privacy actions) as well as error pages (Error action) within the ASP.NET Core application. It also uses a logger to log messages specific to the HomeController.

7.2.5 Infrastructure directory

The Infrastructure folder contains classes that can be reusable in other components.

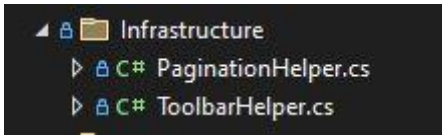

Figure 14. *Infrastructure directory*

Paginationhelp.cs is a class to generate a pagination resuable tags in html pages

ToobarHelper.cs can build dynamic toolbar with declared attributes

7.2.6 Repository

The repositories are classes that serve as an intermediary layer between the application logic and the data access layer.

7.2.7 Authentication

The Identity folder stores classes used for identify the users.

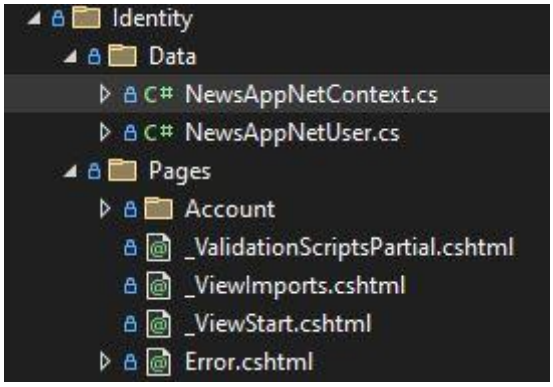

Figure 15. *Identity directory for authentication*

The NewsAppNetContext class serves as the database context for the application. It provides access to the tables in the database through DbSet properties and allows customization of the ASP.NET Identity model. It inherits from IdentityDbContext<NewsAppNetUser>, which provides support for the identity system in ASP.NET Core, including user authentication, authorization, and user management.

The NewsAppNetUser class allows additional profile data to be associated with application users. By adding properties for first name, last name, and role, the class extends the functionality provided by the IdentityUser class to include custom user data.

7.2.8 Data migration

Database migration in ASP.NET projects is the action of moving data from an existing database to a new database, usually as part of an upgrade or migration to a new version of the application. This process is necessary when the existing data needs to be transformed or updated to be compatible with the new version of the application, or when the data needs to be moved to a different database or storage system. Data migrations encompass the outcomes of processes that allow the application to handle changes to the database structure in a controlled and incremental fashion.

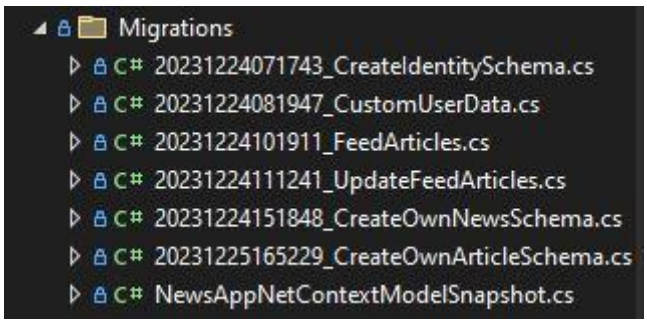

Figure 16. *Data migration*

8. RESULT

8.1 Benchmarks

8.1.1 Loading Time, Scripting Time, Rendering Time, Painting Time, System Time

The numbers measured represents the time taken for each phase of the web page's lifecycle. Lower values generally indicate better performance.

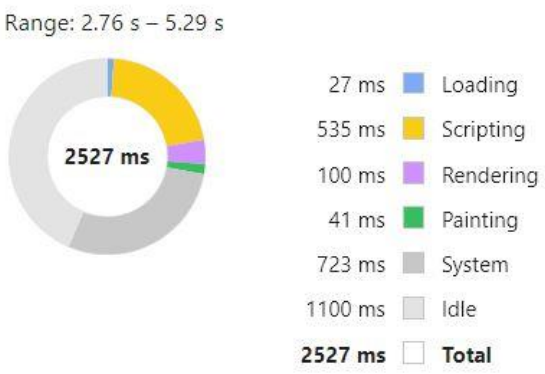

Figure 17. *Nuxt.js application*

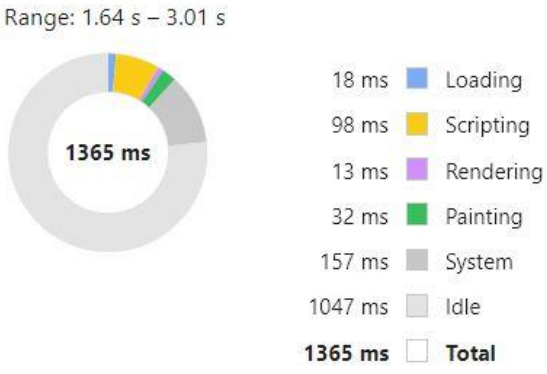

Figure 18. *ASP.NET application*

Loading Time: ASP.NET performs better in the loading phase, taking only 18 ms compared to Nuxt.js's 27 ms. This suggests that ASP.NET loads the necessary resources faster.

Scripting Time: ASP.NET has significantly better scripting performance, taking only 98 ms compared to Nuxt.js's 535 ms. This indicates that ASP.NET executes JavaScript code more efficiently.

Rendering Time: ASP.NET outperforms Nuxt.js in the rendering phase, taking only 13 ms compared to Nuxt.js's 100 ms. This suggests that ASP.NET renders the HTML content more quickly.

Painting Time: ASP.NET has a slightly better painting performance, taking 32 ms compared to Nuxt.js's 41 ms. This refers to the time taken to render the visual content on the screen.

System Time: ASP.NET performs better in the system phase, taking 157 ms compared to Nuxt.js's 723 ms. This includes any system-level tasks or overhead involved in processing the web page.

8.1.2 First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, Speed Index

These metrics provide insights into the loading and rendering performance of the web pages. Lower values suggest better performance.

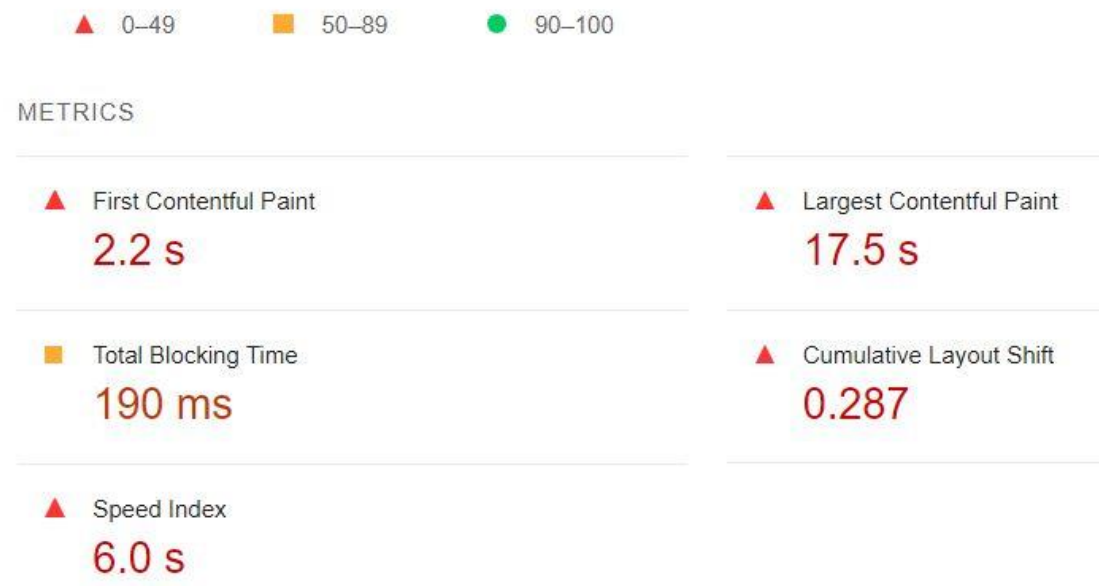

Figure 19. *Nuxt.js application*

Figure 20. *ASP.NET application*

First Contentful Paint: ASP.NET achieves a faster first paint time, taking only 0.7 s compared to Nuxt.js's 2.2 s. This means that ASP.NET displays the first content on the screen more quickly.

Largest Contentful Paint: ASP.NET outperforms Nuxt.js significantly in terms of the largest content paint time. ASP.NET takes 2.2 s, whereas Nuxt.js takes 17.5 s. This indicates that ASP.NET renders the most significant visual element on the screen faster.

Total Blocking Time: ASP.NET has no blocking time, while Nuxt.js has a blocking time of 190 ms. This metric measures the cumulative duration when the main thread was blocked and unable to handle user input. Lower values are better.

Cumulative Layout Shift: Both ASP.NET and Nuxt.js have a cumulative layout shift of 0, indicating that there were no unforeseen alterations in the layout during the loading process of the page. This metric measures the visual stability of the page.

Speed Index: ASP.NET achieves a faster speed index of 1.1 s compared to Nuxt.js's 6.0s. The speed index evaluates the rate at which the elements of a page become visually populated. A lower value indicates faster visible loading.

8.1.3 Performance, Accessibility, Best Practices, SEO

The results are scored using a numerical scale ranging from 0 to 100, with higher values indicating superior performance.

Figure 21. *Nuxt.js application*

Figure 22. *ASP.NET application*

Performance: ASP.NET achieves a significantly higher performance score of 89 compared to Nuxt.js's 37. This indicates that ASP.NET has better overall performance optimizations.

Accessibility: Both ASP.NET and Nuxt.js have high accessibility scores, with ASP.NET at 95 and Nuxt.js at 77. This means that both frameworks have implemented accessible features, but ASP.NET performs better in this aspect.

Best Practices: ASP.NET and Nuxt.js have similar best practices scores, with ASP.NET at 86 and Nuxt.js at 95. This metric evaluates adherence to industry best practices for web development.

SEO: Both ASP.NET and Nuxt.js have the same SEO score of 80. This metric assesses how well the framework supports search engine optimization techniques.

8.1.4 Size of generated code

ASP.NET: The generated code for ASP.NET is larger, with a size of 74 MB. This can be attributed to the features and functionalities provided by ASP.NET, including its extensive framework and libraries.

Nuxt.js: On the other hand, the generated code for Nuxt.js is smaller, with a size of 13.6 MB. Nuxt.js is a lightweight framework that emphasizes optimization techniques.

8.2 Features

8.2.1 Cost

When considering the cost aspect, both Nuxt.js and ASP.NET have their own advantages and considerations. Nuxt.js, being open-source, eliminates licensing fees and allows for the utilization of JavaScript developers, potentially reducing development

costs. It also offers fast development with its tools and libraries and has a large community for support. Nevertheless, it can be challenging to learn and may necessitate developers who are already familiar with Vue.js and modern JavaScript tooling. Additionally, some custom functionalities might rely on paid Nuxt.js modules, which could increase costs.

On the other hand, ASP.NET, backed by Microsoft, is a mature and stable framework with extensive documentation. It has a wide talent pool of experienced ASP.NET developers and is suitable for complex applications. Integration with other Microsoft technologies is seamless. However, it requires commercial licensing for development and deployment, increasing costs. The development curve may be steeper due to its more complex syntax and development environment compared to Nuxt.js. Moreover, it is tightly coupled with Microsoft technologies, potentially resulting in vendor lock-in and expensive framework switching.

In terms of deployment and hosting costs, Nuxt.js provides flexibility as it can be deployed on various platforms like AWS, Google Cloud, Vercel, or Netlify. The open-source nature and wider platform choices might offer cost-effective hosting options. On the other hand, ASP.NET is ideally deployed on Microsoft Azure for optimal performance and integration. However, Azure pricing can be higher compared to other hosting providers.

Regarding maintenance costs, Nuxt.js benefits from an active community for bug fixes and solutions. The frequent updates of Nuxt.js might require occasional code migrations, incurring maintenance costs. ASP.NET, being a stable platform, generally has less frequent updates, resulting in lower maintenance overhead. Additionally, Microsoft offers paid support plans for guaranteed assistance.

8.2.2 Development Complexity

Nuxt.js, being JavaScript-based, offers a familiar syntax for many developers and promotes component-based development. It has a large community and abundant resources for learning, and it comes with an easier initial learning curve for beginners.

On the other hand, ASP.NET, with its mature and established C# language, provides stability and extensive documentation. It follows an object-oriented approach and has a wide talent pool of experienced developers, but it is quite a more difficult initial step for developers compared to modern JavaScript frameworks.

It's also important to consider the development workflow and tooling. Nuxt.js utilizes modern tooling and features like Hot Module Replacement (HMR), while ASP.NET offers

mature development tools and built-in deployment tools. However, Nuxt.js may have potential configuration complexity and require keeping up with rapid updates, while ASP.NET may rely heavily on IDEs and becomes easier for maintenance. At the same time, this advantage of ASP.NET framework may have limited flexibility for custom tooling.

8.2.3 Error Handling

When it comes to error handling, Nuxt.js and ASP.NET have different approaches depending on whether the focus is on the client-side or the server-side.

Nuxt.js primarily handles errors on the client-side using JavaScript's try...catch blocks and Vue's error handling mechanisms. It utilizes middleware for global error handling and error boundaries for component-level isolation. For server-side error communication, it relies on API error responses.

On the other hand, ASP.NET puts emphasis on server-side error handling. It offers built-in error handling features within the server-side framework, using exception handling (try...catch blocks) and custom error pages for centralized error management. It provides a structured way to log errors and view detailed error logs. Additionally, ASP.NET allows error details to be sent to the client-side for UI updates and supports client-side validation for a more interactive experience.

8.2.4 Security

In term of security, there are some differences between Nuxt.js and ASP.NET.

Nuxt.js is built on JavaScript and Node.js, which means it inherits potential security vulnerabilities associated with the ecosystem, such as prototype pollution and code injection. As a frontend framework, Nuxt.js primarily focuses on client-side security, so additional measures are required for server-side protection. Security in Nuxt.js heavily relies on the community and third-party libraries/modules, which can vary in quality and vulnerability management.

On the other hand, ASP.NET is built on the .NET Framework and benefits from Microsoft's security focus and regular updates for known vulnerabilities. It is a full-stack framework that offers built-in security features such as input validation, authorization, and role-based access control. ASP.NET has an extensive resources, including documentation, support, and security best practices. However, configuring security in ASP.NET can be complex, requiring developers to have a good understanding of .NET security concepts.

8.2.5 Tools

Nuxt.js utilizes core languages such as JavaScript (ES6+) and Vue.js. The build tools are Webpack and Babel, and the package manager can be either npm or yarn. ESLint is used for linting. Styling options include CSS preprocessors like Sass, Less, and Stylus, as well as CSS-in-JS libraries. Popular IDEs for Nuxt.js development are Visual Studio Code, WebStorm, and Atom.

ASP.NET core languages are C# and Razor syntax. The primary development environment is Visual Studio, although Visual Studio Code can also be used. NuGet is the package manager for ASP.NET. StyleCop is used for linting, and styling can be done with CSS, LESS, and Sass. ASP.NET provides built-in support for database integration using Entity Framework and ADO.NET.

8.2.6 Community support

Nuxt.js has an active open-source community. It has gained popularity among developers, especially those working with JavaScript and Vue.js. The community actively contributes to the framework by developing plugins, modules, and sharing knowledge through forums, GitHub repositories, and online communities. Nuxt.js also has an extensive documentation and a dedicated Discord channel where developers can seek help and engage with other community members.

On the other hand, ASP.NET has a strong community support backed by MicrosoftDue to its widespread adoption, ASP.NET has a large community of developers who actively engage in forums, Stack Overflow, and Microsoft Q&A to seek and offer assistance. Microsoft itself provides extensive documentation, tutorials, and resources for ASP.NET development. Additionally, there are numerous blogs, online communities, and user groups that focus on ASP.NET, allowing developers to connect, share knowledge, and seek support.

9. DISCUSSION

Overall, the thesis successfully addresses all of the research questions by employing a systematic comparison between the ASP.NET and Nuxt.js frameworks.

The research answers the first research question (RQ1), which involves the criteria for comparing the two frameworks in terms of benchmarks and features. To evaluate the benchmark criteria, various metrics such as Loading Time, Scripting Time, Rendering Time, Painting Time, System Time, First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, Speed Index, Performance, Accessibility, Best Practices, SEO, and Size of generated code are considered. On the other hand, the feature criteria like Cost, Development Complexity, Error Handling, Security, Tools, and Community support are evaluated based on the actual development experience with the chosen frameworks. This approach ensures that the assessment takes into account the practical aspects of working with ASP.NET and Nuxt.js.

For Loading Time, Scripting Time, Rendering Time, Painting Time, and System Time, ASP.NET demonstrates better results compared to Nuxt.js. It has faster loading, scripting, rendering, painting, and system times, indicating better efficiency and responsiveness. Although the difference between MongoDB and SQLite can indeed have an impact on the performance of Nuxt.js and ASP.NET applications in this case, it is important to note that in practical industry usage, MongoDB is commonly used for Nuxt.js applications, while SQL is commonly used for ASP.NET applications. Comparing the performance of Nuxt.js and ASP.NET applications using MongoDB and SQLite respectively makes more sense than attempting to use a single database platform for both, especially considering that it is uncommon for anyone to utilize such an approach.

Based on the result obtained, ASP.NET demonstrates better web Performance and Accessibility compared to Nuxt.js. .NET achieves faster First Contentful Paint, Largest Contentful Paint, Total Blocking Time, and Speed Index. However, both frameworks have the same score in SEO and a similar cumulative layout shift, indicating stable page rendering.

In term of project size, ASP.NET and Nuxt.js differ in their approach to web development, which results in variations in the size of the generated code. ASP.NET is a comprehensive web framework that provides an extensive array of features and functionalities, making it a potent tool for developing robust server-side applications. However, this comprehensive nature leads to a larger size of the generated code compared to Nuxt.js. This is

because ASP.NET includes various components, libraries, and infrastructure to support web development, resulting in additional code for handling server-side rendering, backend capabilities, data access, security, and more. On the other hand, Nuxt.js is primarily focused on providing essential frontend functionalities without the extensive backend capabilities offered like ASP.NET. It aims to simplify the development of server-rendered Vue.js applications.

When evaluating cost, Nuxt.js is typically more budget-friendly for smaller projects because of its open-source nature, broader platform support, and potentially reduced development expenses. However, rapid updates and potential vendor lock-in for custom functionalities might increase maintenance costs. ASP.NET is a suitable choice for enterprise-level applications that require stability, security, and seamless integration with Microsoft technologies. However, the cost for development and hosting can potentially be higher.

For development complexity, the selection between ASP.NET and Nuxt.js should depend on the team's skillset, project goals, and preferred development style as both frameworks have their own advantages and drawbacks. Nuxt.js is a JavaScript-based framework that offers familiarity and promotes component-based development, so it has an easier learning curve for developers. ASP.NET, with its mature C# language, provides stability and extensive documentation, but it can be a more difficult approach for beginners. The choice also depends on the development workflow and tooling preferences. Nuxt.js has modern tooling and features but may have configuration complexity, while ASP.NET offers mature tools and built-in deployment but may have limited flexibility for custom tooling.

In terms of error handling, Nuxt.js provides flexibility for client-side handling, whereas ASP.NET offers a more structured server-side approach. The decision between client-side and server-side error handling relies on the needs and preferences of the project. Nuxt.js is well-suited for applications where quick and interactive error feedback is desired, while ASP.NET provides a more structured and centralized approach for error handling.

In relation to security, ASP.NET generally provides a more robust security foundation with its built-in features, regular updates, and .NET security model. Nuxt.js itself, on the other hand, does not provide specific security features. It requires developers to implement security measures based on their chosen backend, libraries, and development practices.

Moreover, both ASP.NET and Nuxt.js utilize a range of powerful tools and technologies to facilitate web development. These toolsets allow ASP.NET and Nuxt.js frameworks empower developers to build robust, scalable, and secure web applications efficiently.

In term of community support, both Nuxt.js and ASP.NET have strong communities that contribute to their respective frameworks. Nuxt.js has an active open-source community with dedicated channels for support and knowledge sharing, while ASP.NET benefits from a dynamic community supported by Microsoft, with extensive documentation and a wide range of platforms for community engagement.

For the second research question (RQ2), the thesis utilizes tools such as Lighthouse and the Inspect option in Google Chrome to measure Loading Time, Scripting Time, Rendering Time, Painting Time, System Time, First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, Speed Index, Performance, Accessibility, Best Practices, and SEO. These tools enable accurate measurement and assessment of the benchmarks, providing valuable insights into the loading performance of each framework. The Loading Time, Scripting Time, Rendering Time, Painting Time, System Time represents the time taken in milisecond for each phase of the web page's lifecycle. Lower values generally indicate better performance. However, the First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, Speed Index measured in second provide observations into the rendering performance of the web pages. Lower values suggest better performance. For Performance, Accessibility, Best Practices, and SEO, the results were scored using a numerical scale ranging from 0 to 100, with higher values indicating superior performance.

To compare the feature criteria like Cost, Development Complexity, Error Handling, Security, Tools, and Community support, an approach taking into account the practical aspects of working with Nuxt.js and ASP.NET was used to provide an understanding of their capabilities, ease of use, and suitability for different project requirements.

By conducting a thorough comparison and analyzing the obtained results, the thesis is able to provide a recommendation between the two platforms, addressing the third research question (RQ3). This recommendation will consider the strengths and weaknesses of each framework, allowing the development team to make an informed decision on which framework is more likely to yield better results for their specific project. When deciding between Nuxt.js and ASP.NET for a project, several crucial factors must be taken into account. The expertise of the development team, the size and complexity of the project, the desired level of development control, the security requirements, and the community support considerations all play a significant role in making the best choice.

Nuxt.js is more suitable for simpler projects and rapid development, while ASP.NET Core provides scalability and flexibility for complex enterprise applications.

10. CONCLUSION

Based on the analysis, both frameworks, Nuxt.js and ASP.NET, meet the demands of most projects and are suitable for projects ranging from small to large scale. The choice of framework should align with the team's expertise in JavaScript or C#, as familiarity with the programming language facilitates learning and working with.

When considering project size and complexity, Nuxt.js may be more agile and suitable for smaller projects, while ASP.NET's structure and tooling are well-equipped to handle larger, enterprise-level applications.

For development control, Nuxt.js offers more flexibility and customization options, allowing developers to have greater control over the project. On the other hand, ASP.NET provides a structured framework with built-in features, which can be beneficial for engineers who prefer a more standardized approach.

In terms of security, ASP.NET generally offers a more robust security foundation. It has built-in features, a strong .NET security model, and regular updates, making it a dependable option for applications that emphasize security.

Considering performance as a critical criterion, ASP.NET tends to provide better performance compared to Nuxt.js. This can be attributed to various factors, including optimizations within the framework and the underlying technologies it utilizes.

Therefore, the choice between Nuxt.js and ASP.NET should consider factors such as team expertise, project size and complexity, development control, security requirements, and performance considerations to yield the best results for their specific project.

REFERENCES

[1] Singleton JL, Leavens GT. Verily: a web framework for creating more reasonable web applications. Companion Proc. 36th Int. Conf. Softw. Eng., Hyderabad India: ACM; 2014, p. 560–3. https://doi.org/10.1145/2591062.2591069.

[2] Swacha J, Kulpa A. Evolution of Popularity and Multiaspectual Comparison of Widely Used Web Development Frameworks. Electronics 2023;12:3563. https://doi.org/10.3390/electronics12173563.

[3] Curie DH, Jaison J, Yadav J, Fiona JR. Analysis on Web Frameworks. J Phys Conf Ser 2019;1362:012114. https://doi.org/10.1088/1742-6596/1362/1/012114.

[4] Duldulao DB. ASP.NET Core and Vue.js: build real-world, scalable, full-stack applications using Vue.js 3, TypeScript, .NET 5, and Azure. Birmingham, UK: Packt Publishing; 2021.

[5] Kok LT. Learn Nuxt.js web development hands-on server-side web development with Vue.js, Vuex, and Nuxt. Place of Publication Not Identified: Packt Publishing; 2020.

[6] Eseme S. Architecting Vue.js 3 enterprise-ready web applications: build and deliver scalable and high-performance, enterprise-ready applications with Vue and JavaScript. 1st edition. Birmingham, UK: Packt Publishing Ltd.; 2023.

[7] Pasquali S, Faaborg K. Mastering Node.js: build robust and scalable real-time server-side web applications efficiently. Second edition. Birmingham, UK: Packt Publishing; 2017.

[8] Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Comput 2010;14:80–3. https://doi.org/10.1109/MIC.2010.145.

[9] Shah D. Node JS guidebook. Place of Publication Not Identified: BPB Publications; 2018.

[10] Mead A. Advanced Node.js Development: Master Node.js by building real-world applications. Birmingham: Packt Publishing; 2018.

[11] Yang H. Vue. JS framework: design and implementation. 1st ed. 2023. Singapore: Springer; 2023.

[12] Macrae C. Vue.js: Up and Running: Building Accessible and Performant Web Apps. Place of Publication Not Identified: O'Reilly Media : O'Reilly Media; 2018.

[13] Imsirovic A. Vue. js Quick Start Guide: Learn How to Build Amazing and Complex Reactive Web Applications Easily Using Vue. js. Birmingham: Packt Publishing Ltd; 2018.

[14] Pšenák P, Tibenský M. The usage of Vue JS framework for web application creation. Mesterséges Intell 2020;2:61–72. https://doi.org/10.35406/MI.2020.2.61.

[15] Freeman A. Pro ASP.NET Core MVC. Sixth edition. New York, NY: Apress; 2016.

[16] Rozaliuk T, Kopyl P, Smołka J. Comparison of ASP.NET Core and Spring Boot ecosystems. J Comput Sci Inst 2022;22:40–5. https://doi.org/10.35784/jcsi.2794.

[17] Beasley RE. Essential ASP.NET Web Forms Development: Full Stack Programming with C#, SQL, Ajax, and JavaScript. 1st edition. Berkeley, Calif.: Apress; 2020.

[18] Esposito D. Programming ASP.NET Core. Redmond, Wash: MicrosoftPress; 2018.

[19] Ragupathi MTS, Sanctis VD, Singleton J. ASP.NET Core. Birmingham: Packt Publishing; 2017.

[20] Tosato A, Minerva M, Bartolesi E. Mastering Minimal APIs in ASP.NET Core Build, Test, and Prototype Web APIs Quickly Using . NET and C#. Birmingham: Packt Publishing, Limited; 2022.

[21] Strauss D. Creating ASP. NET core web applications: proven approaches to application design and development. Berkeley, CA: Apress; 2021.

[22] Pajo B. Introduction to research methods: a hands-on approach. Second edition. Thousand Oaks: SAGE Publications, Inc.; 2022.

[23] Qualitative versus Quantitative Research. S.l.: IntechOpen; 2017.

[24] Taylor GR. Integrating quantitative and qualitative methods in research. 2nd ed. Lanham, MD: University Press of America; 2005.

[25] Creswell JW, Creswell JD. Research design: qualitative, quantitative, and mixed methods approaches. Sixth edition. Los Angeles: SAGE; 2023.

[26] Győrödi CA, Dumşe-Burescu DV, Zmaranda DR, Győrödi RŞ. A Comparative Study of MongoDB and Document-Based MySQL for Big Data Application Data Management. Big Data Cogn Comput 2022;6:49. https://doi.org/10.3390/bdcc6020049.

[27] SHARMA M. MONGODB COMPLETE GUIDE develop strong understanding of administering mongoDB, commands, MongoDB compass, MongoDB sesrver, MongoDB replication and MongoDB sharding. S.l.: BPB PUBLICATIONS; 2021.

[28] Phaltankar A, Ahsan J, Harrison M, Nedov L. MongoDB Fundamentals: a Hands-On Guide to Using MongoDB and Atlas in the Real World. Birmingham: Packt Publishing; 2020.

[29] Győrödi CA, Dumşe-Burescu DV, Zmaranda DR, Győrödi RŞ, Gabor GA, Pecherle GD. Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application's Data Storage. Appl Sci 2020;10:8524. https://doi.org/10.3390/app10238524.

[30] Giamas A. Mastering MongoDB 6.x: expert techniques to run high-volume and fault-tolerant database solutions using MongoDB 6.x. Third edition. Birmingham, UK: Packt Publishing Ltd.; 2022.

[31] Yu Y. Embedded Internet of Things Applications of SQLite Based on WinCE Mobile Terminal. In Review; 2020. https://doi.org/10.21203/rs.3.rs-37411/v2.

[32] Owens M. The Definitive Guide to SQLite. Berkeley, CA: Apress; 2006. https://doi.org/10.1007/978-1-4302-0172-4.

[33] Taylor A. Creating a Database with SQLite. 1st edition. Apress; 2018.

[34] Agus Kurniawan K. Python and SQLite Development. Lulu; 2021.

[35] Learn SQLite with Python: building database-driven desktop projects. First edition. S.l: Sparta Publishing; 2019.

APPENDIX

Nuxt.js application dependency package utilization

	Name	Version
1	@mdi/font	7.3.67
2	@vitejs/plugin-vue	5.0.0
3	axios	1.5.1
4	bcrypt	5.1.1
5	bcryptjs	2.4.3
6	eslint-plugin-vitest	0.3.20
7	jose	4.15.2
8	jsdom	23.0.1
9	mongoose	7.6.0
10	vue-material-design-icons	5.2.0
11	vue-simple-alert	1.1.1
12	mdi/js	7.3.67
13	@nuxt/devtools	latest
14	@nuxt/test-utils	3.9.0
15	@pinia/nuxt	0.5.1
16	happy-dom	12.10.3
17	nuxt	3.7.4
18	pinia	2.1.7
19	playwright-core	1.40.1
20	sass	1.69.4
21	vite-plugin-vuetify	1.0.2
22	vitest	1.1.0
23	vue	3.3.4
24	vue-router	4.2.5

25	vuetify	3.4.0-alpha.1

ASP.NET application dependency package utilization

	Name	Version
1	Microsoft.AspNetCore.Components.QuickGrid.EntityFrameworkAdapte	8.0.0
2	Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore	8.0.0
3	Microsoft.AspNetCore.Identity.EntityFrameworkCore	8.0.0
4	Microsoft.AspNetCore.Identity.UI	8.0.0
5	Microsoft.EntityFrameworkCore.Sqlite	8.0.0
6	Microsoft.EntityFrameworkCore.Tools	8.0.0