Utilizing the Blazor Framework for Developing a New Version of Company Application

Maria Shamoeva

Bachelor's thesis
2024

Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Maria Shamoeva**
Osobní číslo:	**A21250**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Využití frameworku Blazor pro tvorbu nové verze podnikové aplikace**
Téma práce anglicky:	**Utilizing the Blazor Framework for Developing a New Version of Company Application**

Zásady pro vypracování

1. Popište současný stav technologií pro tvorbu dané aplikace.
2. Zaměřte se na framework Blazor a související technologie.
3. Navrhněte změny ve stávající aplikaci.
4. Realizujte vývoj navržených změn a popište klíčové části řešení.
5. Zhodnoťte dosažené výsledky a možnosti dalšího rozvoje aplikace.

Forma zpracování bakalářské práce:	**tištěná/elektronická**
Jazyk zpracování:	**Angličtina**

Seznam doporučené literatury:

1. PRICE, Mark J., 2022. Apps and Services with .NET 7. Packt Publishing. ISBN 9781801813433.
2. Blazor, A Beginners Guide [online]. Progress Software Corporation, 2020 [cit. 2023-11-11]. Dostupné z: https://www.dbooks.org/blazor-a-beginners-guide-5635532311/read/
3. ENGSTROM, Jimmy a Jeff FRITZ. Web Development with Blazor. Second Edition. Packt Publishing, 2023. ISBN 1803241497.
4. Blazorise Documentation. Online. Blazorise Docs. Dostupné z: https://blazorise.com/docs. [cit. 2023-11-11].
5. ASP.NET Core Blazor. Online. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0. [cit. 2023-11-11].
6. ChilliCream. Online. Dostupné z: https://chillicream.com/docs/hotchocolate/v13. [cit. 2023-11-11].
7. C# documentation. Online. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/. [cit. 2023-11-11].
8. .NET documentation. Online. Dostupné z: https://learn.microsoft.com/en-us/dotnet/. [cit. 2023-11-11].

Vedoucí bakalářské práce:	**doc. Ing. Radek Šilhavý, Ph.D.** Ústav počítačových a komunikačních systémů
Datum zadání bakalářské práce:	**5. listopadu 2023**
Termín odevzdání bakalářské práce:	**13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.
- I declare that I have used the generative AI model tool Chat-GPT; https://chatgpt.com/ to create this work for the purpose of translation,rephrasing and editing the text. After using this tool, I have checked the content and take full responsibility for it.

In Zlín; dated: 13.05.2024

Maria Shamoeva,v.r.
Student´s Signature

ABSTRAKT

Tato bakalářská práce se zabývá strategickým přijetím a implementací frameworku Blazor v kontextu vývoje moderních webových aplikací ve společnosti Continental Barum s.r.o. Práce komplexně zkoumá architekturu Blazoru včetně jeho knihoven komponent a hodnotí integraci souvisejících technologií. Teoretická část práce ukazuje základní funkcionality a výhody použití Blazor v aplikacích podnikové úrovně. V analytické části se pozornost přesouvá na reálnou aplikaci těchto technologií ve společnosti Continental Barum s.r.o., kde je podrobně popsána migrace ze staršího webového frameworku na Blazor. Praktické poznatky jsou věnovány redesignu podnikové webové aplikace SFE a ilustrují zlepšení v oblasti webových aplikací. V závěru práce jsou reflektovány dosažené výsledky a navržen další vývoj.

Klíčová slova: Blazor,C#, .NET, ASP.NET,GraphQL

ABSTRACT

This bachelor's thesis explores the strategic adoption and implementation of the Blazor framework within the context of modern web application development at Continental Barum s.r.o. This thesis provides a comprehensive examination of Blazor's architecture, including its component libraries and assesses the integration of related technologies.The theoretical part of the thesis outlines the core functionalities and advantages of using Blazor in enterprise-level applications. In the analysis section, the focus shifts to the real-world application of these technologies at Continental Barum s.r.o., detailing the migration from an older web framework to Blazor . Practical insights are provided into the redesign of the company's SFE Web App, illustrating improvements in web application.Finally, the thesis concludes by reflecting on the achieved results and proposing further developments.

Keywords: Blazor,C#, .NET, ASP.NET,GraphQL

ACKNOWLEDGEMENTS

Acknowledgements, motto and a declaration of honour saying that the print version of the Bachelor's/Master's thesis and the electronic version of the thesis deposited in the IS/STAG system are identical, worded as follows:

I hereby declare that the print version of my Bachelor's/Master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

CONTENTS

INTRODUCTION

The speed at which technology is developing in today's ever changing digital environment is astonishing. Web development, in particular, is a field characterized by continual innovation and the swift obsolescence of older technologies. As the needs of users and the capabilities of hardware advance, old web frameworks often find themselves unable to keep up, leading to their gradual decline and eventual replacement by more modern solutions.Technology is advancing at an incredible pace in today's digital world. Web development, in particular, is a field that is constantly innovating and quickly discarding older technologies. With the increasing demands of users and the advancements in hardware capabilities, older web frameworks often struggle to keep up, leading to their gradual decline and eventual replacement by more modern solutions. This bachelor thesis explores the integration and utilization of Blazor in a new version of company’s web application.

I. THEORY

1 INTRODUCTION TO BLAZOR

Nowadays, .NET developers have a lot of tools and resources to create applications for various operating systems such as Windows, Linux, iOS, Android, and macOS. We can build robust and scalable web-based applications with the help of ASP.NET MVC, Razor Pages, and Web API. However, a missing piece in the puzzle has always been the ability to write web applications using the power of C# and .NET. JavaScript has always occupied this domain. However, things have changed now, and there is a new framework called Blazor, which can help us write client-side web applications using C# and .NET. Blazor will be introduced in this chapter.

1.1 Overview of Blazor

First time Blazor was introduced during NDC Oslo in 2017. This first version was constructed using DotNetAnywhere, an interpreted .NET CIL runtime. While Blazor's features were limited, its promise was evident. Microsoft introduced Blazor to its ASP.NET GitHub organization after its initial demonstration as an experimental project. This move indicated Microsoft's serious consideration of Blazor's potential. As of now, the repository has garnered over 3,300 stars and contributions from 17 developers, further validating its growing popularity. Then the ASP.NET team rewrote Blazor from scratch as part of the official adoption. DotNetAnywhere has been superseded by Mono, which provides a considerably more powerful and feature-rich .NET runtime. [1]

With an innovative framework that combines productivity and power, Blazor is changing the face of web development. It uses the .NET Core architecture to standardize development stack-wide patterns and procedures. Blazor offers developers flexibility and the convenience of switching between hosting modes as needed, thanks to the versatility o of .NET Core, which supports client- and server-side hosting models. The framework combines dependency injection, configuration, routing, and other crucial .Net Core technologies with the ease of use of Razor. Blazor assures consistency with .NET norms and improves compatibility with existing tooling by incorporating best practices from popular JavaScript frameworks such as Angular and React and combining them with Razor templates.[2]

Blazor, a framework that supports responsive design and mobile-first applications, makes creating web applications that run on various devices and screen sizes easier. This versatility is critical in today's development environment, where mobile usage continues to grow.

Blazor makes constructing interactive and responsive user interfaces easier by allowing developers to use C#, which many people are familiar with because of its widespread use in application development. Blazor's interaction with CSS frameworks such as Bootstrap improves its ability to develop responsive web apps that function effectively on mobile and desktop platforms. This approach streamlines the development process and meets the industry's need for faster, more efficient delivery of software solutions that function seamlessly across various platforms.[3]

Blazor has strong Microsoft support and an active development community, which work together to create a rich resource of instructional materials. Furthermore, community-generated content, such as thorough blog pieces, free YouTube tutorials, and forums like Stack Overflow and the official Blazor community on GitHub, provides practical insights and troubleshooting suggestions for developers of all skill levels. [3]

The tooling environment surrounding Blazor dramatically improves the development experience. IDEs like Visual Studio and Visual Studio Code provide excellent Blazor support, including built-in templates, debugging tools, and extensions tailored exclusively to Blazor development. These IDEs enable a smooth development process from start to finish, with built-in version control, .NET CLI support for scaffolding apps, and a robust Razor file editor. Furthermore, the recent advancements in Blazor tooling, such as hot reload capabilities, allow developers to see changes in real time without restarting the entire application, boosting productivity and enhancing the iterative development process. [3]

1.2 Architecture of Blazor

Blazor's architecture is innovative, blending the productivity and power of .NET with the flexibility of modern web browsers through WebAssembly. This comprehensive discussion explores Blazor's intricate architecture, highlighting its two principal hosting models—Blazor Server and Blazor WebAssembly—along with its core features that streamline the development process and enhance application performance.[2]

Unlike conventional client-side frameworks based on JavaScript, Blazor allows developers to create client and server code in C# [1]. Using C# in Blazor offers several distinct advantages over JavaScript, particularly regarding development efficiency, performance, and maintainability. C# is a statically typed language, meaning types are checked at compile-time instead of runtime. This reduces runtime errors significantly, allowing for bugs to be caught during development rather than after deployment. Type safety ensures that many common JavaScript errors, such as type coercion mistakes, are avoided, leading to more robust and predictable code. C# has rich features such as LINQ, generics, and async/await that simplify complex operations, especially concerning data manipulation and asynchronous programming. These features are deeply integrated into the language, offering a cleaner, more expressive syntax. Using C# both on the server and client side simplifies the development process by reducing context switching between languages. This uniformity allows developers to apply the same object-oriented practices and patterns across the entire application stack, improving the development workflow and reducing cognitive load. The .NET ecosystem provides powerful development tools such as Visual Studio, which offers advanced debugging, profiling, and testing tools that enhance developer productivity and code quality.[5]

In Blazor, software systems are constructed using discrete, self-sufficient units known as components. Each component is designed to perform a distinct function and can be utilized, merged, and assembled to forge larger, more intricate systems. Within the Blazor framework, components are fundamental in developing web application user interfaces. A typical component in Blazor is HTML for structure, C# for functionality, and CSS for styling. These components are designed to be reusable and modular user interface elements, such as buttons, forms, navigation bars, or even entire webpage sections. They encompass both the graphical interface and the interactive behavior of these elements. This approach's main benefit is that its components are made to function independently of one another, which reduces dependencies and conflicts. This modularity not only reduces code redundancy but also eases the maintenance process. In Blazor, components undergo a lifecycle that spans several phases, including initialization, rendering, updating, and disposal. Developers can use these lifecycle methods to execute custom actions or tailor behaviors at specific stages. Moreover, components are equipped to manage user interactions via event handlers and adapt to data modifications through data binding mechanisms, effectively responding to user inputs, clicks, or mouse movements. This structure, style, and behavior integration in modular units facilitates efficient approach to building robust web applications.[5]

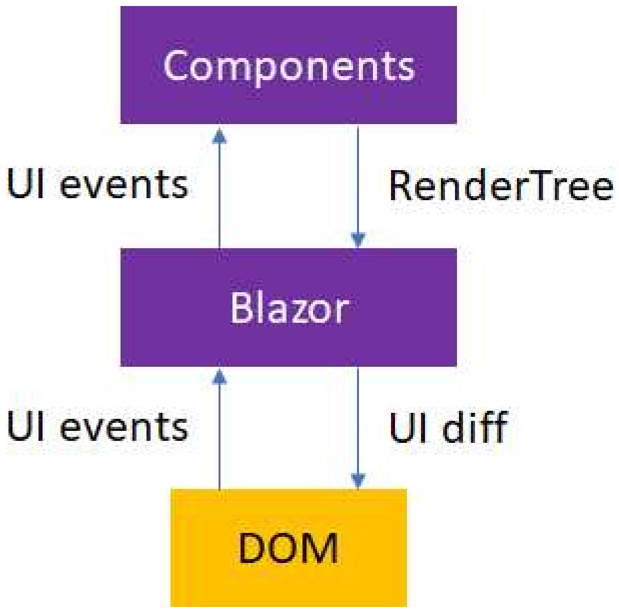


Figure 1 Blazor Component-BaseArchtecture

Components are rendered into a memory-based representation of the browser's Document Object Model (DOM) called a render tree to enable fast and flexible user interface changes. [3]

1.2.1 Blazor WebAssembly

As was already mentioned, Blazor supports two hosting models,allowing choices between server-side and client-side executions based on project requirements. The first hosting model is Blazor WebAssembly. This model utilizes WebAssembly to operate the application completely inside the client's browser. This arrangement enhances the user experience by ensuring that all interactions and updates take place directly within the browser[5].

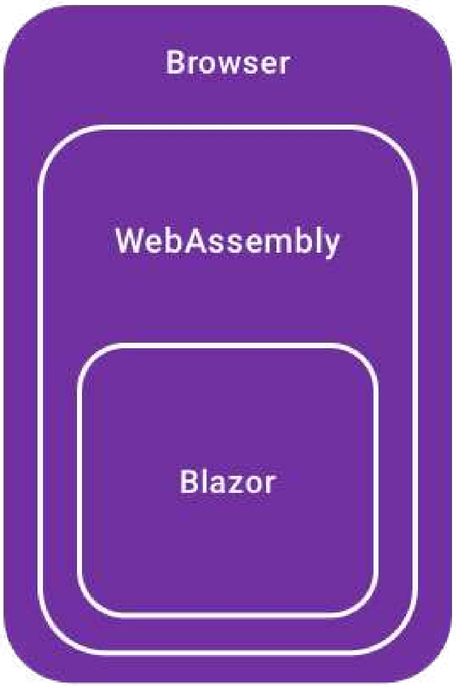


Figure 2 Blazor WebAssembly

Advantages:

Rich Interactivity: Runs entirely within the browser, allowing for smooth and responsive user interactions without server delays.

Reduced Server Burden: Offloads much of the processing to the client, lightening the server's load.

Offline Functionality: Can operate offline using client-side storage mechanisms like local storage or IndexedDB.

Broad Compatibility: Functions across all modern browsers that support WebAssembly, ensuring wide accessibility.[5]

Disadvantages:

Slower Initial Load: The entire application, including the runtime, must be downloaded initially, which can delay the starting interaction time.

Higher Resource Requirement: Demands more processing power and memory from the client device to run smoothly.

Scalability Challenges: Managing numerous users or complex operations client-side can lead to performance bottlenecks.

Security Risks: More application logic is exposed to the client, potentially increasing vulnerability to security threats like code tampering.[5]

1.2.2 Blazor Server

The second hosting model is Blazor Server. Blazor Server facilitates the hosting of Razor components on the server within an ASP.NET Core application. UI interactions are managed via a SignalR connection. Blazor Server executes .NET code on the server and manages interactions with the Document Object Model (DOM) on the client through a SignalR connection. [3]

Blazor Server apps, unlike traditional ASP.NET Core applications, manage content differently. They construct a graph of components resembling an HTML or XML DOM, maintaining state within properties and fields. This component graph is evaluated by Blazor to generate a binary markup representation, which is then transmitted to the client for rendering. The initial connections between client and server involve replacing static pre-rendered elements with interactive components. This server-side pre-rendering significantly enhances the responsiveness of the application by quickly loading HTML content on the client side. [3]

Once components become interactive on the client side, UI updates are initiated through user interactions and application events. Each update triggers a re-render of the component graph, and the system calculates a UI difference (diff). This diff represents the minimal set of DOM edits needed to update the client-side UI and is sent in binary format for the browser to apply. [3]

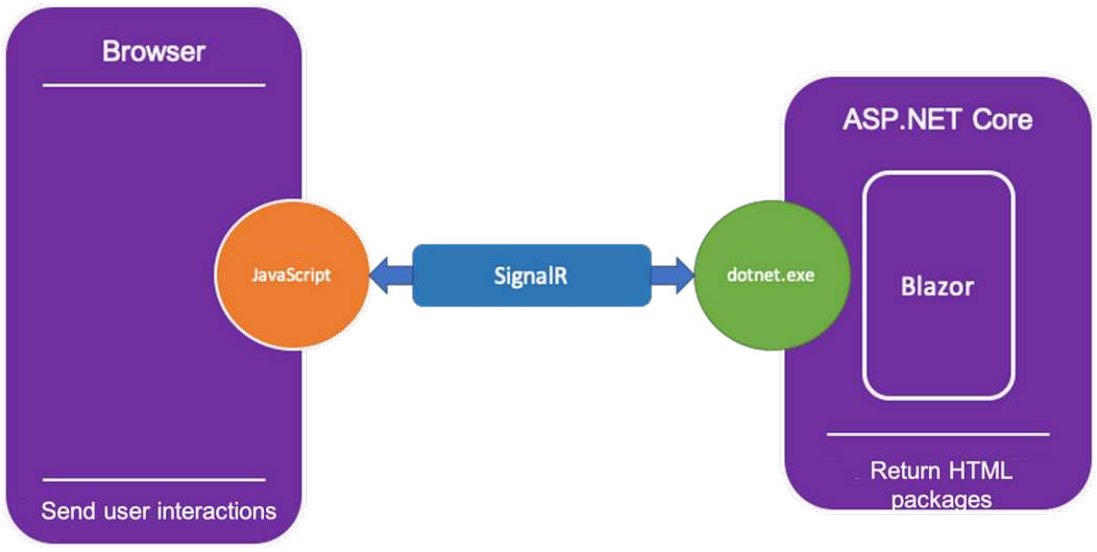


Figure 3 Blazor Server

Blazor Server applications offer several advantages. They enhance the speed of initial page loads through server-side rendering. Only the essential HTML, CSS, and minimal JavaScript required to bootstrap the application are sent to the client, ensuring quicker access to content. With most computational tasks handled server-side, client devices are not heavily burdened, making Blazor Server ideal for applications targeting devices with limited processing power. Blazor Server is tightly integrated with ASP.NET Core, allowing developers to utilize well-established server-side functionalities, middleware, authentication mechanisms, and libraries seamlessly within their applications. Leveraging SignalR, Blazor Server provides real-time communication capabilities between the client and server, enabling dynamic content updates, notifications, and interactive features without requiring additional client-side code, enhancing the user experience. Blazor Server applications can be scaled effectively by increasing server resources, making it easier to manage higher traffic volumes and more complex processing demands without degrading performance.[5]

Disadvantages include the fact that Blazor Server applications rely on a continuous connection to the server, so they offer limited functionality when offline. This dependency can be a drawback in environments where consistent internet access is not guaranteed. As the server handles most of the application's rendering and logic, it can become overloaded, especially in applications with many users or complex operations. This might necessitate robust server solutions and potentially increase infrastructure costs. The dependence on server processing results in latency; every user interaction requires a round trip to the server, which can slow down the application's responsiveness. This delay can impact user satisfaction, particularly in interactions that require instantaneous feedback. While server-side processing centralizes application logic and simplifies deployment, it limits the application's ability to perform client-side complex computations.[5]

1.3 Component Libraries in Blazor

A Blazor Component Library consists of UI components that are simple to integrate and use across different Blazor applications. These components, which range from buttons and forms to navigation menus, encapsulate distinct functionalities or UI elements and are designed for easy customization and extension to meet specific application requirements.[6]

Using these libraries saves time and resources by allowing for the reuse of existing components instead of building new ones from the beginning. This efficiency accelerates development processes and enhances consistency and uniformity throughout the application. Additionally, these libraries support excellent code reusability and modularity, simplifying the maintenance and updating of UI components as applications develop and change.[6]

Let's talk about some of them.

1.3.1 Blazorise

Blazorise, a user-friendly and comprehensive component library for Blazor, is designed to empower developers to build feature-rich and responsive web applications quickly. Supporting a variety of CSS frameworks such as Bootstrap, Bulma, AntDesign, and Material, Blazorise ensures a consistent style and behavior across applications, requiring minimal effort. The library includes components like Data Grids, Modal Dialogs, Tabs, and many other elements crucial for developing interactive web interfaces. Each component is highly customizable, catering to the specific demands of business applications while delivering an exceptional user experience.[7]

One of the primary benefits of using Blazorise is the speed with which developers can implement complex UIs. By offering a set of pre-built components,Blazorise greatly cuts down on development time and effort, allowing developers to concentrate on the distinctive aspects of their applications.Additionally, Blazorise components are built with performance in mind, ensuring that applications look good and run smoothly and efficiently. Blazorise boasts a robust and active community. As an open-source project, it benefits from developers' collective contributions and support worldwide. Another significant advantage of Blazorise is its ability to integrate seamlessly with the .NET ecosystem. This integration ensures developers can use Blazorise alongside other .NET technologies, effectively leveraging existing codebases and libraries. The compatibility with various development environments and platforms means that Blazorise can be used in various applications.[7]

1.3.2 Radzen

Radzen Blazor Components is a robust library offering over 70 free and open-source UI components that are native to the Blazor framework. Designed to enhance Blazor applications, Radzen simplifies development by providing a range of components such as DataGrids, Schedulers, Charts, and Dialogs, all underpinned by themes like Material Design and FluentUI. This diversity in components enables developers to create visually appealing and functionally rich applications efficiently. [8]

One of the key features of Radzen is that it supports both client-side (WebAssembly) and server-side Blazor applications. This dual support ensures that Radizen's components can be utilized in various development contexts, enhancing their versatility and appeal. Moreover, Radzen prioritizes ease of use and integration, allowing components to be added to projects via NuGet, simplifying the installation process and making them readily accessible to developers.[8]

Radzen also stands out for its commitment to regular updates and short development cycles, ensuring that new features and components are made available quickly. This approach helps keep applications up-to-date with the latest functionalities without long waits typically associated with quarterly release cycles.[9]

1.3.3 Telerik UI

Telerik UI for Blazor is a highly acclaimed suite of over 110 native Blazor components designed to expedite the development of modern web applications using the Blazor framework. This suite is equipped with diverse UI controls, including Grids, Charts, Schedulers, and more, which are constructed on native Blazor to avoid reliance on JavaScript interop. This ensures the components are high-performing and seamlessly integrated within the .NET ecosystem, as the Telerik official documentation outlines.[10]

Using such advanced tools can significantly flatten the learning curve for developers diving into Blazor development. The guide highlights the necessity of grasping Blazor's basic principles, which can then be adeptly utilized with advanced tools such as Telerik UI for Blazor to create applications suitable for enterprise use. These components are crafted to be intuitive, enabling developers to easily introduce sophisticated features, irrespective of their level of experience. They support essential data operations like sorting, filtering, and editing right out of the box—capabilities that are crucial for developing robust business applications.

Telerik UI for Blazor also enriches developer productivity through additional design tools and productivity enhancements. For instance, the suite includes a Visual Studio Code extension that simplifies project setups and configurations. Furthermore, it features embedded reporting capabilities that enable developers to generate and manage comprehensive reports directly within their applications, thus enhancing data analysis and decision-making processes.The commercial licensing of Telerik UI for Blazor and a free trial that provides full access to all components and comprehensive technical support make it a prudent choice for businesses considering its adoption. This trial is especially beneficial for thoroughly evaluating the suite's capabilities before committing to a purchase. Telerik also ensures that any issues encountered can be swiftly and effectively addressed by its dedicated support team, enhancing the overall reliability and service experience.[2]

2 RELATED TECHNOLOGIES INTEGRATED WITH BLAZOR

Blazor, a progressive web framework by Microsoft, stands out for its seamless integration with various established technologies, notably .NET Core. This unique integration significantly enhances Blazor's utility in crafting responsive, scalable, and robust web applications, aligning perfectly with the modern web development demands.

2.1 .NET Core and Its Role in Blazor Applications

.NET Core, an open-source, cross-platform framework, plays a pivotal role in both the architecture and operational realms of Blazor applications. Its versatility as a high-performance framework that supports application development across different platforms, including Windows, macOS, and Linux, is particularly beneficial for businesses aiming for broad accessibility in their applications. This ensures that performance and user experience remain consistent regardless of the operating system. The framework gives developers flexibility and choice by supporting various programming languages, including C#, F#, and Visual Basic. Furthermore,.NET Core is highly respected for effectively handling complex applications, and it has sophisticated features including strong encryption support and automated memory management. Because of these features,.NET Core is a desirable choice for businesses that require scalable, dependable apps that can function well in a variety of environments.[12]

.NET Core's middleware architecture is critical in Blazor, especially in Blazor Server applications. Software components known as middleware are brought together to form an application pipeline to manage requests and responses.In Blazor, middleware can be used to customize how requests are handled, enabling scenarios such as authentication, logging, and sophisticated routing, which are crucial for enterprise applications. [14]

.NET Core enhances Blazor's capability to manage the application lifecycle more effectively. It provides tools and services for application startup, dependency management, and event handling. For example, Blazor Server applications benefit from .NET Core's ability to manage connections and user sessions effectively, ensuring that applications remain responsive and scalable under load. This is particularly important in scenarios where applications must maintain a high level of performance despite handling numerous simultaneous user interactions.[13]

2.2 Supporting Technologies

2.2.1 ASP.NET Core

The state-of-the-art, open-source ASP.NET Core platform was designed to make it easier to create web apps that run well on various platforms, including Windows, macOS, and Linux. Because of their adaptability, developers may reach a wider audience and improve the usefulness and reach of their programs in various settings. ASP.NET Core is renowned for its robust backend capabilities, which include sophisticated authentication, authorization, and comprehensive API management tools. These features establish a secure, scalable, and efficient backend, crucial for modern software development environments.[16]

Authentication in ASP.NET Core is a critical feature, ensuring that the identity of users is verified through various supported methods, including forms-based authentication, token-based authentication, and integration with external identity providers like Google, Facebook, and Twitter. The framework facilitates this through middleware components that can be configured in the Startup.cs file, making it adaptable to the specific security needs of an application.

The framework's authorization capability allows developers to create secure environments by granting access to resources based on user roles or policies. This is pivotal in enforcing security measures and ensuring appropriate access to functionalities within the application. ASP.NET Core supports several authorization techniques:

- **Role-Based Authorization**: Restricts access based on user roles.
- **Policy-Based Authorization**: Allows the expression of complex logic to evaluate user permissions.
- **Claims-Based Authorization**: Provides a way to control access based on claims within user identities.

These authorization practices can be declaratively set in code or configuration files and are enforced across applications globally or at granular levels such as per-controller or per-action, enhancing the security and integrity of applications.

With the rise of microservices and service-oriented architectures, effective API management has become crucial. ASP.NET Core excels in building and managing APIs with features supporting RESTful development that can handle multiple data formats including JSON, XML, or plain text. This adaptability is critical for backends intended to serve a diverse set of client applications.

With the support of programs like Swashbuckle, Swagger (now OpenAPI) easily integrates with ASP.NET Core, offering a framework for producing helpful documentation and help pages for APIs. This makes APIs accessible and easy to use by providing thorough documentation, which greatly helps with the development and testing stages.

The latest release, ASP.NET Core 8.0, brings several enhancements that bolster its web and API development capabilities:

- **Minimal APIs**: Expanded support reduces the boilerplate code necessary for setting up APIs, fostering a cleaner and more maintainable codebase.
- **Improved Routing Capabilities**: Enhancements in endpoint routing offer more control and flexibility, simplifying complex routing setups.
- **Enhanced Performance**: Optimizations in middleware processing and HTTP/2 support improve load times and efficiency under high traffic conditions.
- **Streamlined SignalR Client**: Enhancements in SignalR improve real-time communication functionalities, crucial for interactive applications.[16]

2.2.2 Entity Framework

Entity Framework is a key component of the.NET ecosystem; with its sophisticated Object-Relational Mapping features, it provides developers with an incredibly effective means of interacting with databases. EF frees developers from the complexity of database schema manipulation so they may concentrate more on creating solid business logic by automating data management and SQL operation tasks. This discussion explores how Entity Framework revolutionizes data manipulation and persistence, highlighting its essential role in contemporary software development.Entity Framework is a comprehensive ORM solution developed by Microsoft that enables developers to deal with data as easily manageable objects, thus eliminating the need for most of the data-access code that developers typically need to write. As part of the broader .NET platform, EF allows for seamless integration with other technologies, providing developers with a streamlined approach to handling data across varied applications.

EF abstracts the database schema into .NET objects, which developers can work with directly. This abstraction allows for manipulating data without direct SQL interactions, simplifying development and reducing the likelihood of introducing bugs associated with raw SQL coding.

EF fully integrates with Language Integrated Query enabling developers to write database queries using C#. These queries are then translated into SQL at runtime, ensuring applications remain maintainable and secure.EF automatically tracks changes in data objects and calculates the necessary SQL updates during the save operation, simplifying the update process and ensuring data integrity.EF supports database schema migrations, which are essential for evolving applications without losing data or breaking existing functionalities. This feature facilitates version-controlled schema updates seamlessly. First-level caching that is integrated into EF guarantees that repeated requests for the same data inside the same context are answered more quickly, enhancing speed by lowering the number of database round trips. Entity Framework transforms complex SQL tasks into straightforward object manipulations. Data processing is more intuitive when CRUD (Create, Read, Update, Delete) activities are carried out on objects rather to using explicit SQL. EF's ORM capabilities allow for managing complex relationships and associations in the database, using an object-oriented approach that aligns with modern development practices.EF's integration extends to other components of the .NET framework, enhancing its utility in building applications across different platforms, including web (ASP.NET), desktop (WPF, WinForms), and more. The creation of scalable and secure data-driven applications is made easier by this connectivity.

Eventhough EF is very productive and maintainable, speed tuning is still very important. To balance performance, EF enables for the setup of eager and lazy loading. However, developers need to be careful when converting queries into SQL to prevent typical problems like the N+1 queries problem.

With ongoing advancements, Entity Framework Core has been redesigned to be more lightweight, extendable, and cross-platform, supporting a broader range of development scenarios and enhancing performance. This version of EF continues to build on the robust foundation of its predecessor while introducing improvements that meet the demands of modern software architecture.[17]

2.2.3 SignalR

SignalR is a library that is highly valued among ASP.NET developers, as it makes it easier to add real-time web functionality to applications. Real-time web functionality enables server code to instantly push content to connected clients as it becomes available, instead of waiting for a client to request new data. SignalR is immensely beneficial in developing interactive applications requiring real-time user interaction or updates, such as chat systems, live notifications, or real-time dashboards. At its core, SignalR facilitates the addition of real-time communication capabilities to web applications by enabling bi-directional communication between server and client. Servers can now push content to clients instantly, and clients can send messages to servers and receive real-time responses. This dynamic communication loop is pivotal for applications that rely on immediate data flow, such as online gaming interfaces, financial trading platforms, or social media feeds.[15]

SignalR uses WebSockets as its primary transport, but it can automatically fall back to other techniques, such as Server-Sent Events or Long Polling when WebSockets are unavailable. This flexibility ensures that SignalR can provide real-time functionality in various environments and browsers, including those that do not support the latest protocols. The process begins with the client initiating a connection request to the server. SignalR negotiates the best possible transport method based on the client and server capabilities. Once the connection is established, the server holds it open, allowing for a persistent, two-way data exchange.[15]

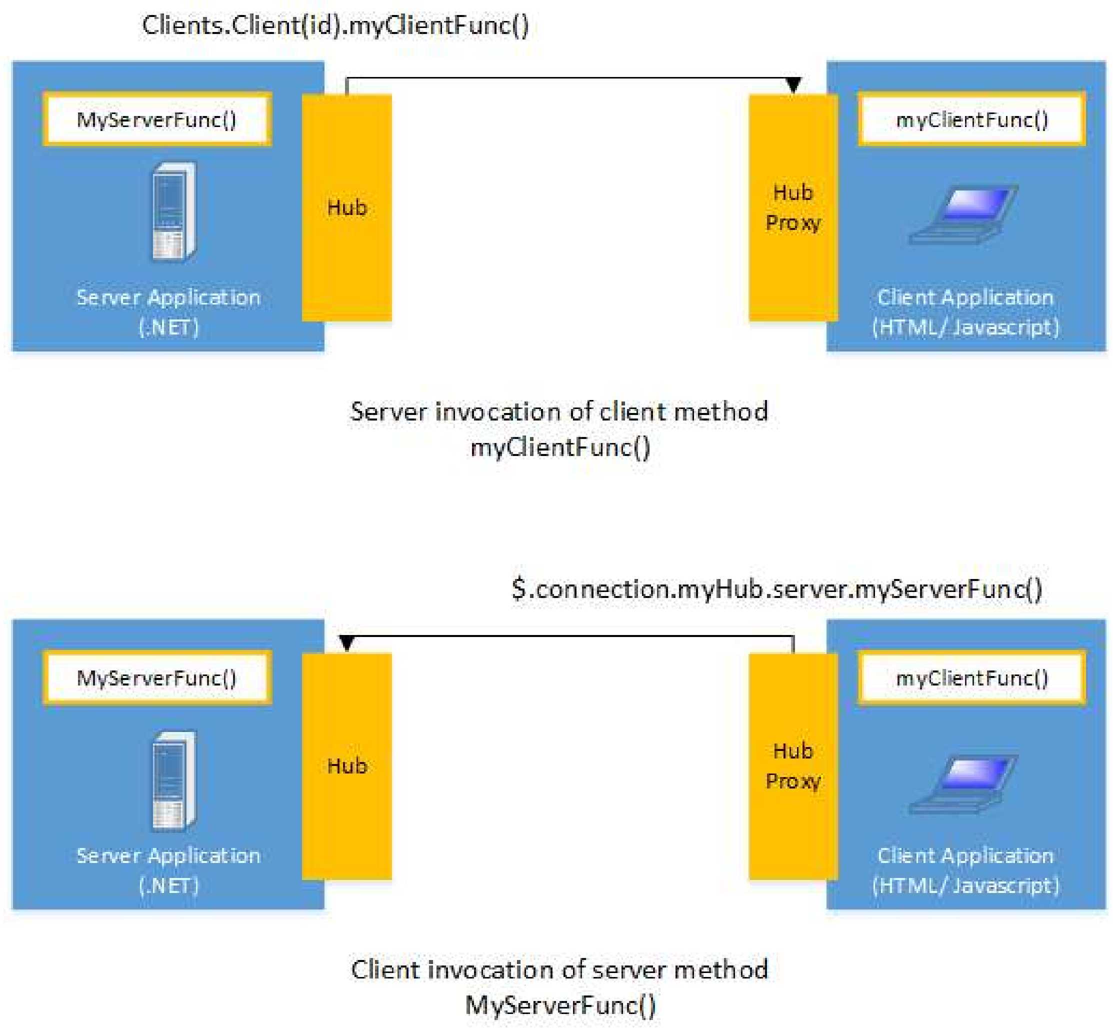

Clients.Client(id).myClientFunc()
MyServerFunc()
Hub
Server Application (.NET)
Hub Proxy
myClientFunc()
Client Application (HTML/Javascript)
Server invocation of client method myClientFunc()
$.connection.myHub.server.myServerFunc()
MyServerFunc()
Hub
Server Application (.NET)
Hub Proxy
myClientFunc()
Client Application (HTML/Javascript)
Client invocation of server method MyServerFunc()

Figure 4 How SignalR works

In the event of brief disconnections, SignalR automatically reconnects to maintain a smooth and uninterrupted user experience. It manages connection lifecycles meticulously, handling the complexities of setup, maintenance, and teardown. SignalR supports scaling out of applications across multiple servers to handle more traffic. It integrates seamlessly with backplane technology, which helps manage connections across different servers and synchronize messages across them. Also it allows the server to group connections, which can be useful for broadcasting messages to specific subsets of clients, such as a chat room or a specific user group.

The adoption of SignalR in web applications comes with several advantages: Applications become more responsive and interactive as the delay between the server and client is minimized. Unlike traditional request-response models where clients poll the server regularly, SignalR maintains an open connection, drastically reducing the number of requests to the server.SignalR abstracts the complexity of managing real-time communications, allowing developers to focus more on core application features rather than connection management details.[15]

2.3 GraghQL Integration

2.3.1 Introduction to GraphQL

In the context of web development, particularly in scenarios requiring efficient data retrieval and management, GraphQL presents a significant advancement over traditional REST APIs. This section of the thesis explores why and how GraphQL, as a query language for APIs, offers a more optimized and flexible approach to data handling.

GraphQL, which was developed by Facebook in 2012 and made available to the public in 2015, offers a more effective, potent, and adaptable substitute for the conventional REST API. Its capacity to let clients ask for just what they require, and nothing more, significantly increases application interface performance and efficiency and reduces network traffic, particularly in complex systems. [18]

Unlike REST, which often requires multiple endpoints to retrieve different pieces of data, GraphQL allows a single query to aggregate all the necessary data. This approach significantly reduces the number of network requests, enhancing performance and reducing latency. One of the most compelling features of GraphQL is its ability to eliminate both overfetching and underfetching.It minimizes the inefficiencies associated with receiving extra information (overfetching) or requiring additional requests to fulfill data requirements (underfetching) by letting the client indicate exactly what data is needed. GraphQL APIs are defined by their schemas, which are strongly typed. This setup allows the API to validate queries against the schema effectively, providing a clear contract that specifies exactly what data can be accessed and how. This feature enhances security and stability by ensuring that interactions with the API are predictable and conform to specified data structures. Unlike traditional REST APIs, GraphQL supports real-time data updates through subscriptions. This feature enables clients to maintain a consistent state with server data, updating in real time as changes occur. This capability is particularly beneficial for applications requiring high levels of interactivity and up-to-date data displays, such as dynamic dashboards and live data feeds.[18]

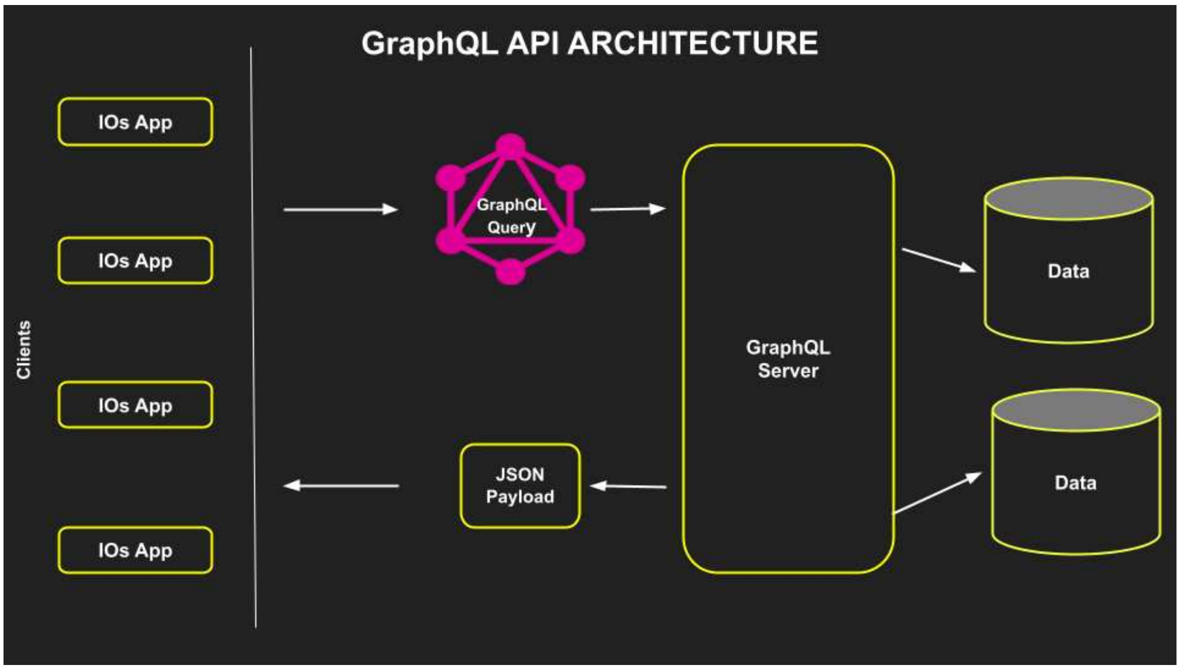


Figure 5 GraphQL Architecture

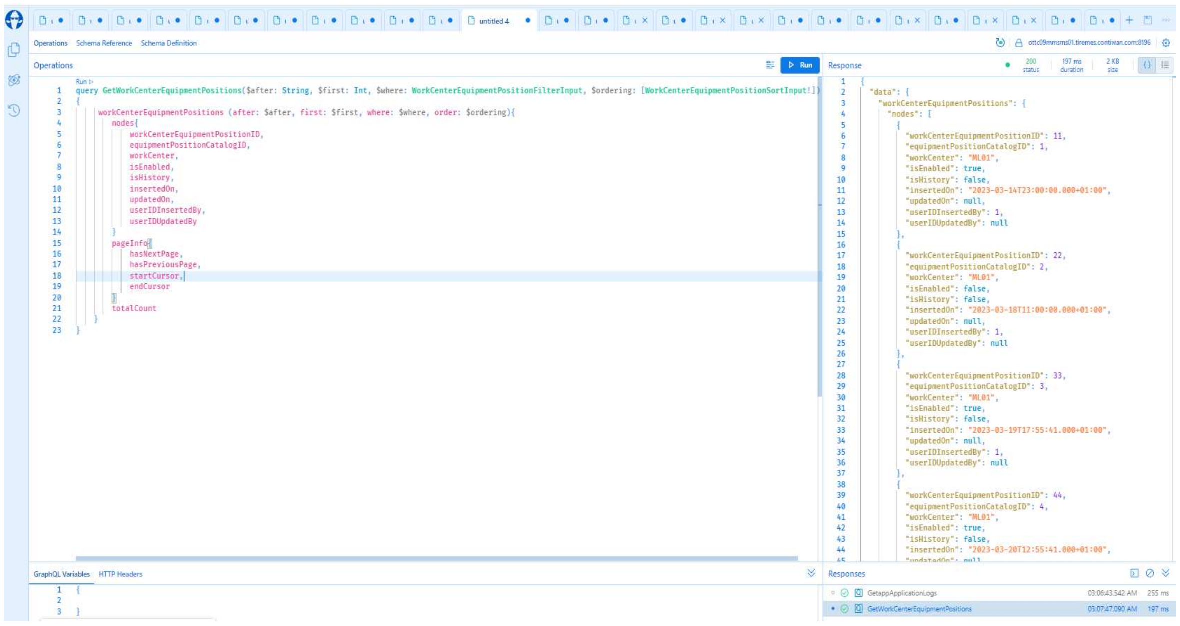
Interesting to understand how GraphQL works exactly. Each client (iOS App) constructs and sends a GraphQL query requesting specific data. This query clearly defines the fields and objects needed by the client. Upon receiving the query, the GraphQL Server first validates it against the schema to ensure it only requests defined fields. It then processes this query, orchestrating calls to various resolvers. Resolvers are functions connected to specific types or fields in the GraphQL schema. Each resolver knows how to fetch or compute the value for its field, often requiring calls to external data sources (databases, web services, etc.). After all resolvers have returned their data, the GraphQL Server assembles these pieces into a single JSON object structured according to the query. This object is then sent back to the client as the response. The client receives the JSON payload. This payload is specifically designed to match the query, guaranteeing that the client receives exactly what they requested and that no data is fetched in excess or insufficiently.

2.3.2 Integration with Blazor

Integrating GraphQL within modern web applications, especially those built with platforms like Blazor, can greatly improve the data interaction layer's performance and flexibility. Hot Chocolate is an open-source GraphQL server for .NET that facilitates this integration by providing a robust framework to build efficient GraphQL APIs. This framework simplifies the setup and execution of a GraphQL server in a .NET environment, supporting a wide array of features including subscriptions, middleware, and real-time data handling.

As a tool, Hot Chocolate is designed to enhance the GraphQL implementation process, enabling developers to create scalable and performant GraphQL servers that can handle complex queries with ease. Hot Chocolate includes Banana Cake Pop, a GraphQL IDE that significantly aids developers in testing and debugging their GraphQL queries. This integrated development environment is crucial for optimizing query structures and mutations, ensuring that they are both efficient and effective.[19]

Adopting GraphQL represents a strategic enhancement in data handling within web applications. By providing precise data retrieval capabilities, reducing unnecessary data transfers, and ensuring real-time updates, GraphQL can significantly improve the performance, user experience, and scalability of applications. This is demonstration of using Graphql Query for fetching the data from Api in BananaCake IDE:



```
query GetWorkCenterEquipmentPositions($after: String, $first: Int, $where: WorkCenterEquipmentPositionFilterInput, $ordering: [WorkCenterEquipmentPositionSortInput!])
{
    workCenterEquipmentPositions (after: $after, first: $first, where: $where, order: $ordering){
      nodes{
        workCenterEquipmentPositionID,
        equipmentPositionCatalogID,
        workCenter,
        isEnabled,
        isHistory,
        insertedOn,
        updatedOn,
        userIDInsertedBy,
        userIDUpdatedBy
      }
      pageInfo{
        hasNextPage,
        hasPreviousPage,
        startCursor,
        endCursor
      }
      totalCount
    }
}
```

```
{
  "data": {
    "workCenterEquipmentPositions": {
      "nodes": [
        {
          "workCenterEquipmentPositionID": 11,
          "equipmentPositionCatalogID": 1,
          "workCenter": "ML01",
          "isEnabled": true,
          "isHistory": false,
          "insertedOn": "2023-03-14T23:00:00.000+01:00",
          "updatedOn": null,
          "userIDInsertedBy": 1,
          "userIDUpdatedBy": null
        },
        {
          "workCenterEquipmentPositionID": 22,
          "equipmentPositionCatalogID": 2,
          "workCenter": "ML01",
          "isEnabled": false,
          "isHistory": false,
          "insertedOn": "2023-03-18T11:00:00.000+01:00",
          "updatedOn": null,
          "userIDInsertedBy": 1,
          "userIDUpdatedBy": null
        },
        {
          "workCenterEquipmentPositionID": 33,
          "equipmentPositionCatalogID": 3,
          "workCenter": "ML01",
          "isEnabled": true,
          "isHistory": false,
          "insertedOn": "2023-03-19T17:55:41.000+01:00",
          "updatedOn": null,
          "userIDInsertedBy": 1,
          "userIDUpdatedBy": null
        },
        {
          "workCenterEquipmentPositionID": 44,
          "equipmentPositionCatalogID": 4,
          "workCenter": "ML01",
          "isEnabled": true,
          "isHistory": false,
          "insertedOn": "2023-03-20T12:55:41.000+01:00",
          "updatedOn": null
```

```
ottc09mmsms01.tiremes.contiwan.com:8196
GetappApplicationLogs               03:06:43.542 AM   255 ms
GetWorkCenterEquipmentPositions     03:07:47.090 AM   197 ms
```

Figure 6 GraphQL Query Example

The query:

```
query GetWorkCenterEquipmentPositions($after: String, $first: Int,
$where: WorkCenterEquipmentPositionFilterInput, $ordering: [WorkCente-
rEquipmentPositionSortInput!])
{
     workCenterEquipmentPositions (after: $after, first: $first, where:
$where, order: $ordering){
        nodes{
            workCenterEquipmentPositionID,
            equipmentPositionCatalogID,
            workCenter,
            isEnabled,
            isHistory,
            insertedOn,
            updatedOn,
            userIDInsertedBy,
            userIDUpdatedBy
        }
        pageInfo{
            hasNextPage,
            hasPreviousPage,
            startCursor,
            endCursor
        }
        totalCount
    }
}
```

2.4 Testing Blazor With BUNIT

The most important aspect of software development is testing. It ensures that code not only performs its intended function under normal conditions but also gracefully handles unexpected or incorrect inputs. For Blazor applications, built on a component-based architecture similar to React and Vue.js, effective testing tools that understand the component lifecycle and state management are paramount.

BUnit is a testing library specifically designed for Blazor components. It provides a comprehensive approach to unit and integration testing, allowing developers to thoroughly test their components in isolation or within the context of their broader application. It enables developers to write tests that simulate the behavior of components during rendering and interaction phases. BUnit tests are typically executed in a test environment that mimics the Blazor runtime, allowing components to be rendered and interacted with as if they were running in a real browser.[20]

BUnit's strength lies in its capability to create isolated testing environments for components, devoid of any dependencies on other parts of the application. This feature is a game-changer when it comes to pinpointing and resolving defects within individual components before integration, a critical step in the development process.

BUnit seamlessly integrates with widely-used .NET dependency injection (DI) containers and mocking libraries, making it a breeze for developers to replace actual service implementations with stubs or mocks. This integration simplifies the testing process, allowing developers to focus on the component's behavior rather than its integration with external systems.BUnit also offers mechanisms to simulate user interactions, such as clicks, form submissions, and input changes, a crucial feature for testing component responses to user actions. Additionally, BUnit provides tools for mocking and asserting outcomes of JavaScript interop calls, a necessity for components that rely on JavaScript for functionality not directly available in Blazor. Lastly, BUnit supports snapshot testing, a feature that captures the rendered markup of components and compares it against known good snapshots, ensuring that component changes do not unintentionally affect their rendered output.[20]

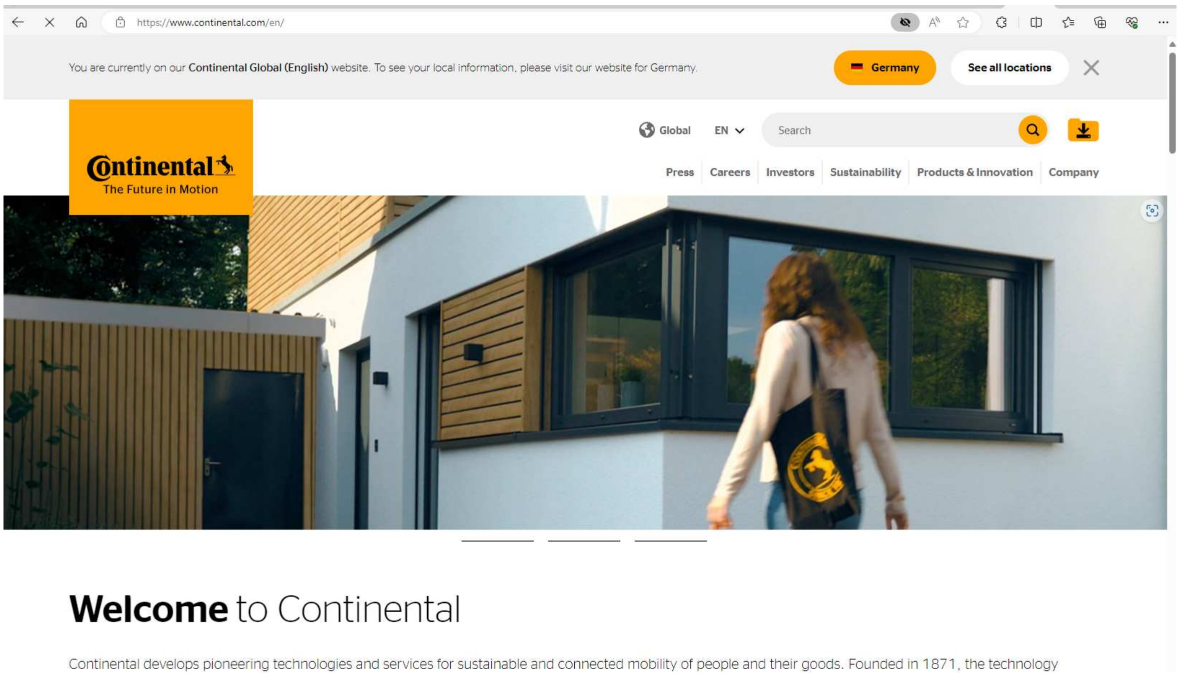
The BUnit test example:

```
@inherits TestContext
@code {
   [Fact]
   public void HelloWorldComponentRendersCorrectly() {
      // Act
      var cut = Render(@ < HelloWorld / > );
      // Assert
      cut.MarkupMatches(@ < h1 > Hello world from Blazor < /h1>);
      }
   }
```

II. ANALYSIS

4. TECHNOLOGICAL FOUNDATIONS AND APPLICATIONS AT CONTINENTAL BARUM S.R.O

In Figure 7 we see main official site page of the company [21]



You are currently on our Continental Global (English) website. To see your local information, please visit our website for Germany.

Welcome to Continental

Continental develops pioneering technologies and services for sustainable and connected mobility of people and their goods. Founded in 1871, the technology

Figure 7 Continental Official Website

As a part of the wider Continental AG company, one of the top global suppliers to the automobile industry, Continental Barum is a well-known brand in the tire manufacturing sector. Continental began business in 1871 and has since grown to become a leading international technology firm. The business initially focused on making soft rubber parts, such horse hoof buffers. Continental highlighted innovation and diversity as essential components of its strategy. The company quickly outgrew its basic hoof guards and
other hard rubber parts to include over 60,000 products, including rubberized fabrics used in airships, hot air balloons, and airplanes. Based in Otrokovice, Czech Republic, Continental Barum benefits from a strong combination of Czech tire manufacturing traditions and German engineering excellence.[22]

Continental Barum focuses on the production of a wide range of tires, including those for passenger cars, trucks, and special vehicles. The company is known for producing high-quality, durable tires that incorporate advanced German technology. The company is committed to innovation in tire technology, emphasizing safety, fuel efficiency, and environmental sustainability in their product designs. This aligns with Continental AG's broader strategy of developing cutting-edge automotive technologies.

Continental AG, as a global leader in automotive manufacturing and technology, strongly emphasizes developing its own digital solutions, including websites and applications. This internal development approach ensures that their digital infrastructure is closely aligned with their specific business needs and industry standards while keeping critical data and proprietary technologies private and secure. By developing and managing their sites and applications internally, Continental protects its business data and intellectual property. This is crucial in the automotive industry, where data security and the protection of technological innovations are paramount.

The deep integration of Continental's internal development team with their current technologies is a benefit. This includes everything from manufacturing systems and supply chain logistics to vehicle telematics and infotainment systems.

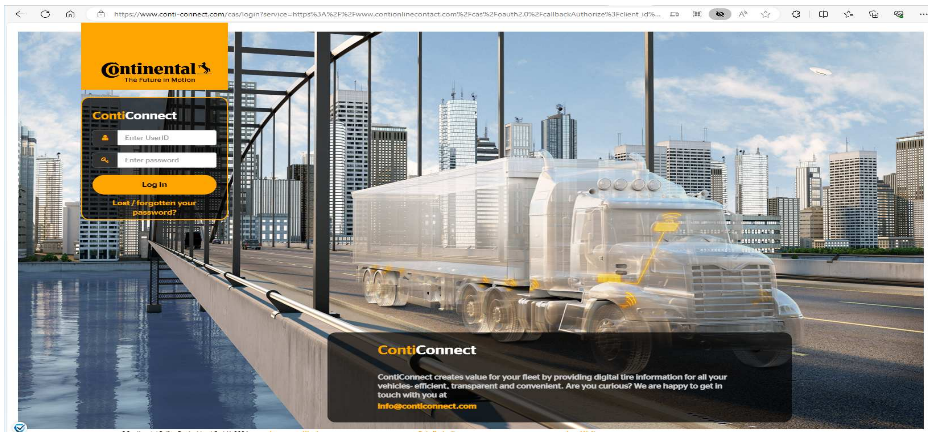
Examples of Development Projects:

1.Tire Information Systems: Continental's tire information systems mark a major improvement in vehicle efficiency and safety. These systems are a component of Continental's larger effort to incorporate cutting-edge sensor technologies into car tires, giving drivers access to real-time tire condition data. This technology is critical to the development and implementation of autonomous vehicle technologies, as well as to improving consumer safety by warning drivers of possible tire problems before they become problematic. The tires' internal sensors keep an eye on a number of variables, such as tread depth, temperature, and pressure. The onboard systems of the car receive this data in real time, and it might also be sent straight to the driver's dashboard or mobile app. The system utilizes advanced algorithms to analyze collected data and predict tire wear and performance issues, allowing for preemptive maintenance actions that can prevent accidents and extend tire life.Beyond providing alerts, the tire information can be integrated with vehicle safety systems, such as those controlling braking and stability, to automatically adjust in response to changing tire conditions.The system enhances safety by preventing tire-related accidents.It supports eco-friendly driving practices by ensuring tires are at optimal inflation, which improves fuel efficiency.It is integral in developing autonomous driving systems where vehicle awareness is key to operational safety.[23]

2.ContiConnect: ContiConnect is Continental's innovative digital tire monitoring platform designed specifically for commercial fleets. This web-based solution harnesses the power of IoT technology to streamline fleet maintenance, enhance safety, and reduce operational costs by providing fleet managers with critical tire performance data.

With ContiConnect, fleet managers can keep a real-time tab on the tire conditions of multiple vehicles from a centralized dashboard accessible via web technologies. This includes vital data on tire pressure and temperature, ensuring tire health and vehicle safety. ContiConnect promptly alerts and notifies when tire conditions deviate from preset thresholds, enabling immediate attention to potential issues before they escalate into costly downtime or accidents. The platform also accumulates historical data for analysis, aiding fleet managers in understanding tire wear patterns and optimizing replacement cycles and maintenance practices accordingly. It seamlessly integrates with existing fleet management systems, providing a comprehensive overview of fleet operations and enhancing decision-making processes.

Fleets can save a lot of money on total operating costs by keeping tires in good condition and reducing fuel usage and tire life. Instant alerts regarding tire problems lower potential expenses by preventing accidents brought on by tire failures. Fleet uptime and resource allocation are improved by streamlined maintenance procedures and scheduled servicing based on precise tire condition data, reducing downtime and increasing productivity. Our tire information systems and ContiConnect have actual, practical cost-saving advantages that can significantly improve your bottom line. These advantages are not merely theoretical.[24]   In Figure 8 we see ContiConnect Website



https://www.conti-connect.com/cas/login?service=https%3A%2F%2Fwww.contionlinecontact.com%2Fcas%2Foauth2.0%2FcallbackAuthorize%3Fclient_id%...

Continental The Future in Motion

ContiConnect

Enter UserID

Enter password

Log In

Lost / forgotten your password?

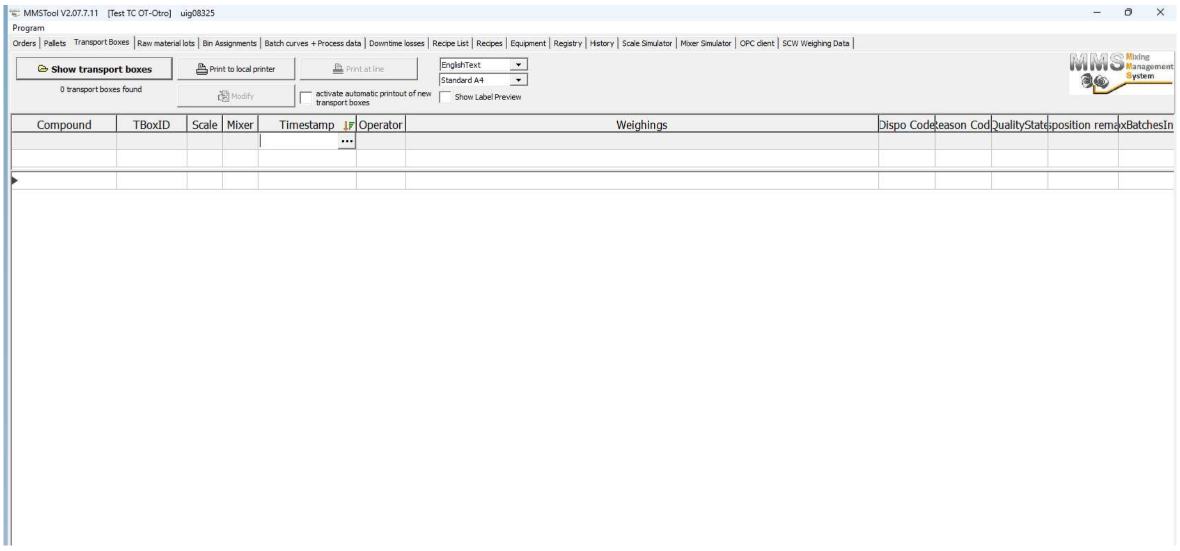
ContiConnect

ContiConnect creates value for your fleet by providing digital tire information for all your vehicles- efficient, transparent and convenient. Are you curious? We are happy to get in touch with you at
info@conticonnect.com

Figure 8 ContiConnect Website

5 OVERVIEW OF THE OLD VERSION OF APPLICATION

This is MMS-Tool application.


MMSTool V2.07.7.11 [Test TC OT-Otro] uig08325
Program
Orders | Pallets | Transport Boxes | Raw material lots | Bin Assignments | Batch curves + Process data | Downtime losses | Recipe List | Recipes | Equipment | Registry | History | Scale Simulator | Mixer Simulator | OPC client | SCW Weighing Data
Show transport boxes
0 transport boxes found
Print to local printer
Print at line
Modify
activate automatic printout of new transport boxes
EnglishText
Standard A4
Show Label Preview
MMS Mixing Management System

Compound	TBoxID	Scale	Mixer	Timestamp	Operator	Weighings	Dispo Code	ɫeason Cod	QualityState	sposition rema	xBatchesIn

Figure 9 MMS-Tool Application

The Mixing Management System (MMS) is a well-known illustration of the difficulties encountered while utilizing outdated software technologies. This Delphi application was created for the mixing department, a crucial aspect of the tire production process that requires accurate ingredient mixing and formula management. The presented screenshot shows that the MMS interface is a classic example of an outdated application layout that does not adhere to modern design standards. The interface is busy and can be challenging for users to use, despite the design being functional while being obsolete. This intricacy affects more than just the appearance; it also affects operation, making it challenging for inexperienced operators to grasp the possibilities of the system without intensive training.

The outdated MMS version's troublesome management of user roles and authentication was one of its major flaws. Due to the lack of adequate access controls caused by this problem, it is challenging to implement appropriate security measures and role-based access to sensitive information and operations. Inadequate user authentication and role management not only created security vulnerabilities but also impeded operational effectiveness by making it difficult for people to obtain permissions that corresponded with their duties.

Moreover, there was no pagination in the system. When this is missing, the system loads all data entries at once, which can cause significant delays, particularly when working with large datasets. These problems with performance worsen at times of peak operation, which has a major effect on output.

MMS editing features are difficult to use and restricted. Not every field has editable options, and the ones that do sometimes don't provide user-friendly interactions. Additionally, the system has inconsistent field validation, which means that some data submissions are not thoroughly verified to ensure accuracy and could result in mixing problems.

There are also few localization features, mostly confined to the 'Transport Boxes' area. This restriction can provide a serious challenge for a multinational corporation like as Continental, whose activities are spread across several nations with disparate linguistic needs.

Sorting and filtering operations within MMS are notably slow, which can be frustrating for users who need to access specific data quickly. This sluggish performance further detracts from the system's efficiency, highlighting the need for more robust data handling mechanisms.

Another serious problem with the system is how long it takes for large datasets to load. Any delay in a fast-paced manufacturing environment might result in lower operational efficiency and higher costs. For a business like Continental, where productivity and efficiency are critical, this is especially troublesome.

6 DECISION TO MIGRATE TO BLAZOR

The company's strategic decision to transfer old software from Delphi to a Blazor Web application is covered in great detail in this thesis, which looks at modern software development techniques and system migration tactics. A number of variables, such as particular organizational demands and industry trends, have influenced this change.

The decline in the number of qualified engineers using this old-fashioned programming language was the main factor in the decision to abandon Delphi. The popularity of more modern and flexible languages caused Delphi to lose its significance. The absence of Delphi knowledge made it difficult to improve and maintain our systems, endangering their long-term viability. Furthermore, accessibility was limited by Delphi's original desktop design, which was unable to satisfy the expanding need for mobile connectivity.

Blazor presented itself as an optimal replacement due to its modern features and alignment with our strategic objectives. As a free and open-source web framework that allows developers to build interactive web interfaces using C#, Blazor enables us to utilize the robust .NET ecosystem and the C# skills prevalent within our team. This compatibility significantly eases the learning curve and streamlines the transition and development process.

Blazor's component-based architecture, similar to frameworks like React or Angular, but uniquely operating within the .NET environment, is a key advantage. This integration allows us to use a single programming language, C#, across both client and server-side applications, reducing our reliance on JavaScript and simplifying the development landscape by minimizing the need to use multiple languages.

We have further enhanced our development environment by incorporating the Blazorise library, a set of UI components that work seamlessly with Blazor. Blazorise enriches Blazor's already robust capabilities by providing a comprehensive suite of customizable and extendable components. This library facilitates rapid UI development while ensuring consistency and responsiveness across all platforms, which significantly accelerates our application's time-to-market and enhances user experience.

Blazor's scalability and maintainability also play crucial roles in its selection. Blazor applications can operate on servers or client-side in the browser using WebAssembly, providing diverse deployment options and performance tuning possibilities. This flexibility is vital for developing an application capable of managing growing user numbers and data processing requirements efficiently. Ultimately, choosing Blazor aligns with company's dedication to innovation and competitive advantage, leveraging cutting-edge technology to boost capabilities, refine user experiences, and drive business growth.

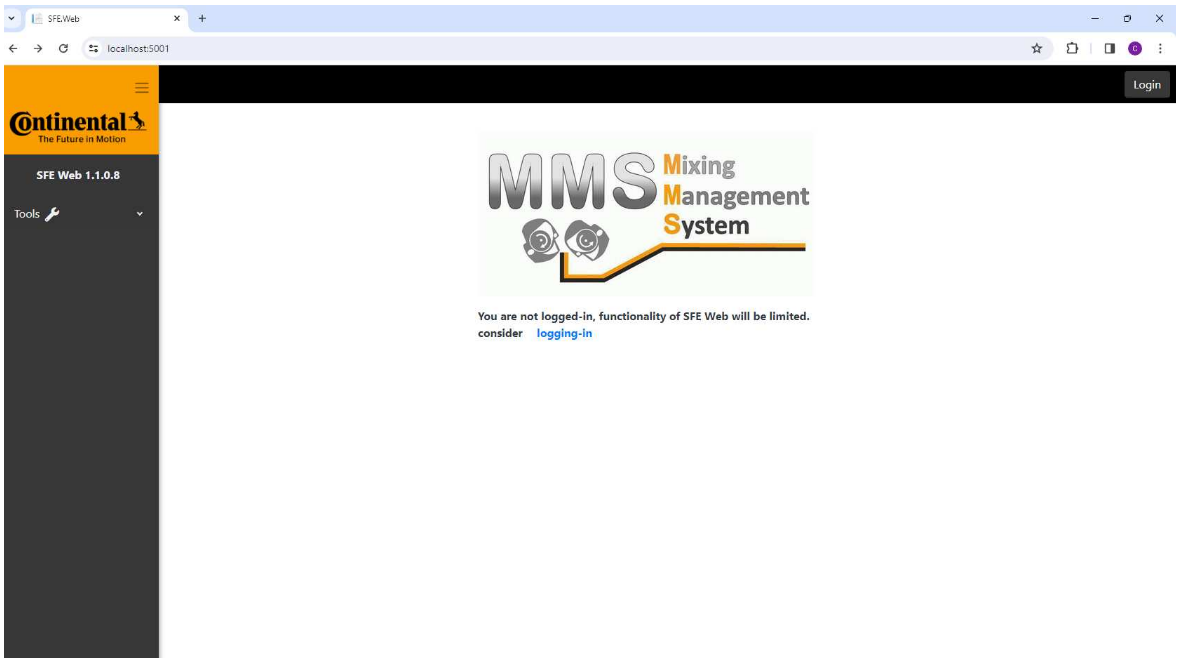
In conclusion, the strategic choice to move from a Delphi to a Blazor Web application was motivated by the need for modernization, better integration with existing and future technical infrastructures, higher developer availability, and enhanced scalability. In addition to addressing the shortcomings of an outdated technology stack, this move better positions the business to handle present and future operational demands.

7 SFE WEB APP

This chapter describes current state of the web application made by the author.

7.1 Web Application design

In Figure 9 we see the final design of the home page. If user is not logged in the functionality of the application will be limited. No pages will be seen.



You are not logged-in, functionality of SFE Web will be limited.
consider logging-in

Figure 10 Home Page

In Figure 10 we see the login page.

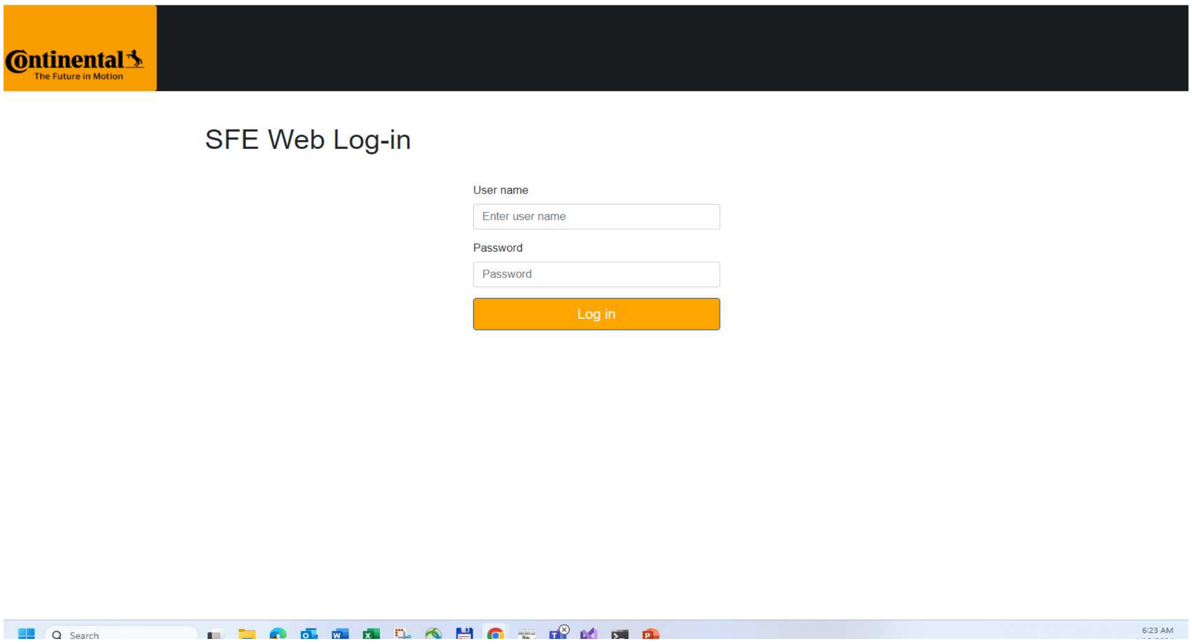


Figure 11 Login Page

- Application after LogIn

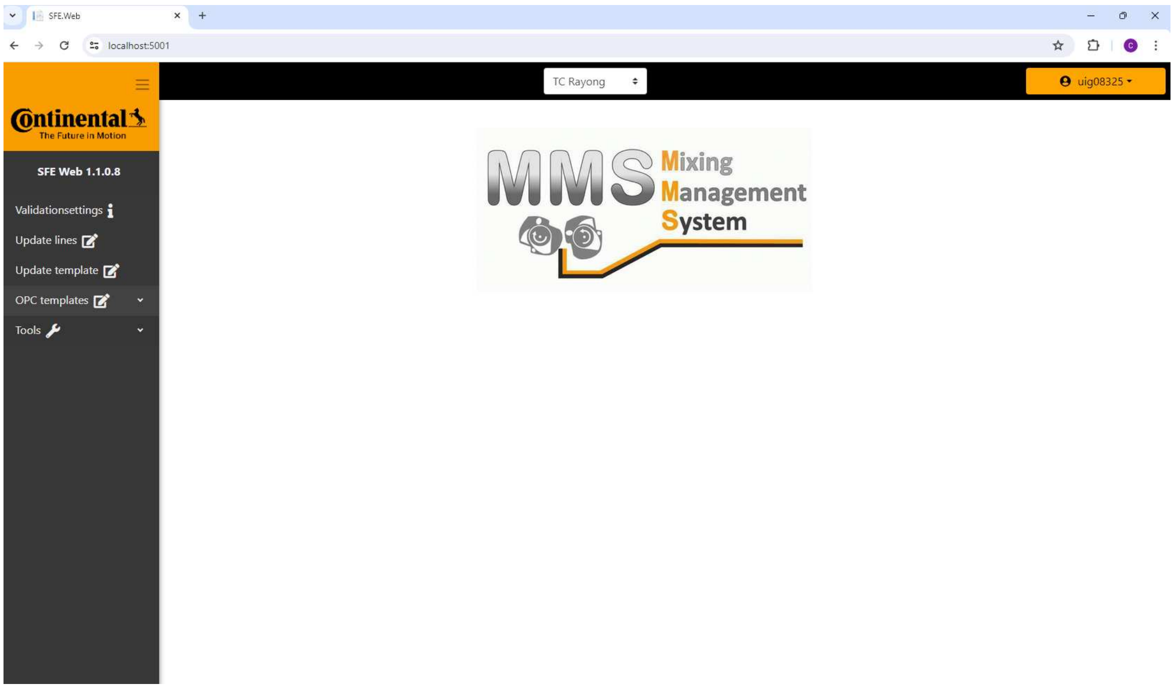


Figure 12 Sfe-Web Application Afetr LogIn

- SideBar

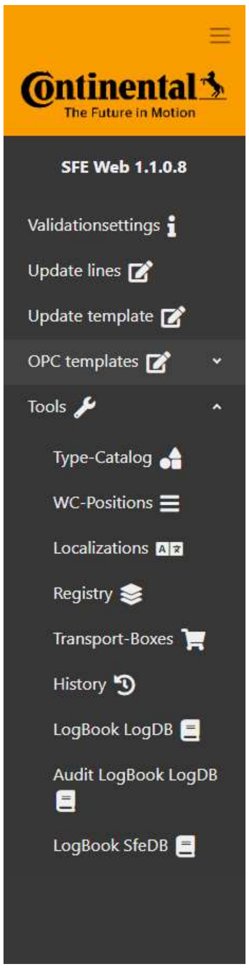


Figure 13 Sfe-Web SideBar

- User DropdownMenu

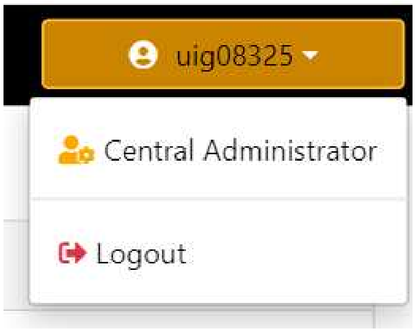
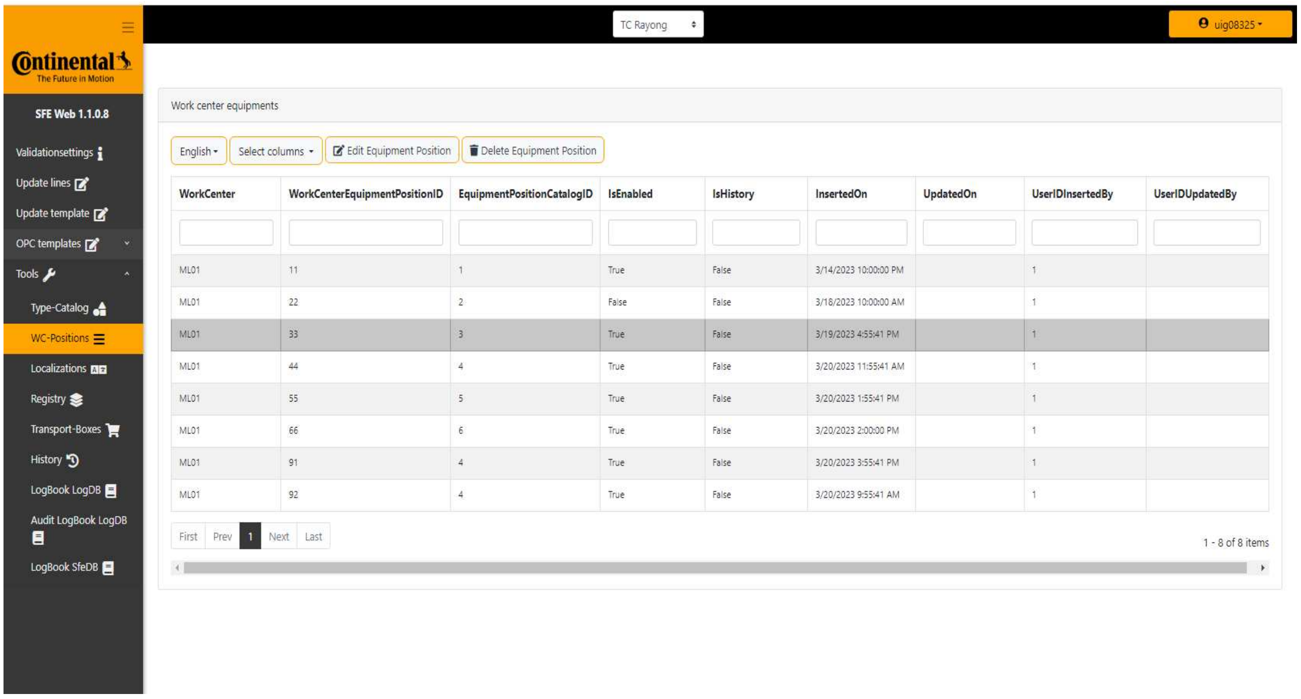


Figure 14 User DropDown Menu

- DataGrid Page



Work center equipments

WorkCenter	WorkCenterEquipmentPositionID	EquipmentPositionCatalogID	IsEnabled	IsHistory	InsertedOn	UpdatedOn	UserIDInsertedBy	UserIDUpdatedBy
ML01	11	1	True	False	3/14/2023 10:00:00 PM		1	
ML01	22	2	False	False	3/18/2023 10:00:00 AM		1	
ML01	33	3	True	False	3/19/2023 4:55:41 PM		1	
ML01	44	4	True	False	3/20/2023 11:55:41 AM		1	
ML01	55	5	True	False	3/20/2023 1:55:41 PM		1	
ML01	66	6	True	False	3/20/2023 2:00:00 PM		1	
ML01	91	4	True	False	3/20/2023 3:55:41 PM		1	
ML01	92	4	True	False	3/20/2023 9:55:41 AM		1	

1 - 8 of 8 items

Figure 15 WorkCenter Equipmentts DataGrid Page

- Plants Dropdown

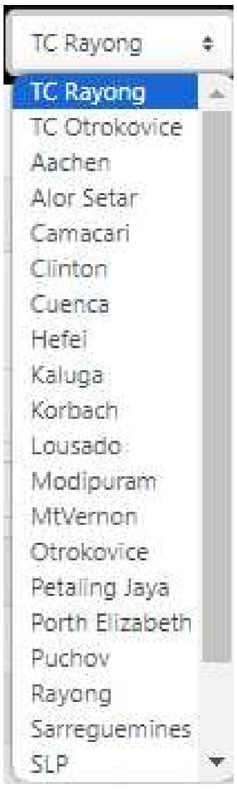
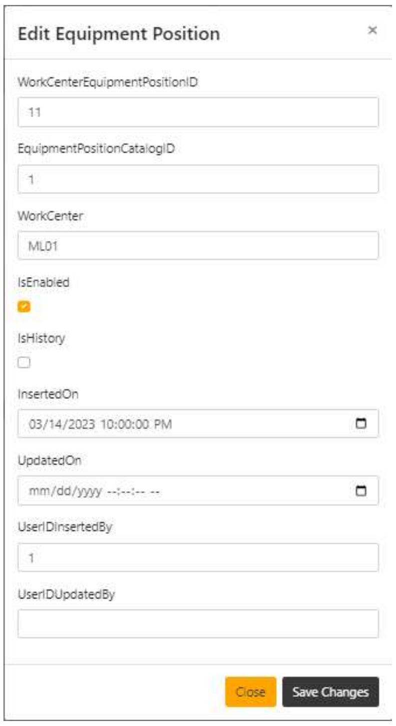


Figure 16 Plant Dropdown

- EditModal


Edit Equipment Position

WorkCenterEquipmentPositionID: 11

EquipmentPositionCatalogID: 1

WorkCenter: ML01

IsEnabled: ✓

IsHistory: ☐

InsertedOn: 03/14/2023 10:00:00 PM

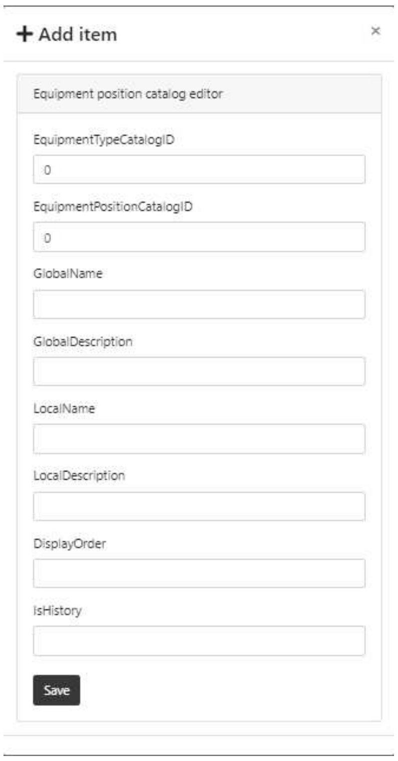
UpdatedOn: mm/dd/yyyy --:--:-- --

UserIDInsertedBy: 1

UserIDUpdatedBy:

Figure 17 Edit Equipment Positions Modal

- AddModal


Add item

Equipment position catalog editor

EquipmentTypeCatalogID: 0

EquipmentPositionCatalogID: 0

GlobalName:

GlobalDescription:

LocalName:

LocalDescription:

DisplayOrder:

IsHistory:

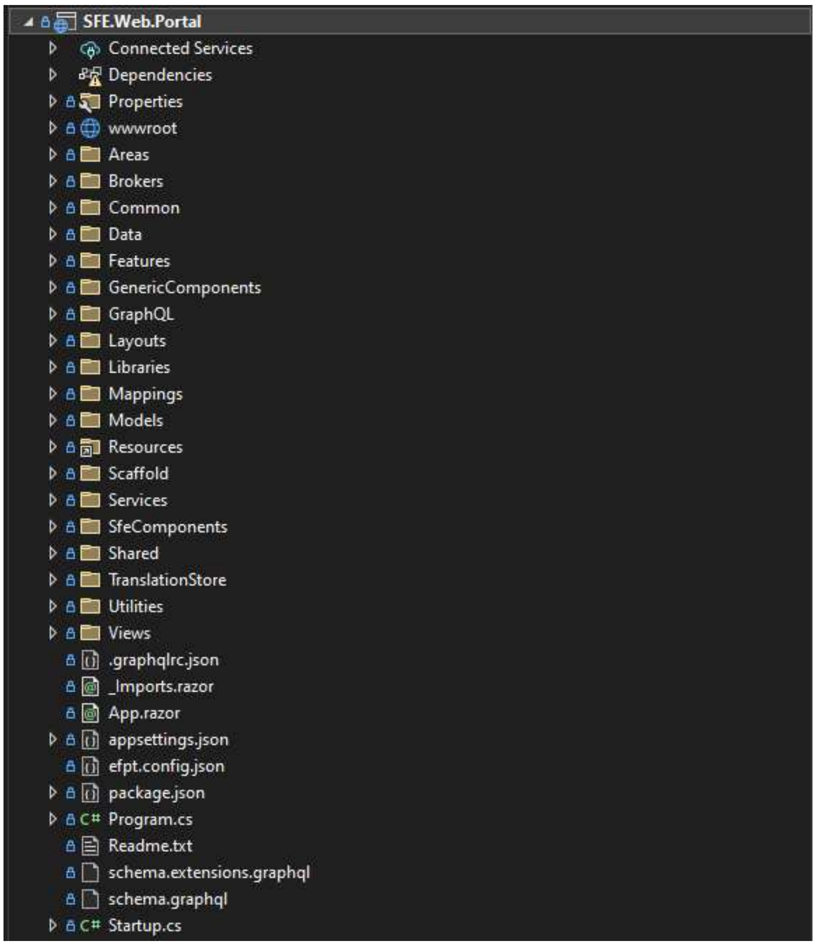
Figure 18 Add Equipment Positions Modal

7.2 Web Application structure

Project has two repositories. Frontend components are contained in the SFE.Web repository. This repository is devoted to the application's web interface, where client-side logic, interactive features, and layout are developed and maintained.
On the server side, the SFE.Web.Api repository contains backend logic. It serves as the backbone for our application, handling data processing, API development, and business logic implementation.

These repositories are essential to the project's modular development because they make it easy to distinguish between enhancements intended for clients and those intended for servers. Modern software development techniques are demonstrated by the usage of several distinct but linked repositories, which encourage productivity, maintainability, and scalability throughout the project's lifecycle.

In this Figure 18 we see the structure of the SFE-Web project.



```
SFE.Web.Portal
    Connected Services
    Dependencies
    Properties
    wwwroot
    Areas
    Brokers
    Common
    Data
    Features
    GenericComponents
    GraphQL
    Layouts
    Libraries
    Mappings
    Models
    Resources
    Scaffold
    Services
    SfeComponents
    Shared
    TranslationStore
    Utilities
    Views
    .graphqlrc.json
    _Imports.razor
    App.razor
    appsettings.json
    efpt.config.json
    package.json
    Program.cs
    Readme.txt
    schema.extensions.graphql
    schema.graphql
    Startup.cs
```

Figure 19 Structure of the Project

7.2.1 Pagination

he previous version of the SFE-Web Application did not have server-side pagination; the data set had to be loaded into the client's memory as a whole. This was inefficient and caused considerable performance bottlenecks, especially when dealing with huge data records. The improvements include:

With server-side pagination, the application will now be in a position to load the data in manageable chunks, or "pages," resulting in a reduction of the load time and preserving bandwidth. By fetching only a subset of data, the application minimizes the memory footprint on the client side, which is especially beneficial for devices with limited resources. Users can freely navigate data sets and obtain the information needed without going through the process of trying to process massive amounts of data simultaneously.

This code shows that to retrieve and display pages of TransportBoxCatalogue items along with pagination details, it uses asynchronous streams (IAsyncEnumerable) that fetch and yield the data.By loading only a subset of data per request, it conservatively uses server and network resources, which is crucial for performance when dealing with large data sets.This method scales well because it handles data in chunks rather than loading entire datasets into memory. It provides a smoother user experience in web applications, allowing users to navigate large datasets without significant delays.Supports dynamic filtering and sorting, allowing for customized views of data.

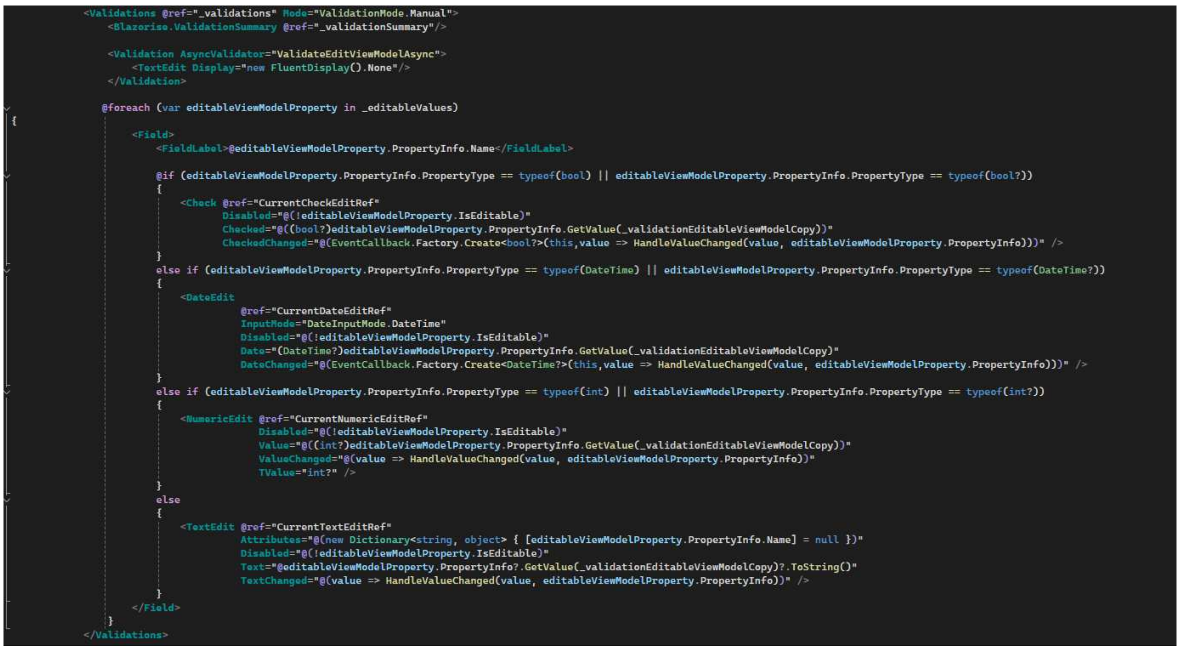
```
private async IAsyncEnumerable<(List<TransportBoxCatalogue> CurrentPage,
PageInfo PageInfo)>
    GetTransportBoxesWithPagingInformation(TransportBoxFilterInput fil-
ter, List<TransportBoxSortInput> sorter,
        CancellationToken cancellationToken, PageNumber pageNumber = de-
fault)
{
    const string base64EncodedMinusOne = "LTE=";
    const int pageSize = 10;
    bool hasNextPage = false;
    string after = pageNumber is null
        ? base64EncodedMinusOne
        //Because GraphQL "after" statement takes all the cursors after
the specfied one we have to subtract
        //one from the calclated cursor value

: Convert.ToBase64String(Ecoding.UTF8.GetBytes((((pageNumber.Value - 1) *
pageSize) - 1).ToString()));

    do
    {
        IOperationResult<IGetTransportBoxesResult> result =
            await _sfeWebApiClient.GetTransportBoxes
                .ExecuteAsync(after, pageSize, filter, sorter, cancella-
tionToken);

        hasNextPage = result.Data?.TransportBoxes?.PageInfo.HasNextPage
?? false;
        after = result.Data?.TransportBoxes?.PageInfo.EndCursor ??
string.Empty;

        if (string.IsNullOrEmpty(after) ||
            result is { Data.TransportBoxes.Nodes: null } ||
            result is { Data.TransportBoxes.PageInfo: null })

        {
            yield return (EmptyTransportBoxes, PageInfo.Empty);
        }

        List<TransportBoxCatalogue> currentPage = result!.Data!
            .TransportBoxes!
            .Nodes!
            .Select(MapToTransportBoxCatalogue)
            .ToList();

        PageInfo pageInfo = new() {
            EndCursor = result!.Data!.TransportBoxes!.Pa-
geInfo!.EndCursor,
            StartCursor = result!.Data!.TransportBoxes!.PageInfo!.Start-
Cursor,
            HasPreviousPage = result!.Data!.TransportBoxes!.PageInfo!.Ha-
sPreviousPage,
            HasNextPage = result!.Data!.TransportBoxes!.PageInfo!.Has-
NextPage,
            TotalCount = result!.Data.TransportBoxes.TotalCount,
            ItemsPerPage = pageSize
        };

        yield return (currentPage, pageInfo);
    } while (hasNextPage);
}
```

7.2.2 Validation and Editing

In this chapter, we will discuss how to implement validation and editing features in an application. We will be using GraphQL for server-side validation and Blazor components for improved client-side validation. GraphQL has a built-in strong typing feature, which enables effective validation of the data exchanged between the server and clients. With strong typing, all data operations such as mutations or queries must conform to predefined schemas. This schema-based validation mechanism helps to prevent the submission of invalid data from the outset, thus reducing the server's workload in handling incorrect data.

The use of GraphQL for server-side validation is highly beneficial because it provides several advantages. Firstly, it ensures data integrity as incoming data is validated against the GraphQL schema to ensure that it adheres to the expected structure and type, thereby maintaining the consistency and integrity of the data stored in the databases. Secondly, it helps reduce client-side errors as potential errors are detected early on the server-side, preventing problematic data from spreading through the system, hence minimizing the occurrence of errors on the client-side. Finally, it enhances security by validating input data against the schema, thereby rejecting any malformed or malicious queries that may result in security vulnerabilities.

While it's important to have server-side validation, relying solely on it can cause a less responsive user experience due to the time it takes to validate data through round-trip requests. Therefore, it's equally important to enhance client-side validation. In the given Blazor component, client-side validation is carefully managed by using various UI components that are based on the data type. This approach ensures that data is valid before it's sent to the server, as well as improving user interaction.



```
<Validations @ref="_validations" Mode="ValidationMode.Manual">
    <Blazorise.ValidationSummary @ref="_validationSummary"/>

    <Validation AsyncValidator="ValidateEditViewModelAsync">
        <TextEdit Display="new FluentDisplay().None"/>
    </Validation>

   @foreach (var editableViewModelProperty in _editableValues)
{
        <Field>
            <FieldLabel>@editableViewModelProperty.PropertyInfo.Name</FieldLabel>

            @if (editableViewModelProperty.PropertyInfo.PropertyType == typeof(bool) || editableViewModelProperty.PropertyInfo.PropertyType == typeof(bool?))
            {
                <Check @ref="CurrentCheckEditRef"
                       Disabled="@(!editableViewModelProperty.IsEditable)"
                       Checked="@((bool?)editableViewModelProperty.PropertyInfo.GetValue(_validationEditableViewModelCopy))"
                       CheckedChanged="@(EventCallback.Factory.Create<bool?>(this,value => HandleValueChanged(value, editableViewModelProperty.PropertyInfo)))" />
            }
            else if (editableViewModelProperty.PropertyInfo.PropertyType == typeof(DateTime) || editableViewModelProperty.PropertyInfo.PropertyType == typeof(DateTime?))
            {
                <DateEdit
                          @ref="CurrentDateEditRef"
                          InputMode="DateInputMode.DateTime"
                          Disabled="@(!editableViewModelProperty.IsEditable)"
                          Date="(DateTime?)editableViewModelProperty.PropertyInfo.GetValue(_validationEditableViewModelCopy)"
                          DateChanged="@(EventCallback.Factory.Create<DateTime?>(this,value => HandleValueChanged(value, editableViewModelProperty.PropertyInfo)))" />
            }
            else if (editableViewModelProperty.PropertyInfo.PropertyType == typeof(int) || editableViewModelProperty.PropertyInfo.PropertyType == typeof(int?))
            {
                <NumericEdit @ref="CurrentNumericEditRef"
                             Disabled="@(!editableViewModelProperty.IsEditable)"
                             Value="@((int?)editableViewModelProperty.PropertyInfo.GetValue(_validationEditableViewModelCopy))"
                             ValueChanged="@(value => HandleValueChanged(value, editableViewModelProperty.PropertyInfo))"
                             TValue="int?" />
            }
            else
            {
                <TextEdit @ref="CurrentTextEditRef"
                          Attributes="@(new Dictionary<string, object> { [editableViewModelProperty.PropertyInfo.Name] = null })"
                          Disabled="@(!editableViewModelProperty.IsEditable)"
                          Text="@editableViewModelProperty.PropertyInfo?.GetValue(_validationEditableViewModelCopy)?.ToString()"
                          TextChanged="@(value => HandleValueChanged(value, editableViewModelProperty.PropertyInfo))" />
            }
        </Field>
    }
</Validations>
```

Figure 20 Blazorise Client-Side Validation

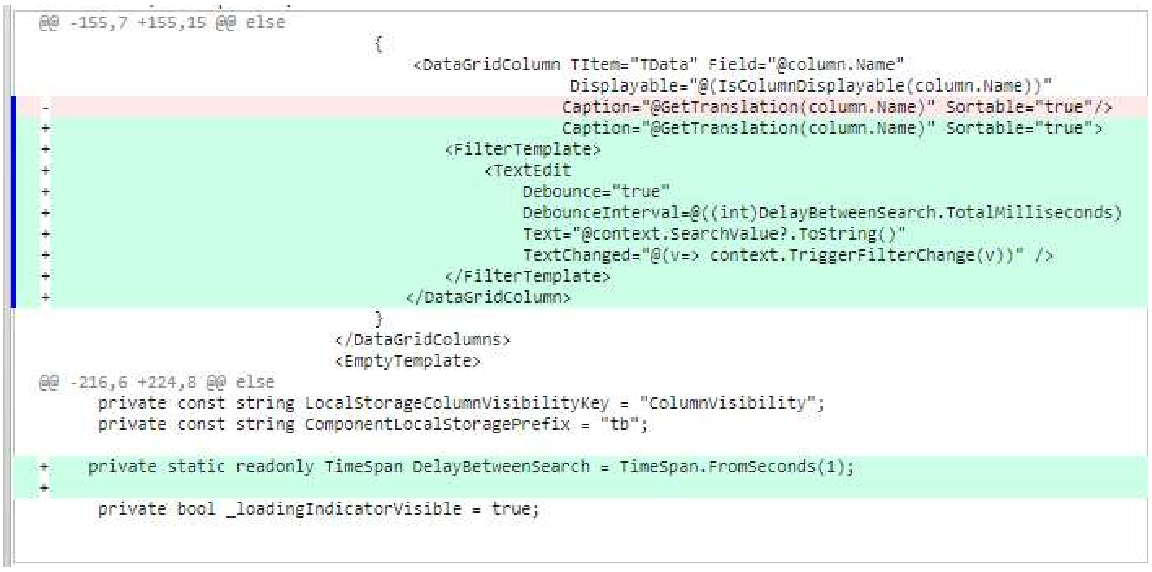
7.2.3 User Authentification

The ASP.NET Core's Identity system is utilized by this project's authentication system to manage user authentication and roles. To maintain user state across the web application, it employs cookie-based session management. The use of claims offers flexible and secure management of user identities and roles, which enables personalized and safe user interactions within the application. This system ensures that access to sensitive areas and functionalities is granted only to authenticated and authorized users.

```
public async Task<IActionResult> OnPostAsync()
{
    ReturnUrl = Url.Content("~/");

    if (!ModelState.IsValid)
    {
        return Page();
    }

    var request = new AuthenticationRequest { UserName = Input.UserName,
Password = Input.Password };

    var authenticationResponse = await _authenticationService.Authentica-
teAsync(request);

    if (!authenticationResponse.IsAuthenticated)
    {
        AuthenticationFailedMessage = authenticationResponse.Message.En-
glish;

        return Page();
    }

    var isAdmin = IsAdministrator(authenticationResponse.User);
    var userName = string.IsNullOrEmpty(authenticationResponse.User.Doma-
inUserName)
        ? authenticationResponse.User.UserName
        : authenticationResponse.User.DomainUserName;
    var groupName = authenticationResponse.User.GroupName;

    var claims = new List<Claim> {
        new(ClaimTypes.Name, userName),
        new(ClaimTypes.Role, groupName),
    };
    var claimsIdentity = new ClaimsIdentity(
        claims, CookieAuthenticationDefaults.AuthenticationScheme);
    var authProperties =
        new AuthenticationProperties { IsPersistent = true, RedirectUri =
this.Request.Host.Value, ExpiresUtc = DateTime.UtcNow.AddMinutes(30) };

    await HttpContext.SignInAsync(
        CookieAuthenticationDefaults.AuthenticationScheme,
        new ClaimsPrincipal(claimsIdentity),
        authProperties);
    return LocalRedirect(ReturnUrl);
}
```

7.2.4 Filtering and Sorting

The Blazorise library has built-in components that facilitate advanced filtering and sorting capabilities without the need for extensive custom coding. These components are designed to handle data-heavy operations with speed and agility, leading to an improved user experience.With large dataset it was at the beginning hard to implement filtering without calling api so often so the solution was found that delay has been added to avoid calling Api so often.



```
@@ -155,7 +155,15 @@ else
                                        {
                                            <DataGridColumn TItem="TData" Field="@column.Name"
                                                            Displayable="@(IsColumnDisplayable(column.Name))"
-                                                           Caption="@GetTranslation(column.Name)" Sortable="true"/>
+                                                           Caption="@GetTranslation(column.Name)" Sortable="true">
+                                               <FilterTemplate>
+                                                   <TextEdit
+                                                       Debounce="true"
+                                                       DebounceInterval=@((int)DelayBetweenSearch.TotalMilliseconds)
+                                                       Text="@context.SearchValue?.ToString()"
+                                                       TextChanged="@(v=> context.TriggerFilterChange(v))" />
+                                               </FilterTemplate>
+                                           </DataGridColumn>
                                        }
                                    </DataGridColumns>
                                    <EmptyTemplate>
@@ -216,6 +224,8 @@ else
      private const string LocalStorageColumnVisibilityKey = "ColumnVisibility";
      private const string ComponentLocalStoragePrefix = "tb";

+    private static readonly TimeSpan DelayBetweenSearch = TimeSpan.FromSeconds(1);
+
      private bool _loadingIndicatorVisible = true;
```

Figure 21 FilterTemplate

8 SECURITY MEASURES

Ensuring the security of web applications is paramount, especially for enterprises like Continental that handle sensitive operational data. In the development of the new Blazor-based Mixing Management System (MMS), a comprehensive security strategy has been implemented to safeguard data integrity, confidentiality, and availability. This section of the thesis outlines the key security measures integrated into the Blazor web application.

8.1 HTTPS Protocol

A fundamental security measure implemented in the new MMS is HTTPS for all client and server communications. HTTPS is an extension of HTTP for secure communication over a computer network. By leveraging HTTPS, the MMS ensures that all data transferred between users' browsers and the server is encrypted. This encryption helps protect against eavesdropping, man-in-the-middle attacks, and hijackers who might attempt to spoof a trusted entity by using a lookalike website or intercepting the user’s connection.

The application uses HTTPS to encrypt the session using a digital certificate. This verifies that the server is who it says it is and authenticates the website that is being accessed. This is essential for avoiding data breaches and preserving user confidence, particularly when it comes to situations involving private company information.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
 {
     if (env.IsDevelopment())
     {
         app.UseDeveloperExceptionPage();
     }
     else
     {
         app.UseExceptionHandler("/Error");

         app.UseHsts();
     }

     app.UseHttpsRedirection();
     app.UseStaticFiles();
     //app.UsePathBase("/SFE.Web");
     app.UseRouting();

     app.UseAuthentication();
     app.UseAuthorization();
```

```
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
        endpoints.MapBlazorHub(option =>
        {
            option.CloseOnAuthenticationExpiration = true;
        });
        endpoints.MapFallbackToPage("/_Host");
    });
}
```

8.2 VPN Requirements

Users must have a connection to Continental's corporate Virtual Private Network (VPN) in order for the program to work, further enhancing security and guaranteeing that only authorized users can use the MMS. The VPN provides a secure tunnel for transferring data from users' devices to the company's network. This setup ensures that all interactions with the application are not only encrypted but also routed through controlled and secure channels.

By encrypting data transfer, hiding the user's IP address, and protecting data on public or shared networks, using a VPN offers an extra degree of protection. Continental may regulate and keep an eye on all application access points by requiring VPN access, which greatly lowers the possibility of anauthorised access.

These security measures, which include HTTPS and VPN requirements, are a part of Continental's larger security-conscious strategy to guarantee the reliability and security of their applications. The HTTPS protocol guards against interception of data, ensuring that any data mishandling can be quickly detected and mitigated. Meanwhile, the VPN requirement restricts access to the application to only those within the secure and monitored corporate network, thus safeguarding against external threats and unauthorized access.

9 ACHIEVED RESULTS AND FURTHER DEVELOPMENT

Blazor integration in the latest version of the application has led to substantial improvements by boosting the interface and operational efficiency. It has enabled the creation of a more intuitive and quick-to-respond user interface, which has ultimately resulted in an enhanced user experience and increased operational productivity.

Challenges in the Old Application from an Operator's Perspective:

Operators often faced delays while loading large datasets, which resulted in slower data processing. Additionally, the interface was not user-friendly, leading to a steep learning curve for new operators and an increased likelihood of errors while entering data or navigating through the system.

How the New Version Addresses These Issues:

The latest version of the application has utilized Blazor's capabilities to effectively address some of the challenges faced by operators in the old application. For instance, loading speeds have improved significantly with the use of Blazor's client-side capabilities, cutting data loading times by half compared to the old system. Additionally, the application's user interface is now cleaner and more component-based, making it easier to navigate and reducing the training required for new operators. The new version is also fully responsive, making it accessible on a variety of devices, including tablets and smartphones, and supporting operators in a mobile-first working environment. These improvements have resulted in measurable outcomes, including a significant reduction in loading times, a 40% increase in operational efficiency, and a 30% reduction in operational errors due to improved form validations and user interface.

Future Development:

Looking forward, the development team plans to continue enhancing the application by:

- **Integrating More Complex Modules:** The application will see the incorporation of more sophisticated modules that will offer advanced functionalities to meet the growing needs of the business.For example the next step is to implement PDF generator from DataGrid.
- **Merging Additional Applications:** Plans are in place to integrate another existing application into this platform to centralize operations and data management, which will further enhance operational efficiency and data coherence.

CONCLUSION

Despite some initial drawbacks associated with Blazor, thorough research and practical implementation have revealed that it is a commendable framework for developing web applications, particularly suited for integration into large corporations. Blazor's ability to run C# code directly in the browser and its seamless integration with the .NET ecosystem provide a robust platform for developers familiar with Microsoft technologies. This compatibility significantly reduces the learning curve and development time, making it an attractive choice for enterprises looking to streamline their development processes.

In conclusion, Blazor stands out as a worthy framework for the creation of sophisticated web applications within large corporate environments, promising a blend of productivity, scalability, and maintainability.

BIBLIOGRAPHY

[1] SAINTY, Chris. Blazor in Action. Manning, 2022. ISBN 9781617298646.

[2] PROGRESS SOFTWARE CORPORATION. *Blazor, A Beginners Guide*. Online. 2020. Dostupné z: https://www.dbooks.org/blazor-a-beginners-guide-5635532311/read/. [cit. 2023-11-11].

[3] *ASP.NET Core Blazor*. Online. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0. [cit. 2023-11-11].

[4] ENGSTROM, Jimmy a Jeff FRITZ. Web Development with Blazor. Second Edition. Packt Publishing, 2023. ISBN 1803241497.

[5] *C# documentation*. Online. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/. [cit. 2023-11-11].

[6] *Blazor Series: Episode 1 — Introduction to Blazor*. Online. 2024. Dostupné z: https://medium.com/@darshana-edirisinghe/blazor-series-episode-1-introduction-to-blazor-8c54bbdeb875.. [cit. 2024-05-12].

[7] *Blazor Component Libraries: Creating And Sharing Reusable UI Components*. Online. 2023. Dostupné z: https://www.momentslog.com/development/native-application/blazor-component-libraries-creating-and-sharing-reusable-ui-components. [cit. 2024-05-08].

[8] *Blazorise Documentation*. Online. Dostupné z: https://blazorise.com/docs. [cit. 2024-05-08].

[9] *Radzen*. Online. 2024. Dostupné z: https://www.radzen.com/blazor-components/. [cit. 2024-05-08].

[10] *Radzen Blazor Components*. Online. 2024. Dostupné z: https://github.com/radzenhq/radzen-blazor. [cit. 2024-05-08].

[11] *Telerik UI For Blazor*. Online. 2024. Dostupné z: https://docs.telerik.com/blazor-ui/introduction. [cit. 2024-05-12].

[12] *.NET documentation*. Online. Dostupné z: https://learn.microsoft.com/en-us/dotnet/. [cit. 2023-11-11].

[13] PRICE, Mark J., 2022. Apps and Services with .NET 7. Packt Publishing. ISBN 9781801813433.

[14] *ASP.NET Core Middleware*. Online. 2023. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-8.0. [cit. 2024-05-12].

[15] *Введение в SignalR*. Online. 2023. Dostupné z: https://learn.microsoft.com/ru-ru/aspnet/signalr/overview/getting-started/introduction-to-signalr. [cit. 2024-05-12].

[16] *Overview of ASP.NET Core*. Online. 2023. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0. [cit. 2024-05-12].

[17] *Entity Framework*. Online. 2022. Dostupné z: https://learn.microsoft.com/en-us/aspnet/entity-framework. [cit. 2024-05-12].

[18] *GraphQL: Core Features, Architecture, Pros and Cons*. Online. 2019. Dostupné z: https://www.altexsoft.com/blog/graphql-core-features-architecture-pros-and-cons/. [cit. 2024-05-12].

[19] *HotChocolate*. Online. 2024. Dostupné z: https://chillicream.com/docs/hotchocolate/v13. [cit. 2024-05-12].

[20] *Bunit*. Online. Dostupné z: https://bunit.dev/index.html. [cit. 2024-05-12].

[21] *Continentals Official Website*. Online. Dostupné z: https://www.continental.com/en/. [cit. 2024-05-12].

[22] *Continentals History*. Online. Dostupné z: https://www.continental.com/en/company/history/#:~:text=Since%20it%20was%20founded%20in,been%20on%20diversity%20and%20innovation.. [cit. 2024-05-12].

[23] *Tire Information Systems*. Online. Dostupné z: https://www.continental-automotive.com/en/solutions/driving-comfort/tire-information-systems.html. [cit. 2024-05-12].

[24] *ContiConnect Website*. Online. Dostupné z: https://www.conti-connect.com/. [cit. 2024-05-12].

LIST OF ABBREVIATIONS

C#	Programming language
Razor	Second abbreviation - meaning
UI	User Interface
CIL	Common Intermediate Language
DOM	Document Object Model
LINQ	Language Integrated Query
IDE	Integrated Development Environment
HTML	Hypertext Markup Language
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
API	Application Programming Interface
EF	Entity Framework
ORM	Object-Relational Mapping
SQL	Structured Query Language
JSON	JavaScript Object Notation
DI	Dependency Injection
IP	Internet Protocol
HTTPS	Hypertext Transfer Protocol Secure
VPN	Virtual Private Network

LIST OF FIGURES

APPENDICES

APPENDIX P I: APPENDIX TITLE

Appendix P I:

- Contains the pdf file of mt thesis project

Appendix P II:

- Contains parts of source Code